КОНСТРУИРОВАНИЕ И ОПТИМИЗАЦИЯ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ

Серия
“КОНСТРУИРОВАНИЕ
И ОПТИМИЗАЦИЯ ПРОГРАММ”

Под редакцией
доктора физ.-мат. наук, профессора, чл.-корр. РАЕН
В. Н. Касьянова

Выпуски серии:

1. Смешанные вычисления и преобразование программ (1991)
2. Конструирование и оптимизация программ (1993)
3. Интеллектуализация и качество программного обеспечения (1994)
4. Проблемы конструирования эффективных и надежных программ (1995)
5. Оптимизирующая трансляция и конструирование программ (1997)
6. Проблемы систем информатики и программирования (1999)
7. Поддержка супервычислений и Интернет-ориентированные технологии (2001)
8. Касьянов В. Н., Мирзуитова И. Л. Slicing: срезы программ и их использование (2002)
9. Современные проблемы конструирования программ (2002)
10. Новые информационные технологии в науке и образовании (2003)
11. Программные средства и математические основы информатики (2004)
12. Методы и инструменты конструирования и оптимизации программ (2005)
13. Проблемы интеллектуализации и качества систем информатики (2006)
14. Касьянова Е. В. Адаптивные методы и средства поддержки дистанционного обучения программированию (2007)
15. Методы и инструменты конструирования программ (2007)
16. ***Конструирование и оптимизация параллельных программ***

Российская академия наук
Сибирское отделение
Институт систем информатики
имени А. П. Ершова

КОНСТРУИРОВАНИЕ И ОПТИМИЗАЦИЯ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ

Под редакцией
проф. Виктора Николаевича Касьянова

Новосибирск 2008

УДК 519.68; 681.3.06
ББК З 22.183.49+ З 22.174.2

Конструирование и оптимизация параллельных программ. — Новосибирск: Ин-т систем информатики имени А. П. Ершова СО РАН, 2008. — 332 с.

Сборник является шестнадцатым в серии, издаваемой Институтом систем информатики СО РАН по проблемам конструирования и оптимизации программ. Посвящен проблемам, касающимся методов и инструментов конструирования и оптимизации параллельных программ и систем.

Представляет интерес для системных программистов, студентов и аспирантов, специализирующихся в области системного и теоретического программирования, и для всех тех, кто интересуется проблемами современной информатики и программирования.

Siberian Division of the Russian Academy of Sciences
A. P. Ershov Institute of Informatics Systems

PARALLEL PROGRAMS CONSTRUCTION AND OPTIMIZATION

Edited by
prof. V. N. Kasyanov

Novosibirsk 2008

This volume is the sixteenth in the series on the problems of program construction and optimization, published by the A.P. Ershov Institute of Informatics Systems. The volume is devoted to problems dealing with methods and tools of parallel system and program construction and optimization.

It may be of interest to system programmers, students and postrgraduates specializing in the area of system and theoretical programming, as well as all those interested in modern informatics and programming.

ПРЕДИСЛОВИЕ РЕДАКТОРА

Шестнадцатый выпуск серии «Конструирование и оптимизация программ» посвящен решению актуальных проблем конструирования и оптимизации параллельных программ.

Продолжая уже сложившиеся традиции, данный выпуск, как и предыдущие, базируется на результатах исследований, ведущихся в лаборатории по конструированию и оптимизации программ Института систем информатики имени А. П. Ершова СО РАН совместно с кафедрой программирования Новосибирского государственного университета. Основная часть данного сборника написана по результатам второго года работ по проекту «Разработка и реализация интегрированной визуальной среды конструирования и оптимизации параллельных программ», выполняемого сотрудниками лаборатории при финансовой поддержке Российского фонда фундаментальных исследований (грант РФФИ 07-07-12050).

Создаваемая в рамках этого проекта система функционального программирования SFP предназначена для поддержки разработки высококачественного переносимого программного обеспечения для параллельных вычислителей на недорогих персональных компьютерах. Речь идет о создании системы программирования нового поколения, аналогов которых пока нет и которых так не хватает у нас в стране сейчас, когда число супервычислителей все еще мало, но каждый из них можно сделать доступным по сети практически для каждого пользователя. В рамках создаваемой расширяемой интегрированной визуальной системы функционального программирования SFP прикладной программист должен будет иметь возможность на своем рабочем месте, с одной стороны, создавать и отлаживать программу без учета целевого вычислителя, а с другой - производить настройку отлаженной программы на ту или другую целевую параллельную (распределенную) архитектуру с целью достижения высокой эффективности исполнения разработанной программы. Такие возможности делают супервычислители, включенные в сеть, более доступными для использования широкому кругу прикладных программистов, а также позволяют упростить работу прикладным программистам и повысить эффективность использования ими супервычислителей за счет переноса работ по конструированию и отладке программ с дорогих супервычислителей на более дешевые и привычные персональные компьютеры и снятия необходимости выполнять построение и

отладку программы для решения одной и той же задачи каждый раз заново при переходе с одного супервычислителя на другой.

Работа над данным выпуском совпала с 50-летием со дня образования Отдела программирования, и поэтому мы решили включить в сборник статьи В. Н. Касьянова «Всемирные компьютерные конгрессы ИФИП» и Л. С. Мельникова «О семинаре Зыкова в Новосибирске».

История появления этого коллектива начинается в 1957 г., когда Сергей Львович Соболев, один из основателей Сибирского отделения Академии наук СССР и первый директор Института математики СО АН СССР, предложил Андрею Петровичу Ершову, ученику Алексея Андреевича Ляпунова, образовать и возглавить Отдел программирования в своем Институте. А. П. Ершов собрал команду выпускников лучших советских вузов. Первым большим проектом Отдела стало создание системы Альфа – оптимизирующего транслятора с языка Альфа, являвшегося расширением Алгола 60, для ЭВМ М-20. Этот транслятор был признан одним из лучших в своем классе и широко использовался для решения научных и технических задач не только в СО АН СССР, но и во всей стране. Далее силами этого коллектива были созданы системы Алгибр, Эпсилон, Сигма, транслятор Альфа-6 для ЭВМ БЭСМ-6. Среди достижений Отдела следует упомянуть создание первой в СССР системы разделения времени АИСТ-0, многоязыковой системы Бета, исследовательские проекты в области искусственного интеллекта и параллельного программирования, системы автоматизированной обработки текстов, издательские системы и многое другое. В 1964 г., после образования Вычислительного центра СО АН СССР, Отдел программирования вошел в его состав. С течением времени расширялся круг задач, стоящих перед коллективом, менялась его организационная структура, появлялись новые направления исследований, такие, как школьная информатика, смешанные вычисления. В 1990 г. был создан Институт систем информатики, носящий имя Андрея Петровича Ершова и по праву считающийся наследником и продолжателем лучших традиций Отдела программирования. Первым директором института стал Вадим Евгеньевич Котов, ученик А. П. Ершова, в дальнейшем чл.-корр. АН СССР.

С первых дней А.П. Ершов уделял огромное внимание воспитанию кадров, начиная со школьного возраста. В Отделе проходили практику студенты НГУ, одни становились его сотрудниками, другие работали в институтах СО АН, во многих городах страны. В аспирантуре и докторантуре учились специалисты из Кишинева, Таллинна, Киева, других городов страны. Личность А. П. Ершова, его идеи оказали огромное влияние на развитие программирования в нашей стране. Тесные научные и дружеские связи со-

единяли Отдел программирования с ведущими программистскими коллективами нашей страны, с коллегами из США, Франции, Польши, Чехословакии, Германии и других стран. Основополагающую роль в этом сотрудничестве сыграло активное участие А. П. Ершова в работе Международной федерации по обработке информации (ИФИП) – наиболее представительной и влиятельной международной организации по компьютерным наукам. Андрей Петрович не только участвовал во всех семи конгрессах ИФИП, которые состоялись в период с 1965 г. по 1983 г., но и много сил тратил на их организацию. Помимо этого, А. П. Ершов активно участвовал в деятельности Рабочей группы 2.1 по Алголу Технического комитета 2 ИФИП по программированию (ТК 2) и в работе самого ТК 2, а также в организации рабочих конференций, проводимых под эгидой ТК 2 ИФИП. В 1980 г. за плодотворную деятельность в ИФИП по организации конгрессов А. П. Ершов был награжден «Серебряным сердечником» (Silver Core) – одним из высших знаков отличия для членов ИФИП. Статья В. Н. Касьянова – это рассказ о ИФИП и проводимых ею Всемирных компьютерных конгрессах.

Программирование, по словам Ершова, – это новый вид универсальной деятельности, при которой человек должен вложить в ЭВМ все, что видит, слышит, знает, и научить ее всему, что делает сам. Важнейшим свойством информационной модели или управляющей системы является ее структура, или, говоря математическим языком, совокупность бинарных отношений на наборах элементарных единиц данных и действий. Эти структуры данных и структуры действий являются единственными ипостасями программ и обрабатываемой ими информации, в которых они могут существовать в воображении программиста во чреве компьютера. Вот почему, утверждал Ершов, графы являются основной конструкцией для программиста. Он считал, что графы обладают огромной, неисчерпаемой изобразительной силой, соразмерной масштабу задачи программирования, и говорил, что программисту о графах нужно много знать, при этом с большим запасом по отношению к любой конкретной задаче. Поэтому не случайно, что в отличие от Москвы, где, начиная с работ Янова, в большей степени развивался логический подход к программированию, или Киева, где в работах Глушкова и его учеников явно прослеживается приоритет алгебраических методов, Новосибирске стал центром применения графовых моделей и методов в программировании. Статья Л. С. Мельникова – это рассказ о семинаре по теории графов Института математики СО АН шестидесятых годов прошлого столетия.

Проф. В. Н. Касьянов

Р. Н. Арапбаев, А. П. Стасенко

ИНДЕКСНЫЙ АНАЛИЗ ЗАВИСИМОСТЕЙ ПО ДАННЫМ В SISAL-ПРОГРАММАХ

1. ВВЕДЕНИЕ

Функциональные языки и языки однократного присваивания, каковым является Sisal, способствуют разработке корректных детерминированных параллельных программ. Функциональные программы свободны от совмещения имен, сторонних эффектов и ошибок, зависящих от времени. Результаты детерминированы, невзирая на архитектуру, операционную систему или исполняемое окружение. В отличие от императивных языков функциональные языки уменьшают нагрузку на программирование. Пользователи могут определить, что может быть вычислено, и могут только кодировать зависимости по данным между операциями.

Компилятор ответственен за планирование операций, передачу значений данных, синхронизирующих операций, управление памятью. Легко написать функциональную программу, которая является, безусловно, параллельной, так как ее можно писать как последовательные императивные программы. Освобожденный от большинства сложностей параллельного программирования пользователь получает больше возможностей сконцентрироваться на конструкции алгоритмов и разработки прикладных программ [1].

В Лаборатории конструирования и оптимизации программ ИСИ СО РАН, в рамках проекта ПРОГРЕСС [2], создается система функционального программирования SFP [3], где в качестве начальной версии входного языка выбран Sisal 3.2 [4]. В системе SFP Sisal-программы преобразуются в специально разработанные внутренние графовые представления [5, 6], на которых проводятся различные оптимизирующие преобразования [7], производится интерпретация программ, и которые транслируются в программы на языке Си.

Возможности любого компилятора зависят от возможностей его анализаторов. Данная работа посвящена построению (разработке) алгоритма для индексного анализа зависимости по данным в Sisal-программах.

Статья построена следующим образом. В разделе 2 даны основные определения. В разделе 3 подробно описывается построение алгоритма зави-

симости по данным в Sisal-программах. В разделе 3 приводится заключение о проделанной работе и список литературы по данной тематике.

Все понятия, не определяемые в этой работе, могут быть найдены в работах[4–6, 8].

2. ОСНОВНЫЕ ОПРЕДЕЛЕНИЯ

Традиционно под зависимостью по данным понимается зависимость, связанная с совпадением ссылок на элементы массивов. Определение зависимостей по данным представляет собой задачу целочисленного программирования [8]. Собранная при анализе зависимостей информация суммируется и представляется в компактном виде, удобном для дальнейшего использования. Чаще всего эта информация представляется в виде ориентированного графа зависимостей по данным (ГЗД). Известны также и другие способы представления: векторы расстояний и направлений, конус зависимости и др.

Для представления Sisal-программ используется представление программы в виде иерархического ациклического потокового графа (например, IF1-представления [5]). В системе SFP внутреннее представление Sisal-программ (будем называть его IR1-представлением [6]) также реализовано как набор классов языка Си++, реализующих средства для работы с IF1-графами.

2.1. Промежуточное представление IR1

Промежуточное представление IR1 [6], основывающееся на ациклических иерархических орграфах, было разработано для описания модели вычислений языка Sisal 3.0. При разработке языка IR1 основной задачей являлось создание графового языка, который мог бы стать результатом трансляции (front-end) компиляторов нескольких функциональных языков. Это иерархическая граф-модель, явно описывающая зависимости по данным на самом нижнем уровне и неявно – между так называемыми «составными вершинами», представляющими управляющие структуры, и их подграфами. С помощью такого графового языка становится возможным объединение и единые преобразования для программ, написанных с помощью разных функциональных языков.

Программа на языке IR1 состоит из множества графов специального вида, среди которых выделено подмножество графов, соответствующих множеству функций исходной Sisal программы. Граф в IR1 состоит из объектов

трёх видов: вершин, дуг и рамки графа, причём вершинам и рамке графа приписаны упорядоченные множества входов (в которые входят дуги, но не более одной для одного входа) и выходов (из которых выходят дуги). Входы рамки графа рассматриваются как выходы (результаты вычислений) графа, а выходы рамки графа как входы (параметры) графа. Тем самым, рамку графа можно рассматривать как «вывернутую наизнанку» вершину. В дальнейшем, как и в оригинальной терминологии IR1, входы могут называться входными портами, а выходы – выходными портами.

Каждой дуге графа приписан тип пересылаемого значения, если IR1 задаёт строго типизированный язык. Существует, однако, специальный вид дуги, обозначающий литерал любого типа. Эти дуги не имеют начального выхода, но имеют дополнительное свойство, передающее непосредственное значение литерала в какой-либо входной порт.

Вершины обозначают операции над своими входами (аргументами), результаты которых находятся на выходах вершины. Вершины бывают простыми и составными. Простые вершины (или просто вершины) не имеют внутренней структуры помимо ассоциированной с ними операции. Составные вершины дополнительно содержат упорядоченное множество графов. Их количество и все связи между входами (выходами) составной вершины и входами (выходами) этих графов неявно задаются типом (или семантикой) операции составной вершины. Структурированность вычислений, задаваемых IR1 графом, основывается на иерархичности IR1 графов, заданной посредством графов составной вершины.

Тем самым, IR1 задаёт поток вычислений без какого-либо дополнительного управления и исключительных ситуаций. Поэтому предполагается наличие дополнительного, ошибочного значения для каждого используемого типа данных, возвращаемого вершинами операций, определённых не для всех значений своих аргументов (например, деление на ноль).

Очевидно, что вычисления, заданные подобным образом, определяют частичный порядок над общей последовательностью выполнения вершин-операций одного IR1 графа. Несравнимые между собой операции можно выполнить параллельно, что позволяет естественным (не зависящим от машинной архитектуры) образом задать модель параллельных вычислений, причём на очень низком уровне – уровне отдельной операции. К тому же IR1 граф допускает лёгкую интерпретируемость (исполнение), так как он задаёт потоковую модель вычислений, для исполнения которой (с некоторыми ограничениями) можно даже использовать суперкомпьютеры с потоковой архитектурой. В языке Sisal большую важность имеет уменьшение

излишнего копирования значений (особенно в случае больших массивов), возникающее при выходе нескольких дуг из одного порта.

Ниже приведен пример Sisal-функции (см. пример 1) и её внутреннего представления (см. рис.1). Отметим, что составная вершина gtForAll имеет четыре графа [6]:

1. Граф инициализации.
2. Граф генератора диапазона.
3. Граф тела цикла.
4. Граф предложения возврата.

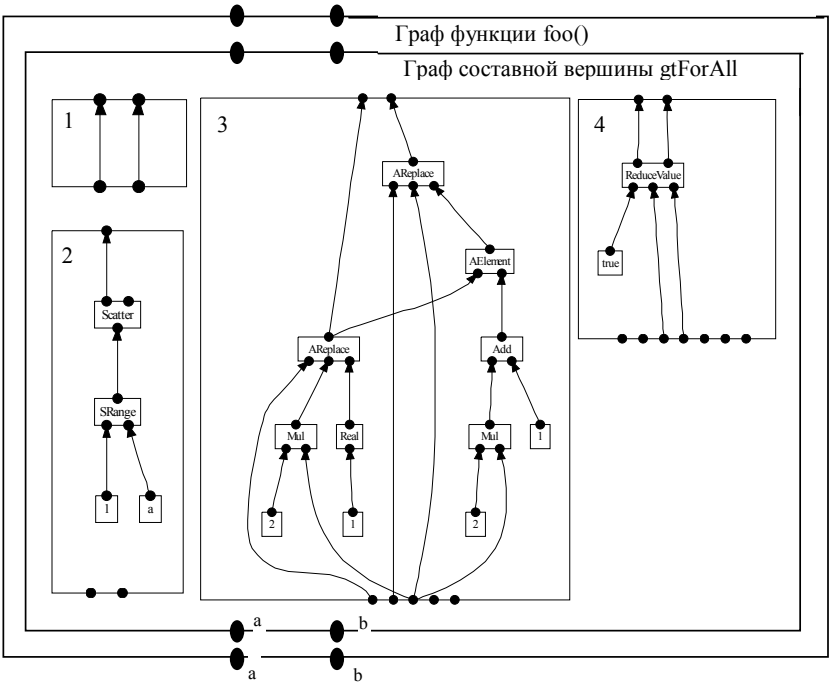


Рис. 1. IR1-граф для примера 1

В этом примере в третьем графе вершины «gtForAll» имеется дублирование массива «a» в двух дугах, исходящих из выходного порта вершины gtAReplace.

Пример 1. Sisal-функция.

```
function foo(a,b: array[real] returns array[real],array[real])
 for i in 1, 10 repeat
  a:= old a[2*i := 1];
  b:= old b[i := a[2*i+1]]
  returns value of a, b
 end for
end function
```

2.2. Определение зависимости по данным

Пусть две операции S_1 и S_2 обращаются к элементу d - мерного массива A и r – число for-циклов («выражений for» в SISAL терминологии) окружающих операции S_1 и S_2: $S_1 \equiv A[f_1(i_1, i_2, \dots, i_r), f_2(i_1, i_2, ..., i_r), \dots, f_d(i_1, i_2,\dots, i_r)]$ и $S_2 \equiv A[g_1(i_1, i_2,..., i_r), g_2(i_1, i_2,\dots, i_r), \dots, g_d(i_1, i_2,\dots, i_r)]$, где $i_1, i_2, ..., i_r$ индексные переменные циклов. Нижние границы $L_1, L_2, ..., L_r$ и верхние границы циклов $U_1, U_2, ..., U_r$ являются целочисленными константами.

Определение 1. Назовем две операции S_1 и S_2 в цикле *зависимыми*, если:

- S_1 и S_2 обращаются к общему участку памяти M;
- S_1 исполняется раньше S_2 в цикле;
- между S_1 и S_2 нет других обращений к участку памяти.

Определение 2. Под равенством $S_1(i) = S_2(j)$ понимается, что данные операции обращаются к общему участку памяти.

Таким образом, система уравнений зависимости по данным для операции S_1 и S_2 имеет вид:

$$\begin{cases} f_1(i_1, i_2, ..., i_r) = g_1(i_{r+1}, i_{r+2}, ..., i_n) \\ f_2(i_1, i_2, ..., i_r) = g_2(i_{r+1}, i_{r+2}, ..., i_n) \\ \quad\vdots \qquad\quad \vdots \qquad\quad \vdots \\ f_d(i_1, i_2, ..., i_r) = g_d(i_{r+1}, i_{r+2}, ..., i_n) \end{cases} \tag{1}$$

$$L_p \le i_p \le U_p, \text{ где } p=1,\dots,n. \tag{2}$$

Между двумя операции S_1 и S_2 имеется *зависимость по данным* тогда, и только тогда, когда существуют целочисленные решения $i_1, i_2..., i_n$ системы линейных диофантовых уравнений (1), удовлетворяющие ограничениям (2).

Следовательно, проблема зависимости данным представляет собой задачу целочисленного программирования. Если имеется *m*-мерный массив **A** и индексные выражения массива линейны, то тогда система (1) может быть записана в следующем виде:

$$\begin{aligned} &a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n + c_1 = 0 \\ &a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n + c_2 = 0 \\ &\qquad \ldots\ldots\ldots\ldots \\ &a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n + c_m = 0 \end{aligned} \qquad (3)$$

и

$$L_i \le x_i \le U_i \text{ где } i = 1, \ldots, n \qquad (4)$$

Определение 2. Будем также говорить, что индексные выражения *сцепленные* (coupled), если они включают в себя одинаковые индексные переменные цикла.

Если потенциальная зависимость включает *сцепленные* индексы, то для её разрушения необходимо одновременное рассмотрение индексов многомерного массива.

При решении этой задачи компиляторы используют тесты на зависимость по данным. Однако прямой подход к решению задачи выявления зависимостей в общем случае невозможен, так как даже для линейных индексных выражений массивов это приводит к NP-полной проблеме отыскания целочисленного решения системы диофантовых уравнений (1) и неравенств (2). Поэтому к настоящему времени разработаны многие тесты, дающие приближенные решения задачи. Среди них на практике наибольшее распространение получили НОД-тест, неравенства Банержи, λ-тест, I-тест и их различные модификации. Точные тесты используют наиболее сложные методы, например: Power-тест, Омега-тест и IR-тест.

Тесты на зависимость используют различные математические инструменты, и каждый из них имеет различную сложность и разрешающую способность. Мощные алгоритмы могут выявлять зависимости по данным с большей точностью, но обычно требуют для этого много времени. Поэтому на практике используется алгоритм зависимости по данным, который состоит из серии тестов исполняемых в определенном иерархическом порядке [9].

3. ПОСТРОЕНИЕ АЛГОРИТМА ИНДЕКСНОГО АНАЛИЗА ЗАВИСИМОСТЕЙ ПО ДАННЫМ

В данном разделе будет описано построение алгоритма индексного анализа зависимостей по данным для Sisal-программ. В этой работе применяется алгоритм индексного анализа на основе системы линейных неравенств [10]. Для формирования этой системы требуется выполнить подготовительные действия по поиску гнезд циклов и индуктивных переменных, нахождению инвариантов цикла, делинеаризации и непосредственному построению системы неравенств. На вход анализатора подается внутреннее представление IR1 анализируемой Sisal-программы. Далее подробно рассматривается каждый этап анализа зависимостей по данным.

3.1. Поиск гнезд циклов, для которых возможен индексный анализ

Первым шагом анализатора является поиск гнезд циклов с генератором диапазона (gtForAll-вершина в терминологии IR1 графа [6]), для которых возможен индексный анализ. Отметим, гнездом называется путь в дереве циклов от корня до одного из листьев, т.е. гнездом циклов является множество циклов, вложенных один в другой. В данной работе рассматриваются только «совершенное» гнездо циклов. Гнездо называется совершенным, если тело каждого следующего цикла, отличного от самого внутреннего, состоит только из следующего цикла гнезда [11].

Поиск реализуется обходом IR1 графа в глубину. Как только находится гнездо for- циклов, далее выполняется шаг подготовки данных гнезда циклов.

3.2. Поиск индексных переменных и подготовка данных гнезда циклов

Диапазонные имена или данные индексных переменных определяются в генераторе диапазона, т.е. находятся на втором графе (граф генератора диапазона) составной вершины gtForAll. При обработке графа генератора диапазона создается матрица LUI содержащая значений границ циклов.

Для правильной проверки на зависимости по данным границы индексов должны быть известными константами, а шаг индексных переменных равным 1.

3.3. Подготовка данных для анализа существования зависимости двух операций обращения к массиву в цикле

На этом этапе входными данными являются операции, для которых будет производиться анализ, и цикл, которому принадлежат данные операции и соответствующее гнездо циклов с уже подготовленными данными. Так как рассматривается только «совершенное» гнездо циклов, кандидаты для зависимости по данным пар операций находятся, только в самом внутреннем теле цикла. По терминологии IR1 это граф тела цикла (третий граф) самой внутренней составной вершины gtForAll.

В представлении IR1 индексные зависимости могут возникать между вершиной gtAReplace (операцией замещения элементов массива) и вершиной gtAElement (операцией выборки элементов массива), если они связаны следующим образом:

1) выходной порт вершины A, которая является вершиной gtAReplace, связан дугой E с первым входным портом вершины B;
2) в первые входные порты этих вершин входят дуги E_1 и E_2, начинающиеся из одного порта.

В первом случае, когда вершина B является вершиной gtAElement, независимость операций A и B позволяет переключить начало дуги E на порт, с которого начинается дуга, входящая в первый порт вершины A. Такое переключение позволяет распараллелить выполнение операций A и B. Во втором случае независимость операций gtAElement и gtAReplace позволяет избежать копирования массива по дугам E_1 и E_2.

По определению, индексная зависимость по данным в Sisal-программах возникает между операцией замещения элементов массива (gtAReplace-вершина по терминологии IR1) и операцией выборки над массивами (gtAElement-вершина), если они обращаются к одной и той же ячейке памяти.

После нахождения соответствующих операций gtAReplace и gtAElement, создается уравнение зависимости от их индексных выражений в виде матрицы Coeff.

После того, как была подготовлена информация о гнезде циклов (матрица границ, вектор номеров переменных) и информация об анализируемых операциях, требуется сделать несколько финальных преобразований. Они необходимы для правильной работы алгоритма, анализирующего операции на предмет зависимости.

Пусть система линейных неравенств, определяющих границы, задается матрицей *LUI* (с размером n, m, где n – число переменных в гнезде циклов, m – количество неравенств).

Разделение матрицы границ на матрицы верхних U и нижних L границ.

В результате выполненных выше этапов получается система уравнений зависимости вида (1) и неравенств (2). Далее выполняется шаг вызова алгоритма анализа зависимостей.

3.4. Особенности используемого алгоритма анализа зависимостей

К особенностям алгоритма можно отнести использование новой стратегии тестирования [10], для выявления зависимости по данным. Алгоритм этого метода состоит из серии эффективных и недорогостоящих тестов на зависимость.

В данной стратегии, в зависимости от значений основных параметров задачи (размерность массивов, количество вложенных циклов, значения коэффициентов индексных переменных и значения границ циклов), в первую очередь выделяют часто встречающие и легко разрешимые случаи. Соответственно каждому случаю применяется один быстрый и точный тест или серия эффективных тестов.

Пусть, на вход алгоритма подается гнездо циклов, в котором r – количество вложенных циклов и операции цикла обращаются к элементам d-мерного массива. Кроме того, считаются постоянными и известными значения коэффициентов индексных переменных a_{11}, a_{12} ,…, a_{mn} и значения границ циклов L_1, L_2, ..., L_n ,U_1, U_2, ...,U_n , где n=2*r и m=d. Задача нашего алгоритма выявить зависимости по данным между операцией в итерациях гнезда циклов, т.е. алгоритм должен возвращать ответ «да/нет» о существовании целочисленных решений i_1, i_2..., i_n системы линейных диофантовых уравнений (1), удовлетворяющих ограничениям (2).

Для этого сначала выделены часто встречающие и легко разрешимые случаи задачи зависимости по данным:

> r=1, d=1, т.е. внутри единственного цикла, операторы обращаются к элементам одномерного массива. В этом случае уравнение зависимости (1) выглядит так: $a_1 x_1 + a_2 x_2 = a_0$ и $L \le x_1, x_2 \le U$. Для уравнения целесообразно применить самый быстрый и точный *SIV-тест* [6].

> r>1, d=1, уравнение зависимости имеет вид: $a_1 x_1 + a_2 x_2 + \ldots + a_n x_n = a_0$, где $L_i \le x_i \le U_i$, $i=1,\ldots,n$. Этот случай несколько усложняет решение, следовательно, применяется серия одномерных тестов на зависимость: тест Банержи, I-тест и IR-тест [10]. Каждый следующий тест выполняется только в том случае, если был получен неточный от-

вет (maybe) предыдущим тестом, кроме того, после применения теста Банержи выполняется проверка коэффициентов индексных переменных для уточнения ответов теста [10].

d=2 и имеются *сцепленные индексы*. Система уравнений зависимости имеет вид:

$$a_{1,1}x_1 + a_{1,2}x_2 + \ldots + a_{1,n}x_n = a_{1,0}$$
$$a_{2,1}x_1 + a_{2,2}x_2 + \ldots + a_{2,n}x_n = a_{2,0}$$
$$\text{и } L_i \le x_i \le U_i \qquad \text{где } i=1,\ldots,n.$$

Этот случай доминирует в реальных последовательных программах, но применение обычных одномерных тестов на зависимость в этом случае бесполезно, так как имеются сцепленные индексные переменные. Поэтому применяется серия многомерных тестов: λ–тест, многомерный I-тест и модифицированный λ–тест [12]. Метод запоминания результатов предыдущих тестов и использование их для последующих тестов, оптимизирует данный случай.

В оставшихся случаях уравнение зависимости имеет вид (1) с ограничениями (2). Каждое уравнение рассматривается в отдельности, и для него последовательно применяется серия одномерных тестов: тест Банержи, I-тест и IR-тест. Этот подход дает более точный ответ, если индексные переменные *не сцеплены.* На практике доля сцепленных индексных переменных в ссылках трехмерных массивов и выше незначительна.

Учитывая все случаи, была собрана и реализована библиотека тестов на зависимость. Библиотека состоит из следующих тестов: ZIV-тест, SIV-тест, НОД-тест, Банержи-тест, I-тест, IR-тест, λ-тест, многомерный I-тест и модифицированный λ-тест. Кроме тестов на зависимость в библиотеке имеются алгоритмы для уточнения ответов теста Банержи. Все алгоритмы имеют линейную временную сложность. Из-за высокой стоимости в библиотеку не вошли точные тесты.

Проиллюстрируем сказанное выше примерами.

Рассмотрим предыдущий пример 1. В этом примере выполняются две операции замещения элементов одномерного массива «a»:

```
S1:  a:= old a[2*i:= 1];
S2:      b:= old b[i:= a[2*i+1]].
```

Уравнение зависимости имеет вид:

$2x_1 - 2x_2 = 1$, где $0 \le x_1, x_2 \le 10$.

По схеме новой стратегий к уравнению применяется *сильный SIV-тест*, так как, d=1, r=1 и a_1=a_2.

Стоит отметить, что SIV-тест [5] применяется, когда в индексном выражении массива используется одна индексная переменная. SIV формы индексных выражений делятся на две категорий: сильные и слабые.

Индексное выражение SIV для индекса i, называется *сильным*, если оно имеет вид ($ai + c_1$, $ai' + c_2$), т.е. если оно линейное и коэффициенты двух экземпляров индекса i равны и являются константами. Для *сильных* индексных выражений SIV, *расстояние зависимости* определяется следующем образом:

$$d = i' - i = \frac{c_1 - c_2}{a}$$

Зависимость существует, тогда и только тогда, когда d - целое число и $|d| \le U\text{-}L$, где U и L – верхние и нижние границы цикла соответственно. Таким образом, сильный SIV тест является точным и эффективным тестом.

В данном случае значение $|d|$ –не целое число и, следовательно, зависимости по данным не существует. Тест возвращает ответ «нет», а алгоритм стратегии останавливается и дает ответ об отсутствии зависимости. Тем самым в примере 1, используя информацию о независимости, как показано в разделе 3.3, можно исключить излишнее копирование массива «a» в цикле.

Рассмотрим второй пример, где определяются зависимости по данным для обращений к элементам многомерного массива.

Пример 2.

```
function foo2(b: array[array[real]]
       returns array[array[real]])
 for i in 1, 100 cross j in 1, 100 repeat
   b := old b [3*i+2*j, 2*j := old b[i-j+6, i+j] ]
   returns value of b
 end for
end function
```

В этом примере операции обращаются к элементам двухмерного массива **B** и индексы массивов являются *сцепленными*. Уравнения зависимости представлены в виде системы:

$$\begin{cases} 3x_1 + 2x_2 - x_3 + x_4 = 6 \\ 2x_2 - x_3 - x_4 = 0 \end{cases}$$

где $1 \le x_1, x_3, x_2, x_4 \le 100$.

В данном случае по схеме стратегии принимается решение о применении серии многомерных тестов: λ-тест, многомерный I-тест и модифицированный λ-тест. Зависимость разрушается только после применения модифицированного λ-теста.

3.5. Интерпретация и использование результатов анализа в целях оптимизации

Результатом анализа является установление независимости операций в пространстве итераций гнезда циклов. Данные о зависимости используется при распараллеливании, векторизации и различных оптимизациях циклов, а также при оптимизации работы с памятью (устранение излишнего копирования значений).

4. ЗАКЛЮЧЕНИЕ

Язык Sisal по своей семантике идеально подходят для разработки алгоритмов для параллельных архитектур. Однако, чтобы гарантировать отсутствие сторонних эффектов и наличие прозрачности ссылок в аппликативной программе, операции, изменяющие значения данных, вынуждены работать с копиями этих данных. С этой точки зрения использование массивов, характерное для научных вычислений, очень дорого по расходам времени и памяти, причем эти расходы иногда перекрывают выгоды от параллельного вычисления. Поэтому необходимо решать задачу эффективного использования памяти вычислительной машины. Необходимость решения таких проблем порождает активное использование различных методов оптимизации, в том числе – анализ зависимостей по данным в Sisal-программах.

Основным результатом данной работы является практическая реализация анализатора зависимости по данным в Sisal-программ в рамках системы SFP.

СПИСОК ЛИТЕРАТУРЫ

1. **McGraw J. R.** Parallel functional programming in Sisal: fictions, facts, and future. – Livermore, CA, June 1993. – (Technical Report / Lawrence Livermore National Laboratory; UCRL-JC-114360)

2. **Kasyanov V.N., Evstigneev V.A. et.al.** The system PROGRESS as a tool for parallelzing compiler prototyping // proc. Of Eighth SIAM Conf. On Parallel Processing for scientific Computing (PPSC-97) – Minneapolis, 997. – P. 301–306.
3. **Kasyanov V. N., Stasenko A. P., Gluhankov M. P., Dortman P. A., Pyjov K. A., Sinyakov A. I.** SFP – An interactive visual environment for supporting of functional programming and supercomputing // WSEAS Transactions on Computers. – Athens: WSEAS Press, 2006. – Vol. 5, N 9. – P. 2063–2070.
4. **Касьянов В.Н., Стасенко А.П.** Язык программирования Sisal 3.2. // Методы и инструменты конструирования программ. – Новосибирск, ИСИ СО РАН, 2007. – С. 56–134.
5. **Густокашина Ю.В., Евстигнеев В.А.** IF1 – промежуточное представление Sisal-программ // Проблемы конструирования эффективных и надежных программ. – Новосибирск, 1995. – С. 70–78.
6. **Стасенко А.П.** Внутреннее представление системы функционального программирования **Sisal** 3.0. – Новосибирск, 2004. – 54 с. – (Препр. / РАН. Сиб. отд-е. ИСИ; № 110).
7. **Пыжов К.А.** Блок редукции в компиляторе 3.0. // Методы и инструменты конструирования и оптимизации программ. – Новосибирск: Институт систем информатики имени А. П. Ершова СО РАН, 2005. – С. 185–196.
8. **Евстигнеев В.А.** Анализ зависимостей: состояние проблемы // Системная информатика: Сб. науч. тр. / Ин–т систем информатики СО РАН. – Новосибирск: Наука, 2000.– Вып. 7. – С. 112–173.
9. **Евстигнеев В.А., Арапбаев Р.Н., Осмонов Р.А.** Анализ зависимостей: основные тесты на зависимость по данным // Сиб. журн. вычисл. математики / РАН. Сиб. отд-ние. – Новосибирск, 2007. – Т. 10, № 3. – С. 247–265.
10. **Арапбаев Р**.Н., Осмонов Р.А. Анализ зависимостей: новая стратегия тестирования // Труды Международной конференции. «Параллельные вычислительные технологии (ПаВТ’2007)». – Челябинск: Ид-во ЮУрГУ,2007. – Т. 2. – С. 16–27.
11. **Евстигнеев В.А., Касьянов В.И.** Оптимизирующие преобразования в распараллеливающих компиляторах // Программирование. – №6. – 1996. – С. 12–26.
12. **Арапбаев Р.Н.**, Осмонов Р.А. Анализ зависимостей по данным для многомерных массивов на базе модифицированного λ-теста // Проблемы интеллектуализации качества систем информатики. – Новосибирск, ИСИ СО РАН, 2006. – С. 7–23.

М. М. Бутовский

РАСЧЕТ ИНТЕГРАЛОВ ПОПЕРЕЧНЫХ МЕР МИНКОВСКОГО, СУММ МИНКОВСКОГО И ПОСТРОЕНИЕ ДИАГРАММЫ БЛЯШКЕ ДЛЯ ВЫПУКЛЫХ МНОГОГРАННИКОВ В ЕВКЛИДОВОМ ПРОСТРАНСТВЕ $\mathbf{R}^3$

ВВЕДЕНИЕ

Выпуклый анализ – раздел математики, в котором изучают выпуклые объекты: выпуклые множества, выпуклые функции и выпуклые экстремальные задачи. Таким образом, этот раздел имеет пересечения с геометрией (выпуклость – геометрическое понятие), анализом (функция – одно из основных понятий анализа) и теорией экстремальных задач.

В 1916 году Вильгельмом Бляшке было предложено отображение пространства трехмерных компактных выпуклых тел, снабженного метрикой Хаусдорфа, в компактную область на плоскости. Образ этого отображения на плоскости $\mathbf{R}^2$ известен теперь как диаграмма Бляшке. Часть границы этой диаграммы определяется из хорошо известных геометрических неравенств, но неравенства, которые замыкают границу, остаются неизвестными.

1. ОСНОВНЫЕ ПОНЯТИЯ

Отметим несколько основных понятий.

Множество $\boldsymbol{R}^n$ является *афинным пространством,* связанным с векторным пространством $\overrightarrow{R^n}$. Это означает, что каждой паре точек $(x, y) \in \boldsymbol{R}^n \times \boldsymbol{R}^n$ поставлен в соответствие вектор $\overrightarrow{xy} \in \overrightarrow{R^n}$. При этом должны выполняться условия: 1) для всякой точки $x \in \boldsymbol{R}^n$ и всякого вектора $\vec{a} \in \boldsymbol{R}^n$ существует единственная точка $y \in \boldsymbol{R}^n$, такая, что $\overrightarrow{xy} = \boldsymbol{a}$; 2) $\overrightarrow{xy} + \overrightarrow{yz} = \overrightarrow{xz}$.

Аффинное пространство $\boldsymbol{R}^n$, для которого соответствующее векторное пространство $\boldsymbol{R}^n$ наделено скалярным произведением, называется *евклидовым пространством.*

Пространственное точечное множество образует *выпуклое тело,* если оно:

1) ограничено;
2) замкнуто;
3) обладает свойством выпуклости, т.е. со всякой пересекающей его прямой имеет общим только один отрезок (он может, конечно, сводиться к единственной точке).

Последнее, важнейшее, требование можно заменить равносильным требованием, чтобы точечное множество вместе с любыми двумя точками содержало и весь соединяющий эти точки отрезок.

Точка выпуклого тела называется *внутренней*, если можно указать некоторый шар с центром в этой точке, целиком принадлежащий данному телу.

Точки выпуклого тела, не являющиеся внутренними, называются *граничными* точками. Граница ∂K выпуклого тела называется *выпуклой поверхностью*.

Граничная точка называется *регулярной*, если через нее проходит только одна опорная гиперплоскость.

Все точки, расстояние которых от выпуклого тела K не больше, чем ρ, образуют снова выпуклое тело K_ρ – «параллельное тело» для тела K.

Пусть K – выпуклое множество, и O – фиксированная точка в $\boldsymbol{R}^n$; рассмотрим все (n-r)-плоскости $L_{n\text{-}r[\mathrm{O}]}$, содержащие O. Пусть K'_{n-r} – ортогональная проекция K на $L_{n\text{-}r[\mathrm{O}]}$, т.е. K'_{n-r} – это выпуклое множество, состоящее из точек пересечения $L_{n\text{-}r[\mathrm{O}]}$ с r-плоскостями, перпендикулярными к $L_{n\text{-}r[\mathrm{O}]}$ и пересекающими K.

Среднее значение объемов проекций $V(K'_{n-r})$ будет равно

$$E(V(K'_{n-r})) = \frac{I_r(K)}{m(G_{n-r,r})} = \frac{O_{r-1}\dots O_1 O_0}{O_{n-1}\dots O_{n-r}} I_r(K),$$

где $m(G_{n\text{-}r,r})$ – объем многообразия Грассмана $G_{n\text{-}r,r}$,
O_i – площадь поверхности i-мерной единичной сферы и

$$I_r(K) = \int\limits_{G_{n-r,r}} V(K'_{n-r})dL_{n-r[O]} = \int\limits_{G_{r,n-r}} V(K'_{n-r})dL_{r[O]}.$$

Для полноты положим $I_0(K) = V(K)$ = объему K.

Вместо интегралов $I_r(K)$, которые связаны с математическим ожиданием объемов проекций $V(K'_{n-r})$, обычно вводят так называемые среднепоперечные меры, или интегралы поперечных мер, определенные Минковским:

$$V_{n-r}(K) = \frac{(n-r)O_{n-1}}{nO_{n-r-1}} E(V(K'_{n-r})) = \frac{(n-r)O_{r-1}\dots O_0}{nO_{n-2}\dots O_{n-r-1}} I_r(K).$$

Для полноты положим

$$V_n(K) = I_0(K) = V(K),$$

Для объема K_ρ имеет место выражение

$$V(K_\rho) = \sum_0^n \binom{n}{i} V_{n-i}(K)\rho^i,$$

которое справедливо для любого $\rho \geq 0$ и называется *формулой Штейнера* для параллельных выпуклых тел.

С интегралами поперечных мер связано множество известных геометрических неравенств. Наиболее известным их общих неравенств между смешанными объемами является неравенство Александрова–Фенхеля

$$V^2(K_1, K_2, \dots, K_n) \geq V(K_1, K_1, K_3, \dots, K_n) \cdot V(K_2, K_2, K_3, \dots, K_n),$$

где $K_1,\dots,K_n$ – непустые выпуклые компакты в $\boldsymbol{R}^n$.

Суммой Минковского двух подмножеств A и B линейного пространства V называется множество C, состоящее из сумм всевозможных векторов из A и B:

$$C = \{c \,|\, c = a + b, a \in A, b \in B\}.$$

2. АЛГОРИТМИЗАЦИЯ ВЫЧИСЛЕНИЙ

Для построения диаграммы Бляшке линейной комбинации выпуклых многогранников необходимо реализовать ряд предварительных вычислений и преобразований:

1) построение выпуклой оболочки из конечного числа точек;
2) расчет интегрально-поперечных мер для построенной оболочки;
3) расчет расстояния от точки до выпуклого многогранника;
4) нахождение объема параллельного тела;
5) построение заданных сумм Минковского;
6) заполнение диаграммы Бляшке.

Для построения выпуклой оболочки из конечного числа точек было разработано огромное количество алгоритмов. Один из них – *алгоритм Quick-Hull*, разработанный в 1996 году.

QuickHull – это довольно простой рекурсивный алгоритм. главное его преимущество состоит в том, что на случайном наборе точек он работает

гораздо быстрее других известных алгоритмов (алгоритм Грэхема, алгоритм Мелькмана, алгоритм «Разделяй-и-Властвуй», алгоритм «заворачивания подарка» и др.). Алгоритм работает с пространствами любой размерности.

Для расчета интегралов поперечных мер Минковского необходимо искать решение системы линейных уравнений. Для данной задачи также разработано множество алгоритмов. Один из наиболее простых в исполнении – *метод Гаусса*. Он состоит в исключении слагаемых системы путем ее равносильного преобразования.

Для расчета расстояния от точки до многогранника был использован *алгоритм Lin-Canny*. Данный алгоритм был предложен Ming C. Lin и John F. Canny в 1991 г. Для данного алгоритма каждый многогранник представлен в виде набора фрагментов – вершин, линий и фейсов. Также для каждого фрагмента известны соседние фрагменты и определена область Вороного.

Область Вороного – это множество точек, расположенных ближе к данному, чем к какому-либо другому фрагменту многогранника. Область Вороного формирует область пространства снаружи многогранника, «прилежащую» к ближайшему фрагменту и также является выпуклым многогранником, хотя и незамкнутым. Набор областей Вороного для всех фрагментов многогранника называется *диаграммой Вороного*.

В качестве структур данных для областей Вороного используются ячейки, в которых хранятся плоскости, ограничивающие область Вороного с ссылками на соседние ячейки (с которыми граничат плоскости). Чтобы найти фрагмент многогранника, ближайший к определенной точке, нужно найти область Вороного, которой принадлежит эта точка. Иначе говоря, если точка P лежит внутри области Вороного f_B объекта B, то f_B является ближайшим фрагментом к точке P.

Рассмотрим проверку попадания точки в область Вороного фрагмента многогранника. Существуют три основных комбинации – точка-вершина, точка-линия и точка-фейс.

1. Точка-Вершина.

Если вершина V многогранника B является ближайшим фрагментом к точке P, то точка P должна лежать внутри области Вороного вершины V, ограниченной плоскостями, перпендикулярными линиям, касающимся V.

Если же точка находится вне области Вороного и лежит с другой стороны одной из ограничивающих плоскостей, то это означает, что по крайней мере одна из соседних линий ближе к точке P, чем вершина V. В этом слу-

чае алгоритм возьмет соответствующую линию и проверит, является ли она ближайшей к точке P.

2. Точка-Линия.

Если линия E многогранника B является ближайшим фрагментом к точке P, то точка P должна лежать внутри области Вороного линии E, ограниченной четырьмя плоскостями, две из которых перпендикулярны линии E и проходит через ее концы, и две содержат линию E и параллельны нормалям прилегающих к ней фейсов.

Если точка находится вне области Вороного и лежит с другой стороны одной из ограничивающих плоскостей, проходящих через концы линии (это означает, что одна из соседних вершин ближе к точке P, чем линия E) или одной из ограничивающих плоскостей, параллельных нормалям прилегающих фейсов (это означает, что один из соседних фейсов ближе к точке P, чем линия E), то алгоритм возьмет соответствующую вершину или фейс и проверит, является ли этот фрагмент ближайшим к точке P.

3. Точка-Фейс.

Если фейс F многогранника B является ближайшим фрагментом к точке P, то точка P должна лежать внутри области Вороного фейса F, ограниченной плоскостями, перпендикулярными фейсу и проходящими через ограничивающие фейс линии.

Если точка находится вне области Вороного и лежит с другой стороны одной из ограничивающих плоскостей, это означает, что одна из соседних линий ближе к точке P, чем фейс F, и алгоритм возьмет соответствующую линию и проверит, является ли она ближайшей к точке P.

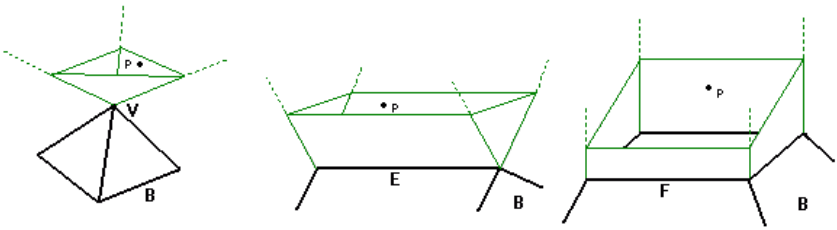


Рис. 1. Примеры попадания точки в области Вороного многогранника

Далее необходимо проверить, чтобы точка лежала над плоскостью, проходящей через фейс F, т.к. в противном случае точка может находиться внутри многогранника. В этом случае расстояние от точки до многогранника принимается равным нулю.

Для нахождения объема параллельного тела целесообразно использовать широко известный *метод Монте-Карло*.

Параллельное тело A целиком помещается в некоторый параллелепипед B с известным объемом. Затем параллелепипед B случайно заполняется большим количеством точек. Проверяем, какие из этих точек принадлежат A. Объем параллельного тела находится по формуле:

$$V(A) = \frac{N'}{N} V(B),$$

где N' – количество точек, попавших в A;
N – общее количество точек.

3. ДИАГРАММА БЛЯШКЕ

Как упоминалось ранее, объем параллельного тела A_ε для многогранника A выражается формулой Штейнера

$$V(A_\varepsilon) = \sum_0^n \binom{n}{i} V_{n-i}(A)\varepsilon^i .$$

Коэффициенты V_{n-i}, называемые интегралами поперечных мер Минковского, имеют определенный геометрический смысл. Так, для любой размерности пространства V_n есть объем выпуклого тела A, V_0 – коэффициент в формуле объема n-мерного шара (π при $n = 2$, $\frac{4}{3}\pi$ при $n = 3$ и т.д.). Следует заметить, что $\binom{n}{i}$ – коэффициенты бинома Ньютона, т.е.

$$\binom{n}{i} = C_n^i = \frac{n!}{i!(n-i)!}.$$

Следовательно, в случае $\boldsymbol{R}^2$ справедливо $V_2 = \frac{1}{2}S$, а в случае $\boldsymbol{R}^3$ имеют место равенства $V_2 = \frac{1}{3}S$ и $V_1 = \frac{1}{3}M$, где S – площадь поверхности, а M – интеграл средней кривизны.

Варьируя значения ε, мы получим различные значения объемов $V(A_\varepsilon)$. Таким образом можно записать систему из n линейных уравнений с n неиз-

вестными $(V_0, V_1, ..., V_{n-1})$ и получить численные значения интегральных поперечных мер. Стоит заметить, что эти значения неотрицательны. Это следует из определения интегрально-поперечных мер и сопутствующих ему рассуждений.

Существует ряд геометрических неравенств, описывающих интегральные поперечные меры. Это широко известные неравенства Минковского, Александрова–Фенхеля, изопериметрические неравенства и многие их следствия.

В 1916 году Вильгельмом Бляшке было предложено отображение пространства компактных выпуклых тел, принадлежащих 3-мерному евклидову пространству $\boldsymbol{R}^3$, в компактную область на плоскости. Образ этого пространства на плоскости $\boldsymbol{R}^2$ известен теперь как диаграмма Бляшке. Часть границы этой диаграммы определяется из хорошо известных геометрических неравенств, но неравенства, которые замыкают границу, остаются неизвестными. Существует предположение Бьери об экстремальных телах, соответствующих неизвестной части границы. Эту гипотезу не удается доказать или опровергнуть уже более пятидесяти лет, хотя образы предложенных тел на диаграмме могут быть найдены с использованием компьютера. Также является неизвестным геометрическое неравенство, которому удовлетворяют эти выпуклые тела.

Пусть κ – пространство компактных выпуклых тел, принадлежащих 3-мерному пространству. K – некоторое выпуклое тело, принадлежащее κ, тогда обозначим через $V(K) = V$, $S(K) = S$, $M(K) = M$ – объем, площадь поверхности и интеграл средней кривизны K соответственно. Интеграл средней кривизны для гладкого тела может быть найден по формуле $\int \frac{1}{2}(k_1 + k_2)dS$, где k_1 *и* k_2 есть главные кривизны границы тела K.

Между характеристиками V, S и M выпуклых тел имеют место изопериметрические неравенства. Выпишем некоторые из них:

$$S^2 \geq 3VM,$$
$$M^2 \geq 4\pi S,$$
$$M^3 \geq 48\pi^2 V.$$

В. Бляшке предложил каждому K из κ сопоставить точку (x, y) на плоскости, где (x, y) определил следующим образом:

$$x = \frac{4\pi S}{M^2}, y = \frac{48\pi^2 V}{M^3}.$$

В понятиях интегрально-поперечных мер эти координаты будут выглядеть:

$$x = \frac{V_2 V_0}{V_1^2}, y = \frac{V_3 V_0^2}{V_1^3}.$$

Очевидно, что x и y принимают неотрицательные значения. Отображение $\kappa \to R^2$, предложенное Бляшке, обозначим через F, а образ пространства к на плоскости $\boldsymbol{R}^2$ через $\bar{\kappa}$. Важной проблемой является поиск границы множества $\bar{\kappa}$.

Рассмотрим выписанные выше изопериметрические неравенства. С учетом введенных переменных x и y эти неравенства примут следующий вид:

$$x^2 \geq y,$$
$$1 \geq x,$$
$$1 \geq y.$$

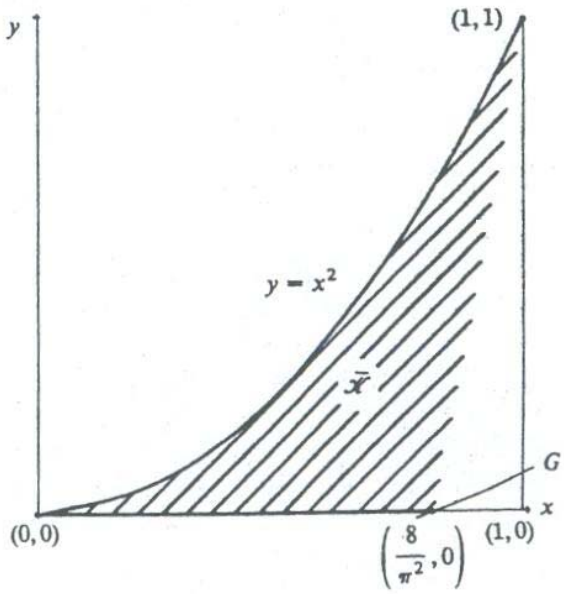


Рис. 2. Диаграмма Бляшке

Полученные неравенства позволяют утверждать, что $\bar{\kappa}$ является ограниченным множеством. Но не все множество, ограниченное этими неравенствами, есть $\bar{\kappa}$, в этом легко убедиться, рассмотрев неравенство $M^2 \geq 4\pi S$

для плоских выпуклых тел. Площадь поверхности $S(K)$ для таких тел равна двум площадям плоского тела, интеграл средней кривизны $M(K)$ равен произведению периметра на угол $\frac{\pi}{2}$, а объем равен 0. Тогда интересующее нас неравенство примет следующий вид:

$$2M^2 \geq \pi^3 S, V = 0,$$

или в терминах x, y:

$$\frac{8}{\pi^2} \geq x, y = 0.$$

Таким образом, граница от точки $\left(\frac{8}{\pi^2}; 0\right)$ до точки (1;1) неизвестна. Некоторую оценку этой границы можно получить, используя неравенство Грёмера:

$$V \geq \frac{\pi S}{24M}\left(S - \frac{2M^2}{\pi^3}\right).$$

В терминах (x, y) получаем:

$$y \geq \frac{\pi}{8} x\left(x - \frac{8}{\pi^2}\right).$$

Пограничная парабола этого неравенства на рисунке обозначена G.

4. ДОПОЛНИТЕЛЬНЫЕ ЗАМЕЧАНИЯ

Очевидно, что все подобные тела под действием F отображаются в одну и ту же точку на диаграмме Бляшке. Единственными прообразами точки диаграммы являются шары. Их образом является точка (1,1).

Прообразами точек параболической границы являются колпаки. Колпаком называется выпуклая оболочка шара и счетного числа внешних точек. Точки при этом заданы таким образом, что часть прямой, соединяющей любые две из них, пересекает шар.

Образом точек и отрезков является точка (0,0) на диаграмме. Плоские выпуклые тела отображаются на ось x, а круги – это единственные тела, чей образ – точка $\left(\frac{8}{\pi^2}; 0\right)$.

5. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

Для расчета характеристик выпуклых многогранников в трехмерном евклидовом пространстве и построения диаграммы Бляшке был создан программный продукт под названием “MBoundVisual”. Среда реализации – Delphi 7. Год создания – 2008.

Программа осуществляет построение выпуклых оболочек в трехмерном евклидовом пространстве по любому заданному набору точек, сохранение и загрузку наборов точек и рассчитанных оболочек, расчет интегралов поперечных мер Минковского, суммирование оболочек по Минковскому, построение диаграммы Бляшке.

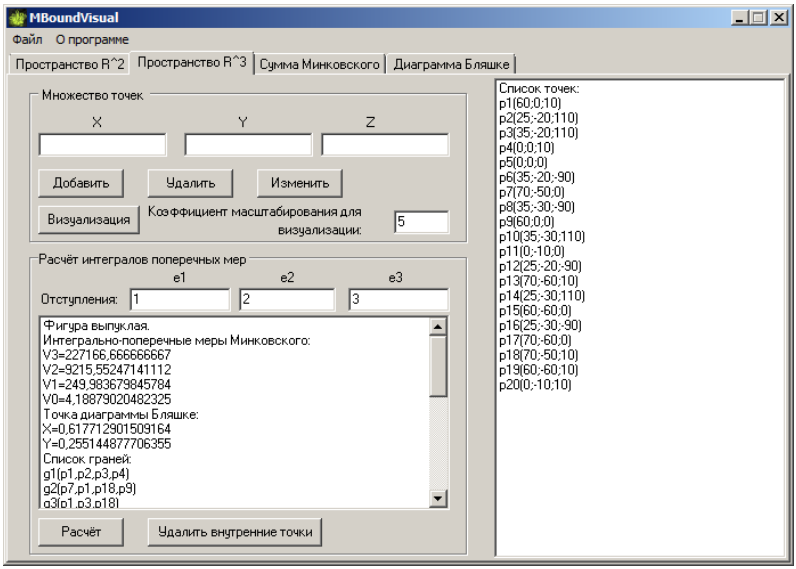


Рис. 3. Рабочее окно программы

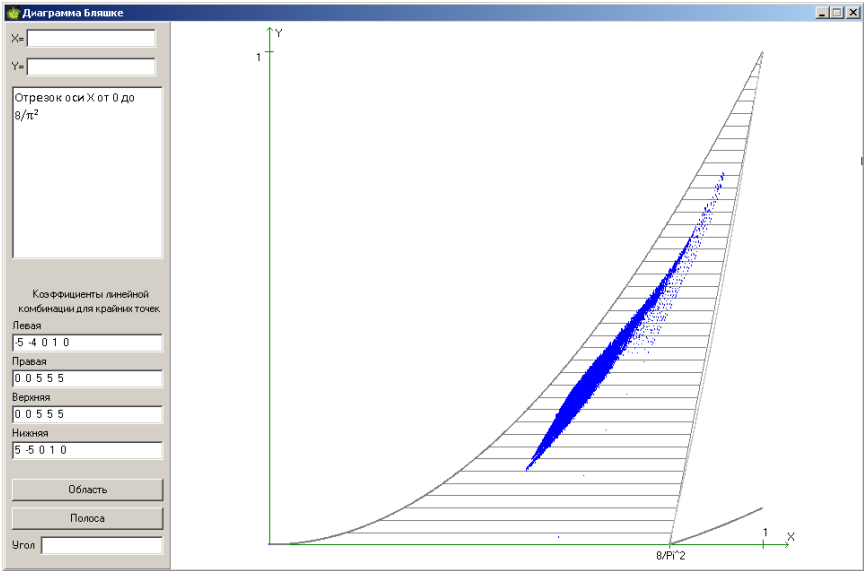


Рис. 4. Диаграмма Бляшке, заполненная точками для конкретной линейной комбинации

СПИСОК ЛИТЕРАТУРЫ

1. Бураго Ю.Д., Залгаллер В.А. Геометрические неравенства. – Л.: Наука, 1980.
2. Бутовский М.М. Исследование свойств выпуклых многогранников в многомерных евклидовых пространствах. Дипломная работа. Рубцовск: Рубцовский индустриальный институт Алтайского государственного технического университета им. И.И. Ползунова, 2008.
3. Васин Д.В. Диаграмма Бляшке множества выпуклых тел // Вестник БГПУ: Естественные и точные науки. Вып. 2. – 2002. – С. 5–9.
4. Тёрстон У. Трехмерная топология и геометрия / Пер. с англ. под ред. О.В. Шварцмана. – М.: МЦНМО, 2001.
5. Чаднов Р.В. Алгоритмы построения выпуклых оболочек и их применение в ГИС и САПР. Дипломная работа. Томск: Томский государственный университет, 2004.
6. Barber C.B., Dobkin D.P., Huhdanpaa H.T. The QuickHull algorithm for convex hull. The Geometry Center, Minneapolis, 1993.

7. Chen J. Computational Geometry: Methods and Applications. Computer Science Department, Texas A&M University, 1996.
8. Peter M. Gruber. Convex and Discrete Geometry. Springer-Verlag, Berlin, Heidelberg, 2007.
9. Sangwine-Yager J.R. The Missing Boundary of the Blaschke Diagram, Amer. Math. Monthly, №96, 1989, p.233-237.
10. Vince John. Geometric Algebra for Computer Graphics. Springer-Verlag, London, 2008.

Д. С. Гордеев

ВИЗУАЛИЗАЦИЯ ВНУТРЕННЕГО ПРЕДСТАВЛЕНИЯ ПРОГРАММ В СИСТЕМЕ ФУНКЦИОНАЛЬНОГО ПРОГРАММИРОВАНИЯ SFP

ВВЕДЕНИЕ

Для решения сложных вычислительных задач можно эффективно использовать многопроцессорные системы, однако для этого нужны специальные средства. В настоящее время в Институте систем информатики им. А. П. Ершова СО РАН ведётся работа над проектом системы функционального программирования SFP [2]. Система функционального программирования SFP предназначена для создания и отладки программ, переносимых на произвольную параллельную архитектуру. Пользователь системы сможет создавать и отлаживать программы на своём рабочем месте и запускать на исполнение на доступном по сети вычислителе с параллельной архитектурой.

В рамках проекта по созданию системы функционального программирования SFP было разработано графовое внутреннее представление IR1 [4]. Далее будут использоваться термины теории графов, с которыми можно ознакомиться в книге [1]. Разработанное внутреннее представление обладает следующими свойствами:

1. Машинная независимость для представления параллелизма и для значений типов данных.
2. Полнота внутреннего представления, позволяющая произвести его ретрансляцию в семантически эквивалентную программу на исходном языке.
3. Возможность ретрансляции после преобразований внутреннего представления, сохраняющих его корректность для исходного языка.
4. Лёгкость исполнения заданных внутренним представлением вычислений без проведения дополнительных преобразований структуры внутреннего представления.
5. Структурированность объектов внутреннего представления для задания естественной иерархии одних конструкций исходного функционального языка в другие.

6. Явное задание любого типа неявных действий с помощью объектов внутреннего представления.
7. Лёгкое введение новых объектов внутреннего представления для задания новых конструкций языков программирования и типов данных.

Основой структуры внутреннего представления являются потоковые графы, которые позволяют задавать явные информационные (семантические) связи (дуги) между операциями (вершинами) и делают процесс интерпретации осуществимым без дополнительных преобразований. Благодаря этому нет побочных эффектов вычислений (ввиду отсутствия понятия переменной) — естественного свойства чисто функциональных языков. Потоковые графы также позволяют задавать параллелизм на уровне отдельных информационно независимых операций, не зависящий от машинной архитектуры.

Внутреннее представление IR1 было разработано на основе языка IF1 [8]. Этот язык основывается на ациклических иерархических графах. Язык IF1 разрабатывался как графовый язык, который мог бы стать результатом трансляции компиляторов нескольких функциональных языков. С помощью такого графового языка становятся возможными объединение и единые преобразования для программ, написанных с помощью разных функциональных языков. Язык IF1 недостаточен для представления программ функциональных языков, но предполагается, что язык IF1 является достаточной основой для более обобщенных промежуточных представлений, требуемых для описания более широкого множества функциональных языков. Программа на языке IF1 состоит из множества графов специального вида, среди которых выделено подмножество графов, соответствующих множеству функций исходной SISAL [3] программы. Граф в IF1 состоит из объектов трёх видов: вершин, дуг и рамки графа. Вершинам и рамке графа приписаны упорядоченные множества входов портов (в которые входят дуги, но не более одной для одного порта) и выходных портов (из которых выходят дуги). Входные порты рамки графа рассматриваются как выходные порты (результаты вычислений) графа, а выходные порты рамки графа, входные порты (параметры) графа. Каждой дуге графа приписан тип (если IF1 задаёт строго типизированный язык) пересылаемого значения. Вершины обозначают операции над своими входами (аргументами), результаты которых находятся на выходах вершины. Вершины бывают простыми и составными. Простые вершины (или просто вершины) не имеют внутренней структуры, помимо ассоциированной с ними операции. Составные

вершины дополнительно содержат упорядоченное множество графов. Их количество и все связи между входными (выходными) портами составной вершины и входными (выходными) портами этих графов задаются типом (или семантикой) операции составной вершины. Структурированность вычислений, задаваемых IF1 графом, основывается на иерархичности IF1 графов, заданной посредством графов составной вершины. Ясно, что вычисления, заданные подобным образом, определяют частичный порядок над общей последовательностью выполнения вершин-операций одного IF1 графа. Несравнимые между собой операции можно выполнить параллельно, что позволяет естественным, не зависящим от машинной архитектуры, образом задать модель параллельных вычислений, причём на очень низком уровне — уровне отдельной операции. К тому же IF1 граф допускает лёгкую интерпретируемость (исполнение), так как он задаёт потоковую модель вычислений, для исполнения которой (с некоторыми ограничениями) можно использовать даже суперкомпьютеры с потоковой архитектурой.

Реализация рассмотренных сущностей графового языка IF1 с помощью интерфейсов классов C++ получила название IR1 (Internal Representation). Нужно отметить, что важную особенность системы введенных интерфейсов, реализующих объекты графа IF1, а именно, то, что интерфейсы для реализации входных и выходных портов не различаются. Это позволяет рассматривать единый интерфейс порта с этих двух точек зрения одновременно. Теперь один и тот же порт может служить выходом графа и в то же время являться входом для дуг, принадлежащих этому графу (и наоборот). Такая унификация позволила рассматривать граф в качестве вершины в графе-владельце, ничего не знающем о внутреннем устройстве такой вершины. Таким образом, можно определить IR1 граф как объект, содержащий упорядоченные множества входных (выходов для дуг графа) и выходных (входов для дуг графа) портов и множество (с задаваемым в некоторых случаях порядком) IR1 графов, являющихся вершинами этого графа. Причем эти графы иногда будут называться подграфами относительно родительского графа, их содержащего. Можно представлять составную вершину как IR1 граф без дуг, содержащий в качестве вершин IR1 графы, соответствующие графам составной вершины. Тогда с помощью одного интерфейса IR1 графа можно реализовать несколько сущностей языка IF1: граф, вершина, составная вершина. Таким образом, внутреннее представление IR1 модуля языка SISAL можно представить в виде множества IR1 графов с пометками, соответствующих множеству функций и редукций исходной SISAL программы, и множества именованных типов, которые будут экспортированы из модуля SISAL программы. Внутреннее представление всей

программы на языке SISAL состоит из множества внутренних представлений IR1 для отдельных модулей программы.

Система SFP использует язык SISAL 3.1 и его транслятор во внутреннее представление IR1. Внутреннее представление IR1 является расширением иерархического ориентированного ациклического графа. Расширение заключается в том, что каждая вершина содержит два упорядоченных набора портов. Каждая программа на функциональном языке вычисляет некоторую функцию, и соответствующее внутреннее представление IR1 отражает способ вычисления, а именно, то, что функция вычисляется через другие функции. Иерархический граф подходит для задания такой структуры. Здесь и далее под IR1-графом подразумевается граф, соответствующий внутреннему представлению IR1. В IR1-графе каждая вершина соответствует некоторой функции. Если функция вычисляется с использованием некоторых других функций, то соответствующая вычисляемой функции IR1-вершина является составной, то есть содержит некоторый IR1-граф. Ниже будет приведено более подробное описание внутреннего представления IR1.

Программы в системе функционального программирования SFP транслируются во внутреннее представление IR1. На полученных представлениях можно проводить интерпретацию, отладку, оптимизацию и трансляцию. Всеми этими процессами полезно управлять, поэтому полезно иметь и визуальное представление программ. В настоящее время существует много систем визуализации графов (некоторые указаны здесь [9]), позволяющих получать изображения графов разных типов, в том числе ориентированных ациклических. Однако в большинстве случаев такие системы имеют слабую поддержку визуализации иерархической структуры либо совсем не поддерживают такой стиль изображения.

В данной статье предлагается алгоритм укладки IR1-графов. Для укладки неориентированных графов известно достаточно много разных подходов: силовой метод, метод концентрических окружностей и т.д. Для укладки ациклических ориентированных графов в основном используется поуровневый метод [5] и различные его модификации; ниже будет приведено общее описание данного подхода.

2. ОБЩИЙ АЛГОРИТМ УКЛАДКИ ОРИЕНТИРОВАННЫХ АЦИКЛИЧЕСКИХ ГРАФОВ

Данный подход позволяет для произвольного ациклического ориентированного графа построить поуровневое представление с дугами в виде ломаных или сплайнов, решая следующие три задачи:

a) распределение вершин по уровням таким образом, чтобы все дуги имели одно направление;

b) выбор порядка вершин на уровне, задающий минимальное число пересечений дуг;

c) определение координат вершин на уровне с целью минимизации числа изломов дуг.

Распределение вершин по уровням

Каждой вершине присваивается число, указывающее уровень вершины так, чтобы все дуги, соединяющие вершины, следовали от большего номера к меньшему номеру; при этом вершины одного уровня не должны быть соединены друг с другом. В общем случае для этого шага формулируется задача линейного программирования. После этого каждая длинная дуга, то есть соединяющая вершины с уровней, номера которых отличаются больше, чем на единицу, заменяется цепочкой фиктивных вершин таким образом, что каждая следующая вершина в цепочке лежит на уровне с номером, ровно на единицу меньшим, чем предыдущая вершина.

На данном этапе уже можно вычислить конечную вертикальную координату вершины. Ясно, что для любого ациклического графа можно найти такое распределение по уровням.

Такое построение приводит к тому, что существуют дуги, соединяющие вершины только соседних уровней. Таким образом, следующий шаг будет решать задачу переупорядочения вершин только для двух соседних уровней. Кроме того, такое решение избавляет от необходимости контролировать в дальнейшем пересечение дуг и вершин, так как при отображении фиктивные цепочки будут заменены ломаными линиями или сплайнами. Ясно, что в таком случае дуги не будут пересекать вершины.

Определение порядка вершин на уровнях

На данном этапе необходимо выбрать порядок вершин на каждом уровне. Выбор оптимального порядка позволит уменьшить число пересечений дуг. Если в графе уже нет “длинных” дуг, то число пересечений определя-

ется только порядком вершин на уровнях. После построения поуровневого разбиения встаёт задача определения порядка расположения вершин на каждом уровне графа. Задача преследует цель минимизировать число пересечения дуг. Следует заметить, что количество пересечений дуг не зависит от конечных координат вершин на уровне, а только от порядка следования вершин внутри каждого уровня. В общем случае задача является NP-полной уже только для двух уровней. Таким образом, методы, получающие точные решения, подходят только для маленьких графов, алгоритмы, вычисляющие приближённое решение, являются более предпочтительными. При этом есть несколько вариаций:

- фиксирование порядка соседних уровней, это может быть нижний уровень, или верхний и нижний уровни;
- просмотр уровня справа налево либо слева направо, также возможно чередование направления просмотра;
- критерий остановки просмотра, просмотры делаются, пока число пересечений уменьшается, либо пока не превышен заранее заданный предел числа просмотров.

Для получения первоначального порядка используется алгоритм, позволяющий избежать пересечения дуг, если бы исходный граф был деревом. Метод состоит в проведении поиска в глубину, при котором вершины с одного уровня получают номера в порядке их просмотра в ходе такого поиска.

Определение координат вершин на уровнях

На данном этапе для каждой вершины вычисляются вертикальная и горизонтальная координаты; вертикальная координата выбирается с учётом номера уровня, к которому приписана вершина, а горизонтальная выбирается так, чтобы минимизировать число изломов длинных дуг. Обычно дуги изображают в виде прямолинейного отрезка; в таком случае длинные дуги изображаются в виде ломаных линий. После определения порядка вершин на уровне необходимо определить их настоящие горизонтальные координаты. Обычно дуги графа изображаются в виде ломаных, где в точках излома находятся мнимые вершины, так что данная задача решает также задачу проведения длинных дуг. В случае наличия длинных дуг, задача нахождения точных координат решается из соображений минимизации числа изломов длинных дуг. При этом не должен быть изменён порядок следования вершин внутри каждого уровня.

3. МОДИФИЦИРОВАННЫЙ АЛГОРИТМ УКЛАДКИ ГРАФА ДЛЯ IR1-ГРАФОВ

Распределение вершин по уровням

Как уже было описано, IR1-граф представляет собой два упорядоченных множества портов и множество вершин, также являющихся IR1-графами, а дуги задаются неявно, с помощью связей между портами. Под неявным заданием подразумевается, что нет отдельной сущности как “дуга”; вместо этого порты сами хранят информацию о том, каким портам они передают данные либо от какого порта получают данные. Порты могут быть входными или выходными; в данной терминологии принято, что входные порты могут быть инцидентны только одной дуге, а выходные порты могут быть инцидентны нескольким дугам. Обход графа можно начинать либо с входных, либо с выходных портов. Однако в IR1-графе не существует вершин, которые не возвращают значения, то есть без выходных портов, но существуют вершины, которые значения только возвращают, то есть только с выходными портами. Поэтому разумно производить обход снизу или с выходных портов.

Для удобства выходным портам графа присваивается нулевой номер уровня. Процесс вычисления номеров уровней является итеративным.

На первом шаге в качестве текущего значения уровня принимается ноль. Далее находится текущее множество вершин: для каждого выходного порта графа вычисляется вершина, которой принадлежит порт, поставляющий данные для выходного порта; множество таких вершин образует текущее множество вершин. Сложность этой операции $O(P^2)$.

Если множество вершин оказалось пустым, то на этом процесс разбиения заканчивается. Если множество таких вершин не пусто, то для каждой вершины устанавливается значение уровня, равное максимуму между текущим значением уровня, увеличенным на единицу, и текущим уровнем вершины. Начальное значение уровня для всех вершин равно –1. Далее для каждой вершины вычисляется множество входных портов, а для каждого такого порта вычисляется дуга, конечным портом которой он является. Далее по дуге вычисляется её начальный порт. Объединение вершин, являющихся владельцами таких начальных портов, даёт текущее множество вершин для следующей итерации. Процесс продолжается, пока текущее множество вершин не окажется пустым. Последний шаг алгоритма разбиения таков: просматривается множество вершин, и все вершины, не получившие номера уровня, получают номер текущего уровня, увеличенный на едини-

цу. Такие вершины могут существовать, так как граф может оказаться несвязным; граф будет таким, когда программа содержит более одной функции.

Сложность этого шага $O(P^2 + V^3 * P)$. Но это очень грубая оценка, и такой вариант более точен: $O(P_{out}{}^2 + L^{\max} * (V_l^{\max})^2 * P_{in}^{\max} *)$, где P_{out} – число выходных портов графа, $L^{\max}$ – число уровней в графе, $V_l^{\max}$ – максимальное число вершин на уровне, $P_{in}^{\max}$ – максимальное число входных портов на вершинах. Однако эта оценка заранее неизвестна.

```
Algorithm
begin
   Level = 0
   Tmp = Vertices = <empty>
   Ports = IR1Graph.OutPorts
   for i := 1 step 1 until Ports.Count do
   begin
      Port := Ports[i]
      for j := 1 step 1 until Port.ParentVertices.Count do
      begin
        V := Port.ParentVertices[j]
        if (V not in Vertices) then Vertices.Add(V)
      end
    end

   while (Vertices is not empty) do
   begin
      for i := 1 step 1 until Vertices.Count do
      begin
          V :=  Vertices[i]
          for j := 1 step 1 until V.ParentVertices.Count do
          begin
            V1 := V.ParentVertices [j]
            if (V1 not in Tmp) then Vertices.Add(V1)
          end
          V.Level := max(V.Level,Level+1)
       end
      Vertices = Tmp;
```

```
            Tmp = <empty>
            Level = Level + 1
          end

        for i := 1 step 1 until IR1Graph.Vertices.Count do
        begin
            Vertex := IR1Graph.Vertices[i]
           if (Vertex not  is marked) then  Vertex.Level = Level+1
         end
      end
```

Для каждой дуги проверяется разность между уровнями начального и конечного портов, и если какой-либо из портов принадлежит вершине, то его уровень определяется уровнем вершины-владельца. Если разность оказывается больше единицы, то вместо дуги вставляется цепочка вершин со специальной пометкой, определяющей то, что эти вершины не будут показываться при конечной визуализации. Сложность операции оценивается так $O(E*V)$, более точная оценка выглядит как $O(E_{long} * L_{\max})$, где E_{long} – число "длинных" дуг, то есть тех, которые соединяют вершины с не соседних уровней, $L_{\max}$ – число уровней. Однако, опять же, эта оценка заранее неизвестна.

```
      Algorithm
       begin
          for i := 1 step 1 until IR1Graph.Edges.Count do
          begin
              Edge := IR1Graph.Edges[i]
              if (Edge.StartPortLevel - Edge.EndPortLevel > 1) then
AddChainInstead(IR1Graph,Edge)
           end
       end
```

Определение порядка вершин на уровне

В общем случае задача очень сложная, однако, в случае IR1-графов можно использовать достаточно жёсткие ограничения, которые позволяют делать некоторые предварительные заключения. Можно считать, что порядок вершин первого уровня, то есть тех, которые лежат на самом нижнем

уровне, задаётся только порядком выходных портов графа. Так как множества портов на графах являются упорядоченными, и для каждого порта существует ровно одна вершина-предок, после внесения мнимых вершин будет именно ровно одна вершина. Таким образом, порядок первого уровня однозначно вычисляется с использованием порядка выходных портов IR1-графа.

Далее будем считать, что порядок вершин предыдущего уровня зафиксирован. Так как для каждой вершины известен упорядоченный набор портов, то также известен и упорядоченный набор вершин-предков. Таким образом, если для каждой вершины предыдущего уровня выписать свой упорядоченный набор вершин-предков, причем так, что порядок наборов предков соответствует порядку вершин на предыдущем уровне, то получается некоторый упорядоченный набор вершин. Если каждая вершина в полученном наборе встречается только один раз, то можно считать, что порядок вершин следующего уровня получен, можно переходить к следующей итерации. Если существуют копии вершин, непосредственно следующие друг за другом, то такие последовательности копий вершин следует заменить на одну соответствующую вершину. После этого следует просмотреть все вершины слева направо, имеющие ровно один выходной порт, а порядок таких вершин следует зафиксировать, так как их перестановки могут только увеличить число пересечений дуг. Как только встретится вершина с большим количеством выходных портов, просмотр следует прекратить. Далее нужно произвести аналогичный просмотр справа налево, начиная с самой правой вершины. Просмотр следует прервать, если встретится вершина более чем с одним выходным портом, или если встретится вершина с фиксированным порядком. Если порядок всех вершин на уровне зафиксирован, то процедуру следует остановить и перейти к определению порядка на следующем уровне. Если остались вершины с незафиксированным порядком на уровне, то к ним применяется следующий этап итерации. Следует заметить, что для всех таких вершин известно следующее: они имеют больше одного выходного порта, и между двумя любыми вершинами нет вершины с зафиксированным порядком.

Суть дальнейшей процедуры заключается в следующем. Необходимо избавиться от копий, таким образом, чтобы число пересечений дуг было минимально, так как перед началом процедуры структура графа такова, что пересечений дуг нет, поэтому для удаления повторов будут выбираться те варианты, которые добавляют меньшее количество пересечений.

Прежде всего, для каждой уникальной вершины из обрабатываемого множества, вычисляется количество копий и минимальное расстояние до

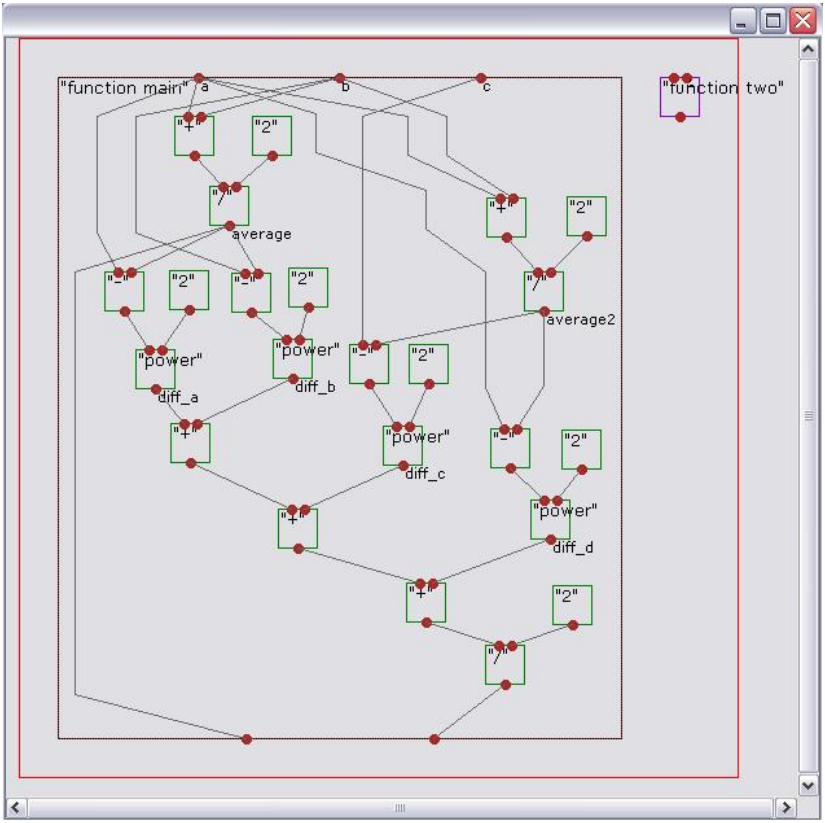
ближайшей копии. Выбор уникальной вершины для удаления следует делать, исходя из минимальности расстояния между копиями, так как это внесёт меньше пересечений. Если есть несколько копий на одинаковом расстоянии друг от друга, для удаления следует выбрать ту, удаление которой внесёт меньше пересечений. После каждого удаления необходимо пересчитывать минимальные расстояния между копиями. Процедура повторяется, пока не будут удалены все копии вершин. После этого можно переходить к обработке следующего уровня.

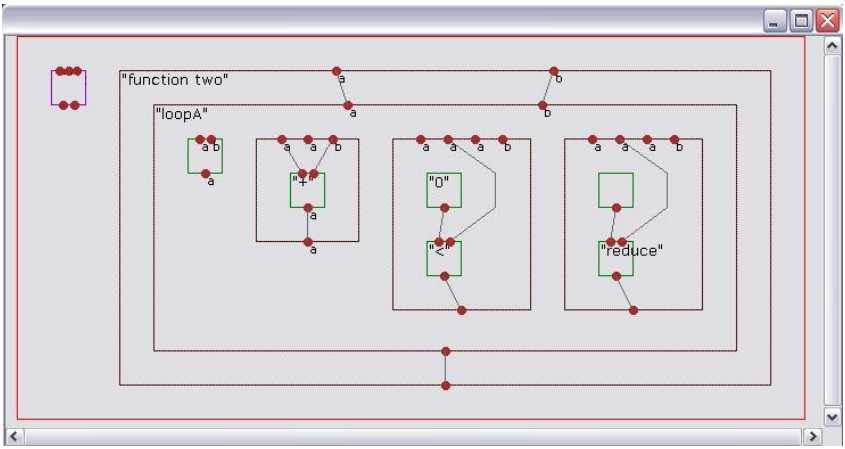
```
Algorithm
begin
   Ports := IR1Graph.OutPorts

      while(CurrentLayer is not Empty) do
      begin //1
         for i := 1 step 1 until Ports.Count do
          begin
            Port :=  Ports[i]
            CurrentLayer.Add(Port.ParentVertex)
           end

         while (true) do
         begin //2
            MergeNearCopies(CurrentLayer)

            if ( not CopiesExists(CurrentLayer) )  then
            begin
               for i := 1 step 1 until CurrentLayer.Count do
                 begin
                   Vertex :=  CurrentLayer [i]
                   for j := 1 step 1 until Vertex.InPorts.Count do
                    Port1 := Vertex.InPorts[j]
                    NewPorts.Add(Port1)
                 end
               break
            end
            else   DeleteApprprpiateVertex(CurrentLayer)

         end //2
```

```
        Ports += NewPorts
        NewPorts := <empty>
      end //1
    end
```

Очевидно, что алгоритм всегда останавливается, так как для каждого уровня генерируется конечное число копий вершин, и за каждую итерацию обязательно удаляется одна копия, а если копий больше нет, то обрабатывается следующий уровень.

Применение алгоритма





Код соответствующей SISAL-программы:

a.dec.sis

```
function main[integer,integer,integer returns integer,integer]
function two[integer,integer returns integer]
```

a.def.sis

```
definition a
```

```
function main(a:integer; b:integer; c:integer  returns integer, integer)
let average := (a + b) / 2;
   average2 := (a + b) / 2;
   diff_a  := (a - average) ** 2;
   diff_b  := (b - average) ** 2;
   diff_c  := (c - average2) ** 2;
   diff_d  := (a - average2) ** 2;
   diff_e  := two(diff_c,diff_a)
```

```
in average, (diff_a + diff_b + diff_c + diff_d)/2
end let
end function

function two(a:integer; b:integer returns integer)
  for repeat
    a := old a + b
  while a > 0
  returns value of a
  end for
end function
```

ЗАКЛЮЧЕНИЕ

В статье описывается алгоритм укладки иерархических ациклических ориентированных графов специального вида, соответствующих внутреннему представлению программ на функциональном языке SISAL 3.1. С использованием алгоритма реализован компонент визуализации, позволяющий получать изображения внутреннего представления и осуществлять некоторые манипуляции с изображением: изменение масштаба, перемещение вершин, перемещение по графу, сворачивание-разворачивание составных вершин.

СПИСОК ЛИТЕРАТУРЫ

1. **Касьянов В. Н., Евстигнеев В. А.** Графы в программировании: обработка, визуализация и применение. – СПб.: БХВ-Петербург, 2003. – 1104 с.
2. **Kasyanov V. N., Stasenko A. P.** A Functional Programming System SFP: Sisal 3.1 Language Structures Decomposition // Parallel Computing Technologies – Lect. Notes Comput. Sci. – Springer, 2007. – Vol. 4671. – P. 62–73.
3. **Стасенко А. П., Синяков А. И.** Базовые средства языка SISAL 3.1. – Новосибирск, 2006. – 56 с. – (Препр. / РАН. Сиб. Отд-ние. ИСИ; № 132)

4. **Стасенко А. П.** Внутреннее представление системы функционального программирования SISAL 3.0. – Новосибирск, 2004. – 54 с. – (Препр. / РАН. Сиб. Отд-ние. ИСИ; № 110)
5. **Sugiyama K., Tagawa S., Toda M.** Methods for Visual Understanding of Hierarhical System Structures // IEEE Transactions on Systems, Man and Cybernetics. – 1981. – Vol. 11, Iss. 2. – P. 109–125.
6. **Emden R. Gansner, Eleftherios Koutsofios, Stephen C. North, Kiem-Phong Vo**. A Technique for Drawing Directed Graph // IEEE Transactions on Software Engineering. – 1993. – Vol. 19 , Iss. 3. – P. 214–230.
7. **Juonger M., Mutzel P.** 2-Layer Straightline Crossing Minimization: Performance of Exact and Heuristic Algorithms. – 1997. – Vol. 1, N 1. – P. 1-25.
8. IF1: an intermediate form for applicative languages / Skedzielewski S., Glauert J — Livermore, CA, 1985. — (Lawrence Livermore National Laboratory; M-170).
9. **Лисицын И. А.** Системы визуализации и редактирования графовых объектов: обзор. – Новосибирск, 2000. – 42. с. – (Препр. / РАН. Сиб. Отд-ние. ИСИ; № 76).
10. **Гордеев Д. С.** Визуализация в системе функционального программирования SFP. Технологии Microsoft в теории и практике программирования // Тез. докл. Конф.-конкурса работ студентов, аспирантов и молодых учёных. – Новосибирск, 2007. – С. 103.
11. **Гордеев Д.С**. Визуализация внутреннего представления программ в системе функционального программирования SFP // Материалы II Международной научной студенческой конференции «Научный потенциал студенчества – будущему России». Том третий. Математика. – Ставрополь, 2008. – С. 180.

И.В. Гужавина

ТЕХНОЛОГИИ ОБРАБОТКИ ИЗОБРАЖЕНИЯ СО СПУТНИКА

ВВЕДЕНИЕ

Снимки земной поверхности, полученные путём космической съёмки, отличаются тем, что при целостном характере изображения местности они охватывают огромные площади (на одном снимке от десятков тысяч $км^2$ до всего земного шара). Это позволяет изучать по космическим снимкам основные структурные, региональные, зональные и глобальные особенности атмосферы, литосферы, гидросферы, биосферы и ландшафты нашей планеты в целом. Космические съемки Земли применяются во многих областях науки; таким образом, накопление базы данных изображений Земли является важной задачей.

Существует несколько видов различных спутников, с которых ведется наблюдение поверхности Земли: геостационарные (ГСО) и высокоэллиптические (ВЭО). Эти спутники различаются орбитами, по которым они совершают движение, углом съемки, стоимостью и прочим. С геостационарного спутника можно получить прямые изображения Земли низкого разрешения, но данный спутник достаточно дорого стоит. В отличие от ГСО, ВЭО позволяет при наблюдении с различных участков орбиты получать изображения фонов Земли в разных масштабах. Последовательные 9 витков орбиты спутников ВЭО имеют устойчивую конфигурацию, охватывающую всю Землю при различных условиях освещенности. Это позволяет в широких пределах выбирать условия наблюдения фонов Земли в интересующих регионах, максимально приближенные к условиям ГСО. Но при съемке с высокоэллиптической орбиты камерой с малым углом зрения ($0{,}12^\circ \times 0{,}24^\circ$) и фотоприемной матрицей небольшого размера (256×500) получаемые изображения покрывают небольшие участки земной поверхности. Для получения изображений высокого разрешения, покрывающих большие площади, необходимо построить панораму. Изображения, получаемые при съемке с высокоэллиптической орбиты, содержат проективные искажения. Поэтому при создании из этих изображений панорам необходимо компенсировать проективные искажения.

Таким образом, с высокоэллиптического спутника можно получить изображения более высокого разрешения, но с проективными искажениями. Стоимость таких спутников относительно мала, поэтому для получения изображений с большим разрешением и из-за более низкой стоимости лучше использовать высокоэллиптические спутники.

1. АНАЛИЗ СУЩЕСТВУЮЩИХ РЕШЕНИЙ

Изображения, получаемые при съемке с высокоэллиптической орбиты, содержат проективные искажения, обусловленные изменением координат спутника и точки прицеливания. После съемки получается большое количество разрозненных изображений, которые необходимо объединить для получения более общей картины.

Для компенсации проективных искажений используется известный алгоритм, описанный в [1]. Он представляет собой ряд геометрических преобразований, которые переводят одну координатную сетку в другую. Но так как изображения имеют разный масштаб (на один пиксель изображения ГСО приходится приблизительно 1,5–6 пикселей изображения ВЭО), преобразование «пиксель в пиксель» при расчете яркости будет некорректным. Необходим алгоритм для пересчёта яркости в каждом пикселе координатной сетки ГСО, который будет усреднять значение яркости всех пикселей ВЭО, попавших в данный пиксель.

Таким образом, для пересчета изображений необходимо пройти 2 этапа:

- пересчет координатной сетки ВЭО в координатную сетку ГСО, то есть сопоставление каждой точке изображения ВЭО точки на изображении ГСО;
- пересчет яркости для каждой точки изображения.

После компенсации проективных искажений и пересчета яркости осуществляется создание панорамного изображения.

На сегодняшний день существует достаточно литературы, описывающей проблему создания панорам из отдельных изображений, существует также несколько коммерческих приложений, решающих данную задачу, например программное обеспечение PhotoStitch для цифровых фотокамер Canon, REALVIZ Stitcher, Arcsoft PanoramaMaker, iSee PhotoVista Panorama и другие [2]. Практически все приложения требуют предварительной инициализации, т.е. набора входных параметров, по которым будет происходить сопоставление. Так PhotoStitch работает только с набором изображений, полученных вертикальным или горизонтальным сдвигом, либо квад-

ратной матрицей изображений. У REALVIZ Stitcher обеспечивает подход «drag, drop and stitch», то есть пользовательский интерфейс для приблизительного расположения изображений, который ограничивает зону поиска особых точек, а также позволяет приблизительно задать поворот одного изображения относительно другого.

Практически все приложения обладают как автоматическим, так и ручным режимом создания панорам, но автоматические режимы зачастую работают некорректно. Так, при создании панорамы с помощью iSee PhotoVista Panorama точки совпадения определились неправильно при автоматическом режиме. Данные программы ориентированы на ручную подстройку изображений; кроме того, ни одна из представленных программ не позволяет компенсировать проективные искажения, не работает с изображениями, имеющими различный масштаб, плохо сглаживает яркость и зачастую оставляет видимые границы на склеенном изображении. Все программные продукты не предусматривают использования дополнительной информации об изображениях, такой как параметры и положение камеры.

Алгоритм создания панорам состоит из следующих этапов:

1. Поиск особых точек (либо других свойств) изображений для сопоставления.
2. Поиск «ближайших соседей» для сопоставления особых точек.
3. Вычисление гомографии для пар изображений.
4. Вычисления параметров камер.
5. Построение панорамы.

В данном случае геометрия съемки известна: матрица внутренних параметров камеры, матрица поворота камеры и её положение. Таким образом, построить панораму можно сразу. Но если данные имеют ошибку, например, в координатах положения камеры, то панорама может получиться разрывной. Для поправки данных можно использовать алгоритм создания панорам, но в урезанном виде. Также необходим алгоритм, который позволит склевать изображения так, что в панораме не будет видно границ стыков между изображениями и яркость изображения будет равномерной.

Существует два подхода для поиска ключевых особенностей (особых точек) изображений – прямые методы [3] и методы, основанные на выявлении ключевых свойств изображения [3]. Основным преимуществом прямых методов является использование всех сведений об изображении, но для их применения нужна точная инициализация, т.е. прямое указание того как нужно склеивать изображения. Методы, основанные на выявлении ключевых свойств изображения, более стойки к движению, работают гораздо быстрее прямых методов. Однако основным преимуществом этих методов

является автоматическое нахождение перекрывающихся участков изображений в несортированном наборе изображений.

Среди методов, основанных на выявлении ключевых свойств изображения, наиболее известным и широко применяемым является детектор Харриса [4]. Сопоставление особенностей происходит путём сравнения их окрестностей кросскорреляционным или другим методом. Однако на больших по размеру изображениях (более 256×256 пикселей) сразу несколько особенностей на первом изображении могут оказаться близкими к одной и той же особенности на втором изображении. Отличить ложные соответствия от верных не всегда возможно. Данный способ плохо работает, и в случае, если изображения имеют разную яркость.

Более эффективным является алгоритм SIFT [5], который лишен недостатков вышеописанного алгоритма, но обладает высокой сложностью обработки.

SIFT (Scale Invariant Feature Transform) преобразует данные изображения в независимые от масштаба координаты, связанные с локальными свойствами изображения. Данный алгоритм находит достаточно большое количество особых точек, которые отвечают различным масштабам и участкам одного и того же изображения. Для каждой точки строится некоторое описание, которое позволяет с большой вероятностью найти точное соответствие для данной точки на других изображениях.

Таким образом, данный алгоритм позволяет выявлять характерные свойства изображения (особые точки), которые не зависят от колебаний яркости, и находить для них точное соответствие.

После того, как были найдены особые точки для всех изображений, необходимо найти соответствия между изображениями, т.е. набор особых точек, совпадающих у двух или более изображений. Эта задача соответствует поиску N ближайших соседей. Для решения этой задачи существует много подходов. Один из прямых методов решения – метод ячеек. Всё пространство делится на равные небольшие k-мерные кубики (k – размерность пространства) – «ячейки» – по определённому ключу, затем поиском по спирали находится лучшее соответствие для заданной записи. Несмотря на то, что эта процедура минимизирует количество рассматриваемых записей, она требует больших временных затрат, а также много памяти.

Другой подход описан в [6]. Этот метод, основан на формировании проекций записей на один или несколько ключей, хранении линейного списка этих ключей и поиска только среди тех записей, которые достаточно близки по значениям к этим ключам. В [6] показано, что данный метод требует

$km^{1/k}N^{1-1/k}$ вычислений, где k – размерность пространства, N – число рассматриваемых записей, m – количество ключей.

Еще один подход, описанный в [7], основан на модели k-d деревьев, которые являются обобщением бинарных деревьев. K-d дерево представляет собой двоичное дерево, каждая вершина которого содержит подмножество записей и разделение данного подмножества. Такая структура данных позволяет эффективно разделить записи и осуществлять поиск в пределах небольших множеств. Как показано в [7], этот метод требует $kN\log N$ вычислений, а время поиска пропорционально $\log N$.

Когда вычислены ближайшие соседи, необходимо склеить изображения. Основной проблемой здесь является разница в яркости изображений и появление вследствие этого границ на склеенном изображении. Для решения данной проблемы существует несколько подходов. Первый – линейно уменьшать влияние одного изображения и увеличивать влияние второго изображения. Данный метод достаточно прост и быстр, но после его применения граница может остаться, хотя и будет менее заметна.

Более гладкий переход может быть получен при использовании техники, описанной в [8]. С использованием данной техники строится «самая гладкая» из возможных функция коррекции для каждого изображения, входящего в панораму. Однако данная техника не применима к большим изображениям, так как для вычисления функции коррекции используется итеративный алгоритм релаксации.

Подход, предложенный в [9] – метод сплайнов с переменной разрешающей способностью. В данном подходе склеиваемые изображения фильтруются полосовым фильтром, после чего получается последовательность изображений с различной пространственной частотой. Затем для каждого изображения, участвующего в панораме, строится взвешивающую функцию, которая также фильтруется. Наконец, для каждой частоты изображения складываются с соответствующей взвешивающей функцией и суммируются по всем частотам. Данный подход обеспечивает склейку панорамного изображения; при этом на изображении не появляются такие искажения, как двойные объекты или видимая граница перехода.

2. ТРЕБОВАНИЯ К РАЗРАБАТЫВАЕМЫМ АЛГОРИТМАМ И ПРОГРАММНЫМ СРЕДСТВАМ

Алгоритмы и программные средства преобразования изображений разрабатываются для пересчета изображений фонов, наблюдаемых с эллипти-

ческих орбит, в вид, характерный для геостационарных условий наблюдения, с целью формирования базы экспериментальных данных наблюдений фона Земли.

В процессе пересчета полученных данных на условия ГСО необходимо решить следующие основные задачи:

- провести пересчет на условия ГСО с учетом различия условий и аппаратуры наблюдения;
- построить панораму;
- пересчет изображений происходит по заранее подготовленному заданию, в котором содержатся необходимые параметры.

Для пересчета изображения необходимо пересчитать координатную сетку, а затем яркость для каждого пикселя. Для пересчета координатной сетки используется известный алгоритм, компенсирующий проективные искажения.

При пересчете яркости необходимо учесть, что пиксели изображений с ВЭО и с ГСО имеют различные размеры. Пиксель ГСО произвольно отображается на координатную сетку ВЭО; таким образом, возникают пиксели ВЭО, которые частично попали в пиксель ГСО. Для наиболее точного пересчета яркости необходимо учитывать все вхождения пикселей ВЭО в пиксель ГСО. Таким образом, необходим алгоритм, учитывающий все особенности данных изображений.

Для построения панорам существует достаточное количество различного программного обеспечения, но оно в большинстве своем рассчитано на интерактивную работу, т.е. ручную настройку пользователем параметров и приблизительное совмещение изображений, и не учитывает информацию о съемке. Так как известны параметры съемки, необходимо разработать алгоритм, который позволит автоматически создавать панорамные изображения, уточнять параметры съемки, сглаживать границы склеенных изображения, усреднять яркость в пределах одного панорамного изображения. Алгоритм для создания панорамы будет состоять из следующих шагов:

1. Поиск особых точек (либо других свойств) изображений для сопоставления.
2. Поиск «ближайших соседей» для сопоставления особых точек.
3. Уточнение параметров съемки.
4. Построение панорамы.

При поиске особых точек необходимо учесть, что изображения могут иметь различную яркость.

Для более точного совмещения необходимо достаточное количество особых точек, поэтому для поиска ближайших соседей необходим доста-

точно простой алгоритм, который будет за небольшое время находить возможные варианты для сопоставления изображений с большой вероятностью.

Также необходимо выбрать алгоритм для «склейки» изображений. Данный алгоритм должен достаточно гладко совмещать изображения, т.е. после склейки граница между фрагментами изображения должна отсутствовать.

Таким образом, необходим алгоритм, который будет удовлетворять следующим требованиям:

- учитывать такие особенности изображений, как различная яркость и различный масштаб;
- достаточно быстро работать с большим количеством изображений;
- автоматически собирать панорамные изображения по уточнённым параметрам съемки;
- обеспечивать отсутствие видимых границ склейки на панорамном изображении.

Для удовлетворения данных требований для каждого этапа были выбраны соответствующие алгоритмы:

- SIFT для поиска особых точек.
- Метод Best Bin First(BBF) на основе k-d деревьев для поиска ближайших соседей.
- Метод сплайнов с переменной разрешающей способностью для склейки изображений.

3. АЛГОРИТМЫ

3.1. Пересчет координатной сетки

Пересчёт координат изображений из ВЭО в ГСО производится в предположении, что участок поверхности, наблюдаемый в текущий момент, аппроксимируется фрагментом плоскости касательной к поверхности Земли в точке наведения P.

Для пересчёта изображения используются следующие входные параметры:

- t – время получения исходного изображения.

- Параметры спутников: векторы положения спутника $\overrightarrow{t_{ВЭО}}$ и $\overrightarrow{t_{ГСО}}$, форматы фотоприёмных матриц ($Nx_{ВЭО}, Ny_{ВЭО}$) и ($Nx_{ГСО}, Ny_{ГСО}$), угловые разрешения $\nu_{ВЭО}$ и $\nu_{ГСО}$, координаты главных точек аппаратур наблюдения в элементах фотоприёмной матрицы $\left(u^0_{ВЭО}, v^0_{ВЭО}\right)$ и ($u^0_{ГСО}, v^0_{ГСО}$).
- Координаты точки P на поверхности Земли, на которую наведена визирная ось аппаратуры спутника, находящегося на высокой эллиптической орбите.

Алгоритм пересчёта координатной сетки состоит из следующих этапов: по входным значениям вычисляется матрица ориентации аппаратуры наблюдения ГСО, затем координаты изображения с ВЭО пересчитываются в ГСО.

Положение и ориентация камеры спутника для текущего кадра задается вектором трансляции $\overrightarrow{t_{ВЭО}}$ и матрицей поворота $R_{ВЭО}$, которая в геоцентрической неинерциальной системе представляется в виде:

$$R_{ВЭО}(t) = \begin{pmatrix} i_1(t) & i_2(t) & i_3(t) \\ j_1(t) & j_2(t) & j_3(t) \\ k_1(t) & k_2(t) & k_3(t) \end{pmatrix} = \begin{pmatrix} i^T(t) \\ j^T(t) \\ k^T(t) \end{pmatrix}.$$

Здесь $k^T(t)$ – вектор, задающий направление оси визирования камеры, а $i^T(t)$ и $j^T(t)$ – векторы, определяющие систему координат плоскости изображения. Если координаты спутника задаются вектором $\overrightarrow{t_{ВЭО}}$, а координаты точки наведения вектором $\vec{P}$, то

$$k^{\mathrm{T}}(t) = \frac{\vec{P} - \overrightarrow{t_{ВЭО}}}{|\vec{P} - \overrightarrow{t_{ВЭО}}|}.$$

Наведение на точку визирования осуществляется сначала поворотом камеры в плоскости орбиты, затем выносом оси визирования в требуемую точку. Следовательно, одна из осей камеры (пусть это будет ось $j(t)$) находится в плоскости орбиты. Это означает, что она ортогональна вектору нормали к плоскости орбиты $n_0(t)$. По определению же она ортогональна вектору $k(t)$. Из этих условий следует, что

$$j(t) = \frac{n_0(t) \times k(t)}{|\, n_0(t) \times k(t) \,|}.$$

Ось $i(t)$ дополняет систему до правой тройки, следовательно $i(t) = j(t) \times k(t)$.

Аналогичные вычисления проводятся для матрицы ориентации $R_{ГСО}$.

Преобразование координат изображений из ГСО в ВЭО производится по формулам:

$$[u_{ГСО}, v_{ГСО}] = \frac{1}{\gamma_{ГСО}} \cdot \frac{R_{ГСО} \cdot \left[\left(R^2 - \vec{p}^{\,\mathrm{T}} \vec{t}_{ВЭО} \right) \cdot \vec{I} + \left(\vec{t}_{ВЭО} - \vec{t}_{ГСО} \right) \cdot \vec{p}^{\,\mathrm{T}} \right] \cdot R_{ВЭО}^{\mathrm{T}} A_{ВЭО}^{-1} \vec{v}_{ВЭО}}{\vec{e}_z^{\,\mathrm{T}} R_{ГСО} \cdot \left[\left(R^2 - \vec{p}^{\,\mathrm{T}} \vec{t}_{ВЭО} \right) \cdot \vec{I} + \left(\vec{t}_{ВЭО} - \vec{t}_{ГСО} \right) \cdot \vec{p}^{\,\mathrm{T}} \right] \cdot R_{ВЭО}^{\mathrm{T}} A_{ВЭО}^{-1} \vec{v}_{ВЭО}} \cdot [\vec{e}_x^{\,\mathrm{T}}, \vec{e}_y^{\,\mathrm{T}}],$$

где

$$A_{ВЭО} = \begin{bmatrix} v_{ВЭО}^{-1} & 0 & u_{ВЭО}^{0} \\ 0 & v_{ВЭО}^{-1} & v_{ВЭО}^{0} \\ 0 & 0 & 1 \end{bmatrix} -$$

матрица внутренних параметров камеры ВЭО, $\vec{s} = (X, Y, Z)^{\mathrm{T}}$ – координаты изображения произвольной точки S, заданной в неинерциальной геоцентрической системе координат, $\vec{v}_{ВЭО} = (u_{ВЭО}, v_{ВЭО}, 1)^{\mathrm{T}}$ – координаты в плоскостях изображения камер в пикселях (аналогичные параметры для ГСО), $\vec{p} \cdot \vec{s} = R^2$.

3.2. Вычисление яркости в пикселе ГСО

Пересчет яркости для каждого пикселя ГСО заключается в интерполяция значений яркости элементов изображения ВЭО, накрываемых им целиком или частично.

Так как размер и ориентация пикселей изображений, получаемых с ВЭО и с ГСО, различны, то необходимо найти проекцию пикселя ГСО на координатную сетку ВЭО и рассчитать яркость, учитывая все вхождения пикселей ВЭО в пиксель ГСО.

Для уменьшения вычислений, используя проективные преобразования, сначала вычисляем координаты углов изображения ВЭО ($\mathrm{Im}_{ВЭО}$) на координатной сетке изображения ГСО и выделяем на этой сетке прямоугольник минимального размера, накрывающий изображение ВЭО.

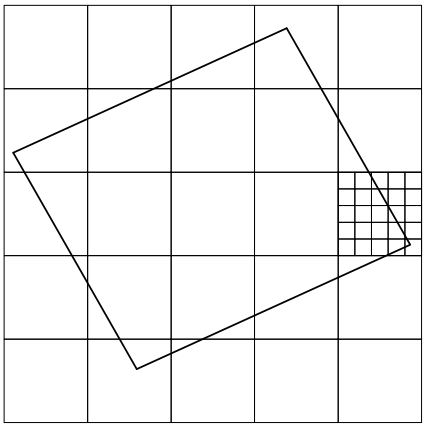


Рис. 1. Пример соотнесения пикселей ВЭО и ГСО

Затем для вычисления яркости в пикселе, отображаем каждый пиксель изображения ГСО ($P_{ГСО}$) на координатную сетку изображения ВЭО. Если $P_{ГСО} \in \mathrm{Im}_{ВЭО}$, то вычисляем яркость, иначе значение яркости не вычисляется.

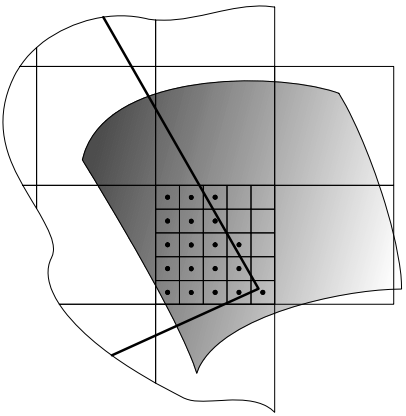


Рис. 2. Разбиение пикселя ВЭО на более мелкую решётку с вычислением значений через сплайн

Рассмотрим рис. 1, те пиксели $P_{ВЭО}$, которые попадают целиком в пиксель ГСО, просто складываем в общую сумму. Остальные рассматриваем

на более подробных решётках $\varepsilon \times \varepsilon$. Для этого натягиваем на соседние точки сплайн рис. 2), вычисляем значения ячеек решётки и считаем среднюю сумму для тех, которые хотя бы частично попали в пиксель ГСО (на рисунке обозначены чёрным точками).

Таким образом, яркость в пикселе ГСО будет иметь следующее значение:

$$\frac{1}{m_{inside}} \sum_{i=1}^{m} P^{i}{}_{B\Theta Ofull} + \frac{1}{n_{inside}} \sum_{j=1}^{n} P^{j}{}_{\varepsilon Spline} .$$

3.3. Алгоритмы для создания панорамы

Для получения панорамы из набора изображений при известной геометрии съемки, необходима только компенсация разницы яркостей на изображениях и краевых эффектов. Но зачастую данные содержат ошибки, поэтому необходимо уточнять параметры съемки. Для этого был использован алгоритм, основанный на выявлении особых точек на изображениях. Между особыми точками на различных изображениях находятся соответствия, затем по этим соответствиям уточняются матрицы гомографии для пар изображений. По уточненным данным изображения склеиваются в панораму, компенсируется разница яркостей разных изображений, и сглаживаются границы склейки.

3.3.1. Алгоритм для вычисления особых точек

Для вычисления особых точек были использованы идеи алгоритма SIFT [5]. Этот подход состоит из 4 шагов:

1. Определение масштабно-пространственных экстремумов: на этом этапе для изображения строится пирамида с её различными масштабами, на каждом уровне которой и находятся особые точки

Для отыскания точечных особенностей была использована функция, получившую название Difference-of-Gaussian (DOG). Необходимо построить целую пирамиду различных масштабов изображения с её применением:

$$D(x, y, \sigma) = (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y) = L(x, y, k\sigma) - L(x, y, \sigma),$$

где $G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2}$ – распределение Гаусса, $I(x, y)$ – исходное изображение, $*$ – операция свертки, а σ – масштаб (степень размытия изображения).

Для уменьшения вычислений использован следующий приём: можно заметить, что уменьшение размеров изображения в два раза приводит к двукратному увеличению её масштаба (то есть σ, с которой она свёрнута). Каждый шаг удвоения σ разбивается на S частей, таким образом, выбираем $k = 2^{1/S}$.

Итак, получен набор разностей изображений с различной степенью размытости (рис. 3).

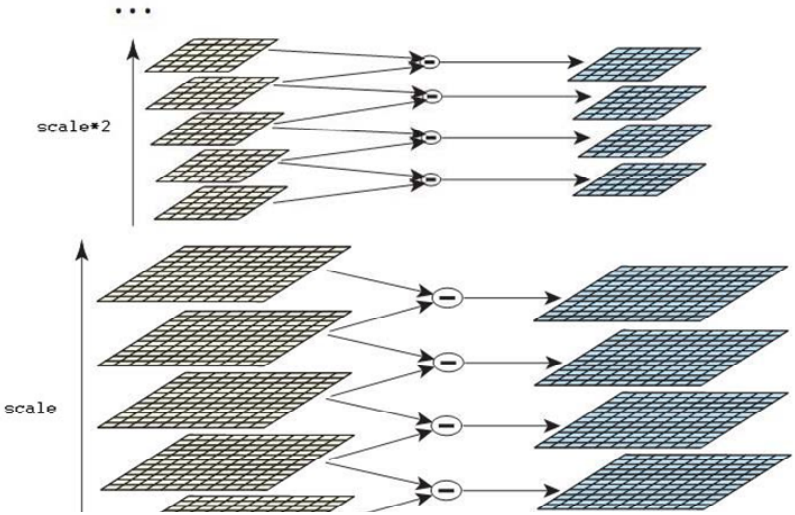


Рис. 3. Изображение последовательно сворачивается с функцией Гаусса, а затем соседние размытые изображения вычитаются друг из друга

2. Локализация особых точек: определяются примерные точки-кандидаты, проверяются на устойчивость, неустойчивые отсеиваются, и для каждой такой точки уточняются ее координаты и масштаб σ.

Для увеличения быстродействия в полученной пирамиде DOG ищутся локальные экстремумы следующим образом. На каждом изображении каждого уровня пирамиды рассматриваются все точки, каждая точка сравнивается с 8 соседними точками на своём уровне и с 9 соседними точками на предыдущем и следующем уровнях (рис. 4). Точка принимается как кандидат в особые точки, если она или больше, или меньше всех своих 26 соседей.

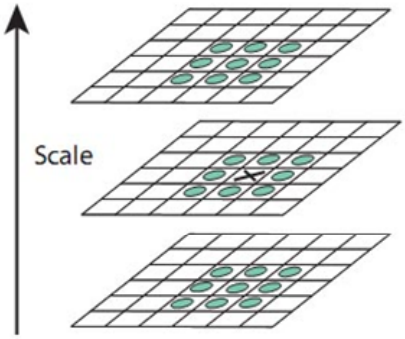


Рис. 4. Поиск локальных экстремумов

Затем найденные кандидаты проверяются дифференциальным путём: если соседняя точка оказывается ближе к локальному экстремуму, то новым кандидатом становится она. Затем отбрасываем точки с низкой контрастностью и большим отношением главной кривизны по X к главной кривизне по Y.

3. Определение направления: определяется градиентное поле вокруг каждой особой точки, выделяется один или несколько векторов основных его направлений, если их оказывается несколько, то для каждого определяется новая особая точка. Вектор основного направление поможет отыскивать совпадения особых точек на различных изображениях, повёрнутых на какой-то угол относительно друг друга.

Все последующие манипуляции производятся с особыми точками, преобразованными в соответствии с вычисленным направлением, масштабом и расположением для каждой особой точки. Это обеспечивает инвариантность относительно подобных преобразований.

4. Вычисление дескрипторов особых точек: для каждой особой точки вычисляем гистограмму направлений градиентного поля вокруг данной точки и поворачиваем её на угол вектора основного направления, вычисленного в предыдущем пункте (рис. 5).

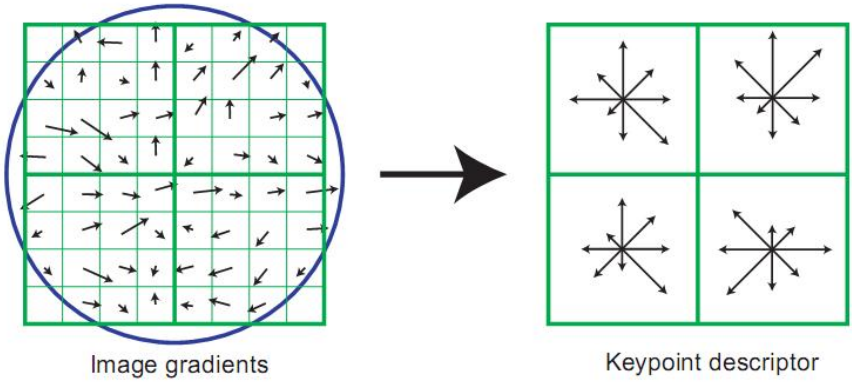


Рис. 5. Дескриптор особой точки

Локальные градиенты вокруг каждой особой точки и масштаб σ обеспечивают устойчивость такого подхода к локальным изменениям формы и яркости изображений.

3.3.2. Алгоритм для поиска ближайших соседей.

Задача поиска ближайших соседей в множестве из N -элементов в пространстве R^k решена при помощи алгоритма Best Bin First (BBF).

В данном алгоритме используется модель k-d деревьев. K-d дерево представляет собой бинарное дерево, которое строится по следующему правилу:

- начальное множество разбивается по значениям векторов в одной из координат, например, по i , где $i = 1,\ldots,k$, на два подмножества;
- i выбирается таким образом, чтобы разброс значений по данной координате был максимальным;
- разбиение проводится по медиане m , так что одинаковое количество точек оказывается с одной и с другой стороны;
- в вершине дерева хранятся значения i , m , разброс значений векторов по каждой координате;
- для полученных вершин процесс повторяется.

В итоге получим бинарное дерево глубины $d = \log_2 N$.

Для поиска n ближайших соседей к вектору q в построенном дереве:

- сначала дерево обходится вниз до листа содержащего «ближайшую» к q точку. Эта точка не обязана быть ближайшей, это только первое приближение;
- во время спуска по дереву заполняется список поддеревьев, которые еще не обходили. Также запоминаются расстояния до них, которое определяется как минимальное расстояние от точки q до любой точки, находящейся в границах значений поддерева;
- из списка выбираем ближайшее к q поддерево и продолжаем поиск в нём;
- расстояние до каждого нового найденного претендента сравнивается с радиусом сферы найденных точек с центром в точке q. Если данное расстояние меньше, то точку на сфере заменяем этим претендентом.

Алгоритм работает до тех пор, пока в списке есть поддеревья с расстоянием, меньшим радиуса сферы найденных точек.

3.3.3. Компенсация отличий в яркости

Изображения, входящие в панораму, могут иметь различную яркость. Вследствие этого на конечной панораме могут появиться области с различной яркостью. Необходимо минимизировать функцию ошибки для разности яркостей в пересекающихся областях изображений [11]. Для этого находим коэффициенты для каждого изображения, на которые необходимо их домножить. Пусть g_i – коэффициент i-го изображения, входящего в панораму. Тогда функцию ошибки представим как сумму:

$$e = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{\substack{u_i \in R(i,j) \\ \tilde{u}_i = H_{ij}\tilde{u}_j}} (g_i I_i(u_i) - g_j I_j(u_j))^2 ,$$

где $R(i,j)$ – область пересечения i-го и j-го изображений. На практике значение $I(u_i)$ аппроксимируется средним значением в пересекающейся области $\overline{I}_{ij}$:

$$\overline{I}_{ij} = \frac{\sum_{u_i \in R(i,j)} I_i(u_i)}{\sum_{u_i \in R(i,j)} 1} .$$

Это упрощает вычисления и дает устойчивость к ложным сопоставлениям особых точек, которые могут возникать ввиду небольших ошибок сопоставления изображений. Чтобы избежать нулевого решения, которое,

очевидно, является оптимальным в данной схеме, добавим дополнительный член, чтобы коэффициенты g_i стремились к единице при совпадении яркости изображений:

$$e = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} N_{ij}\left(\frac{(g_i\overline{I}_i(u_i) - g_j\overline{I}_j(u_j))^2}{\sigma_N^2} + \frac{(1-g_i)^2}{\sigma_g^2}\right),$$

где $N_{ij} = \left|R(i,j)\right|$ – число пикселей в пересечении i-го и j-го изображений. Параметры σ_N и σ_g – стандартное отклонение от нормализованной ошибки интенсивности и ошибки коэффициентов, соответственно. Эта квадратичная функция на вектор параметров $\vec{g}$ может быть разрешена методом приравнивания производной к нулю.

3.3.4. Метод сплайнов с переменной разрешающей способностью

При составлении панорамы необходимо учесть, что входящие в неё изображения могут иметь разную яркость. Из-за этого на конечном изображении будут видны границы склейки.

Для того, чтобы уменьшить этот эффект можно применить метод сплайнов с переменной разрешающей способностью.

Для склейки изображений:

1. Для каждого изображения построим маску

$$W(x) = \begin{cases} 1, r(c_0^i, x) < r(c_0^j, x), \forall j, \\ 0, \text{ иначе} \end{cases},$$

где $r(c_0^i, x)$ – расстояние от центра i изображения до точки x, $j = 1,\ldots,n-1$;

2. Получим набор изображений и масок для разных диапазонов частот;
3. Высокочастотная часть изображения может быть получена следующим образом:

$$B_\sigma^i(x,y) = I^i(x,y) - I_\sigma^i(x,y),$$
$$I_\sigma^i(x,y) = I^i(x,y) * G(x,y,\sigma),$$
$$W_\sigma^i(x,y) = W^i(x,y) * G(x,z,\sigma),$$

где $I^i(x,y)$ – $i = 1,\ldots,n$ изображение, $G(x,y,\sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2}$.

$B_\sigma(x,y)$ представляет пространственную частоту в диапазоне длин волн $\lambda \in [0,\sigma]$

4. Для следующего диапазона частот k:

$$B^i_{(k+1)\sigma}(x,y) = I^i_k(x,y) - I^i_{(k+1)\sigma}(x,y),$$
$$I^i_{(k+1)\sigma}(x,y) = I^i_{k\sigma}(x,y) * G(x,y,\sigma'),$$
$$W^i_{(k+1)\sigma}(x,y) = W^i_{k\sigma}(x,y) * G(x,z,\sigma'),$$

где $\sigma' = \sqrt{(2k+1)}\sigma$.

5. Затем для диапазона пересекающиеся изображения объединяются:

$$I^i_{k\sigma}(x,y) = \frac{\sum_{i=1}^{n} B^i_{k\sigma}(x,y) W^i_{k\sigma}(x,y)}{\sum_{i=1}^{n} W^i_{k\sigma}(x,y)}.$$

Благодаря такому подходу высокочастотные объекты размываются слабее, а низкочастотные сильнее. Таким образом, не теряются мелкие объекты, попавшие в полосу склейки двух изображений.

4. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ И РЕЗУЛЬТАТЫ РАБОТЫ

В результате работы было написано приложение, которое создает панорамы из изображений, содержащих проективные искажения. Для пересчёта яркости после компенсации проективных искажений было реализовано два алгоритма: простой (яркость пикселя ГСО берется из пикселя ВЭО, попавшего в середину пикселя ГСО) и с использованием сплайна.

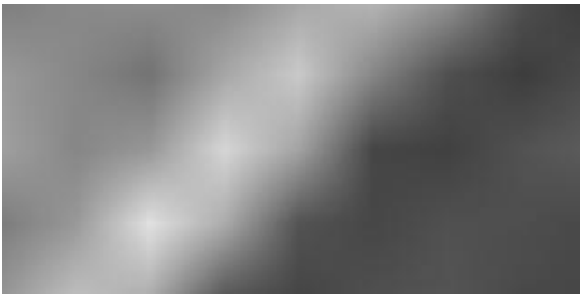


Рис. 6. Изображение с ВЭО, сгенерированное в PovRay

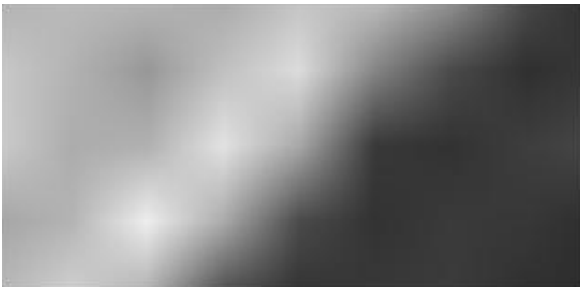


Рис. 7. Результат работы простого алгоритма

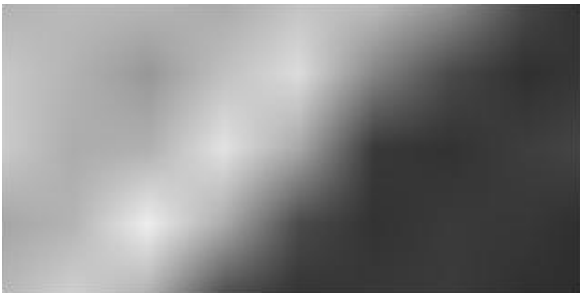


Рис. 8. Результат работы алгоритма с использованием сплайна

Оба алгоритма дают визуально похожие результаты. При подсчете среднеквадратичного отклонения данных двух изображений от исходного получаются следующие величины:

- 4,1501 для простого метода,
- 3,722 для метода с использованием сплайна.

Таким образом, метод с использованием сплайна дает более точный результат.

После компенсации проективных искажений и пересчёта яркости из полученных изображений создаются панорамы. Основной проблемой здесь был выбор алгоритмов сглаживающих границы изображений на панораме. Для сглаживания были выбраны два алгоритма: компенсация яркости и метод сплайнов с переменной разрешающей способностью. Далее приведены результаты работы данных алгоритмов.

Далее представлены результаты работы создания панорам. Если изображения имеют одинаковую яркость, то панорама получается гладкой и границы практически не видны (рис. 9). Но если изображения, которые

склеиваются в панораму, имеют различную яркость, на результирующем изображении границы видны отчётливо (рис. 10).

Для создания панорамы были выбраны три изображения с различной яркостью. Сначала они были соединены без применения алгоритмов сглаживания границ и компенсации яркости. На рис. 10 отчетливо видны границы между изображениями. После применения алгоритма компенсации яркости изображение стало более гладким (рис. 11). Для большей гладкости границ между изображениями используется метод сплайнов с переменной разрешающей способностью, но при применении только его без компенсации яркости результат получается плохим: яркость разных частей изображений сильно отличается (рис. 12). Если изображения сильно отличаются по яркости и, соответственно, имеют ярко-выраженные границы на панораме, наилучшего результата можно добиться, применяя алгоритм компенсации яркости вместе с методом сплайнов с переменной разрешающей способностью (рис. 13). Но при этом теряется четкость объектов на изображении. В случае, когда яркость отличается не сильно, хороший результат даёт применение только алгоритма компенсации яркости.

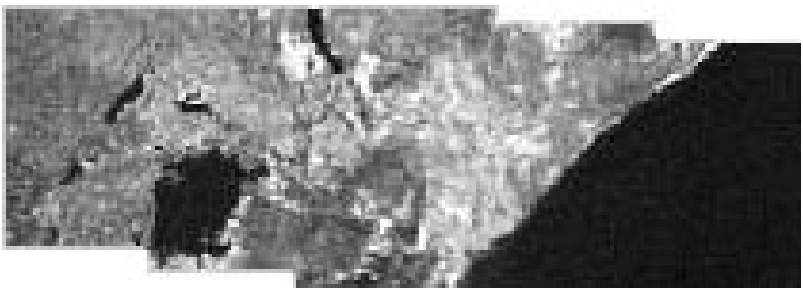


Рис. 9. Панорама из изображений с одинаковой яркостью

Далее приведены результаты работы программы для изображений, смоделированных в PovRay. В PovRay были смоделированы изображения, получаемые с ВЭО. Затем эти изображения были пересчитаны к условиям наблюдения с ГСО. На заключительном этапе из изображений были созданы панорамы с применением описанных методов сглаживания границ. На рис. 14 представлена панорама из трёх изображений, на рис. 15 – из десяти изображений.

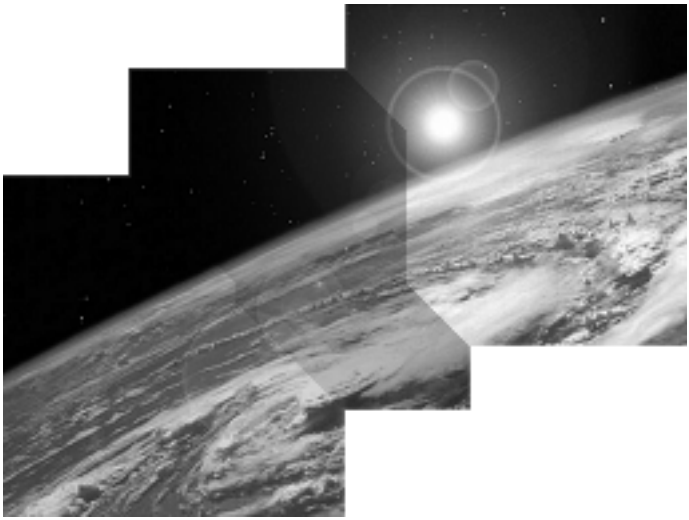


Рис. 10. Склейка изображений без применения алгоритмов сглаживания

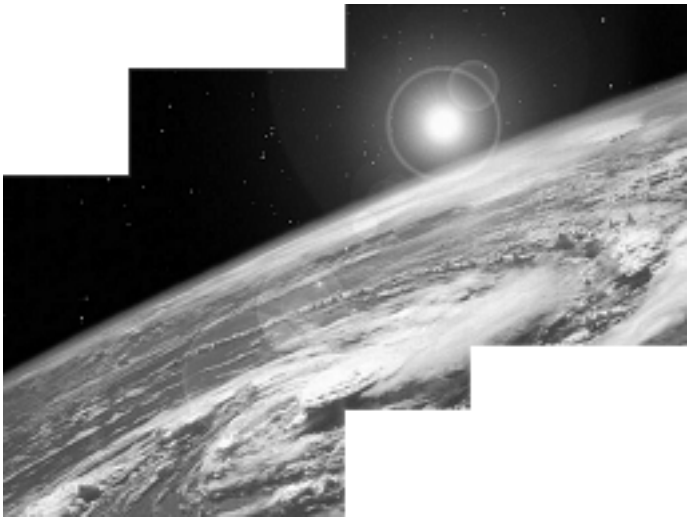


Рис. 11. Склейка изображений с использованием алгоритма компенсации яркости

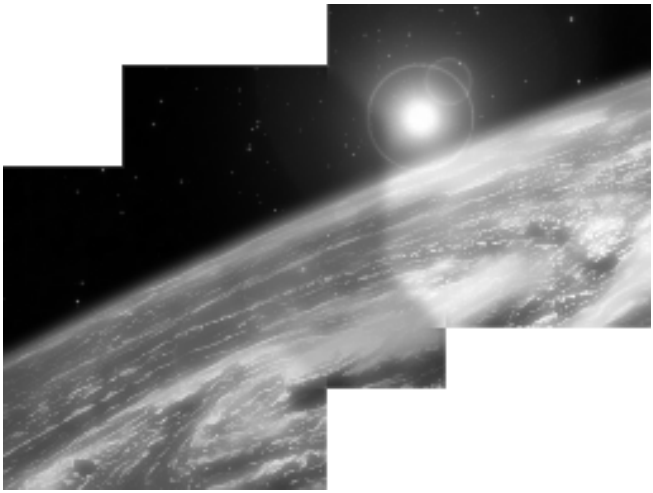


Рис. 12. Склейка изображений с использованием метода сплайнов с переменной разрешающей способностью

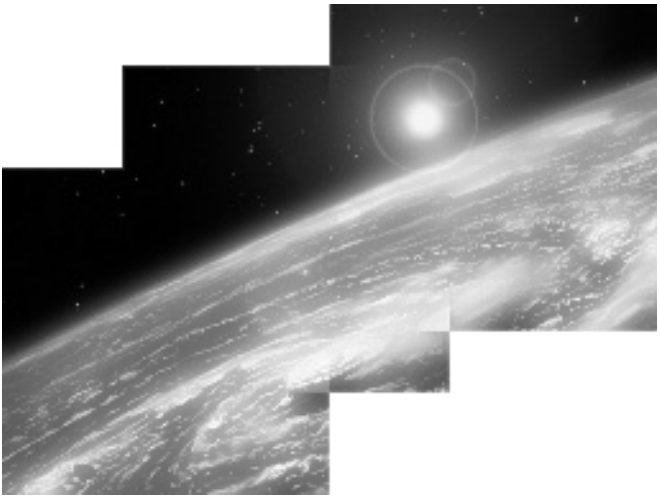


Рис. 13. Склейка изображений с использованием компенсации яркости и метода сплайнов с переменной разрешающей способностью

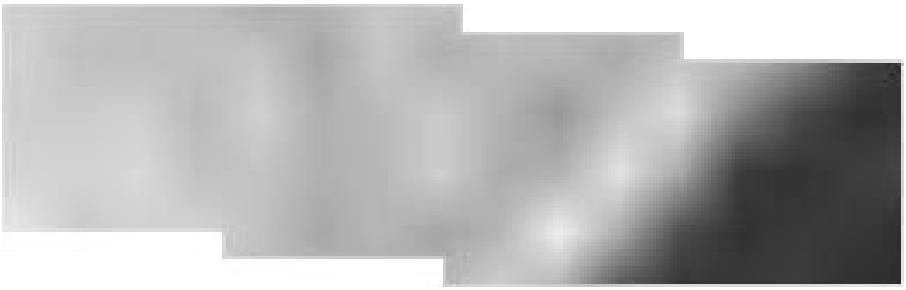


Рис. 14. Панорама из трёх изображений ВЭО, сгенерированных в PovRay

Рис. 15. Панорама из 10 изображений ВЭО, сгенерированных в PovRay

СПИСОК ЛИТЕРАТУРЫ

1. **Анисимов В.А., Курганов В.Д., Злобин В.К.** Распознавание и цифровая обработка изображений. – М.: Высшая школа, 1983.
2. [Электронный ресурс]: Портал программного обеспечения для работы с фотографиями [текст]. Режим доступа: http://photosoft.nnm.ru/.
3. **Szeliski R.** Image Alignment and stitching. – Redmond, WA : Microsoft Research, 2006.
4. **Harris C.** Geometry from visual motion. – MIT Press, 1992.
5. **Lowe D. G.** Distinctive image features from scale-invariant keypoints // International Journal of Computer Vision. – 2004. – Vol. 2.
6. **Friedman J.H., Baskett F., Shustek L.J**. An algorithm for finding nearest neighbors // IEEE Trans. – 1975. – Comput. vol. C-24.
7. **Beis J.S., Lowe D.G.** Shape indexing using approximate nearest-neighbour search in high-dimentional spaces. – Vancouver: University of British Columbia, 1997.
8. **Peleg S.**. Elimination of seams from photomosaics // Computer Graphics and Image Processing, 1981 – Vol. 16.

9. **Burt P.J., Adelson E.H.** A multiresolution spline with application to image mosaics // Transaction on Graphics. – 1983. – Vol. 2.
10. **Brown M., Lowe D.G.** Automatic panoramic image stitching using invariant features. – Vancouver: University of British Columbia, 2007.
11. **Richard H., Zisserman A.** Multiple View Geometry in computer vision. – Cambridge: Cambridge University Press, 2003.
12. **Brown M., Lowe D.G.** Invariant features from interest point groups // British Machine Vision Conference. – Cardiff, 2002. – P. 656–665.
13. **Harris C., Stephens M.** A combined corner and edge detector // Fourth Alvey Vision Conference. – Manchester, 1988. – P. 147–151.
14. **Schmid C., Mohr R.** Local grayvalue invariants for image retrieval // IEEE Trans.on Pattern Analysis and Machine Intelligence. – 1997 – Vol. 19.
15. **Press W.H., Teukolsky S.A., Vetterling W.T., Flannery B.P.** Numerical Recipes in C Second Edition. – Cambridge: Cambridge University Press, 1997.
16. **Киричук В.С., Косых В.П., Перетягин Г.И., Попов С.А.** Использование серии космических снимков с общей точкой наведения для оценивания рельефа местности // Автометрия. – 2002. – Т. 1.

В.С. Денисюк

АЛГОРИТМЫ ВЫДЕЛЕНИЯ ОСОБЕННОСТЕЙ НА ИЗОБРАЖЕНИЯХ С ЦЕЛЬЮ КЛАССИФИКАЦИИ ЗАБОЛЕВАНИЙ РАСТЕНИЙ

1. ВВЕДЕНИЕ

Анализ и классификация изображений – задача, широко востребованная во многих областях. Нередко возникает потребность в обработке и исследовании таких изображений, которые не содержат объектов четко определенной формы. Такие изображения содержат случайно расположенные протяженные фигуры (объекты) разной формы, ориентации и яркости.

Конкретное применение анализа изображений реализовано в системе выделения особенностей с целью диагностики болезней растений.

Рассматриваются статистические и текстурные признаки, которые помогают определить тип заболевания растения. Различные заболевания характеризуются отличающимися друг от друга проявлениями, видными, к примеру, на поверхности листьев растений или срезах клубней. Снимки таких типичных признаков анализируются, накапливается информация о цвете и характере различных пятен, поражений и деформаций. Полученные данные используются в дальнейшем для определения заболевания по изображению.

Для определения границ контрастных объектов может использоваться один из следующих методов: комбинаторный метод или метод порогового градиента, метод выделения контура путем применения оператора Лапласа и фильтра Гаусса, метод, использующий оператор Собеля. Производится классификация точек контура и выделение характерных точек.

Выделение опорных точек будет использоваться для полуавтоматического выделения областей изображения, на которых видны признаки заболевания. Разработан алгоритм нахождения болезни по образцу с помощью анализа гистограмм и текстурных признаков.

Зачастую, чтобы определить конкретное заболевание растения, приходится изучать справочники, классификаторы, просматривать большое количество фотографий пораженных растений, на что уходит достаточно

много времени. Автоматизирование процесса распознавания заболевания было целью, с которой была разработана система.

Разработано два приложения: с помощью одного происходит обучение системы и накопление сведений о болезнях, а посредством второго приложения осуществляется сам процесс распознавания заболевания по снимку.

Система проводит обработку изображений в полуавтоматическом режиме, а пользователю предоставляются средства для удобного выделения интересующих областей изображений. Для выделения особенностей изображения используются алгоритмы выделения контуров и опорных точек, позволяющих пользователю выделить наиболее информативные области и получить достоверную гистограмму, точно характеризующую то или иное заболевание.

Выделение контуров используется как вспомогательный инструмент. Задача состоит в построении изображения именно границ объектов и очертаний однородных областей.

Будем называть контуром изображения совокупность его пикселей, в окрестности которых наблюдается скачкообразное изменение функции яркости. Так как при цифровой обработке изображение представлено как функция целочисленных аргументов, то контуры представляются линиями шириной как минимум в один пиксел.

Если исходное изображение кроме областей постоянной яркости содержит участки с плавно меняющейся яркостью, то введенное определение контура остается справедливым, однако при этом не гарантируется непрерывность контурных линий: разрывы контуров будут наблюдаться в тех местах, где изменение функции яркости не является достаточно резким.

С другой стороны, если на «кусочно-постоянном» изображении присутствует шум, то, возможно, будут обнаружены «лишние» контуры в точках, которые не являются границами областей.

При разработке алгоритмов выделения контуров нужно учитывать указанные особенности поведения контурных линий. Специальная дополнительная обработка выделенных контуров позволяет устранять разрывы и подавлять ложные контурные линии.

Одной из основных характеристик изображения является цвет. С помощью сбора статистической информации о точках можно накопить достаточно данных для последующего анализа. Путём построения и сравнения цветовых гистограмм проводится исследование графических данных. Имея достаточно хорошо обученную систему, можно достаточно достоверно установить, к какому классу относится образец, и по представленному примеру определить болезнь.

Не стоит забывать о других особенностях объектов, таких, как форма и характер их расположения. Для учёта этих параметров используются методы анализа текстур, позволяющие учесть большинство значимых характеристик изображений.

2. ПОЭЛЕМЕНТНАЯ ПРЕДОБРАБОТКА ИЗОБРАЖЕНИЯ

Чтобы улучшить качество получаемых результатов, изображение подвергается некоторой предобработке, предназначенной для того, чтобы убрать шумы, помехи и увеличить четкость изображения. Предобработка проводится поэлементно, то есть преобразованию подвергается каждый пиксель в отдельности.

Подавляющее большинство процедур обработки для получения результата в каждой точке изображения привлекает входные данные из некоторого множества точек исходного изображения, окружающих обрабатываемую точку. Однако имеется группа процедур, где осуществляется так называемая поэлементная обработка. Здесь результат обработки в любой точке кадра зависит только от значения входного изображения в этой же точке. Очевидным достоинством таких процедур является их предельная простота.

Сущность поэлементной обработки изображений сводится к следующему. Пусть $x(i,j) = x_{i,j}$, $y(i,j) = y_{i,j}$ – значения яркости точки, имеющей декартовы координаты i (номер строки) и j (номер столбца) в исходном и получаемом изображениях. Поэлементная обработка означает, что существует функциональная однозначная зависимость между этими яркостями

$$y_{i,j} = f_{i,j}(x_{i,j}), \qquad (2.1)$$

позволяющая по значению исходного сигнала определить значение выходного продукта. В общем случае, как это учтено в данном выражении, вид или параметры функции $f_{i,j}(\cdot)$, описывающей обработку, зависят от текущих координат. При этом обработка является *неоднородной.* Однако в большинстве практически применяемых процедур используется *однородная* поэлементная обработка. В этом случае индексы i и j в выражении (2.1) могут отсутствовать. При этом зависимость между яркостями исходного и обработанного изображений описывается функцией

$$y = f(x), \qquad (2.2)$$

одинаковой для всех точек кадра.

3. ЛИНЕЙНОЕ КОНТРАСТИРОВАНИЕ ИЗОБРАЖЕНИЯ

В качестве рабочего диапазона используется диапазон 0...255; при этом значение 0 соответствует при визуализации уровню черного, а значение 255 – уровню белого. Предположим, что минимальная и максимальная яркости исходного изображения равны $x_{\min}$ и $x_{\max}$ соответственно. Если эти параметры или один из них существенно отличаются от граничных значений яркостного диапазона, то визуализированная картина выглядит как ненасыщенная, неудобная, утомляющая при наблюдении.

При линейном контрастировании используется линейное поэлементное преобразование вида:

$$y = a \cdot x + b, \qquad (2.3)$$

параметры которого a и b определяются желаемыми значениями минимальной $y_{\min}$ и максимальной $y_{\max}$ выходной яркости. Решив систему уравнений

$$\begin{cases} y_{\min} = a \cdot x_{\min} + b \\ y_{\max} = a \cdot x_{\max} + b \end{cases}$$

относительно параметров преобразования a и b, нетрудно привести (2.3) к виду:

$$y = \frac{x - x_{\min}}{x_{\max} - x_{\min}} (y_{\max} - y_{\min}) + y_{\min}.$$

$y_{\max} = 255$. Сравнение двух изображений свидетельствует о значительно лучшем визуальном качестве обработанного изображения. Улучшение связано с использованием после контрастирования полного динамического диапазона экрана, что отсутствует у исходного изображения.

4. СОЛЯРИЗАЦИЯ ИЗОБРАЖЕНИЯ

При данном виде обработки преобразование (2.2) имеет вид (2.1):

$$y = k \cdot x \cdot (x_{\max} - x),$$

где $x_{\max}$ – максимальное значение исходного сигнала, а k – константа, позволяющая управлять динамическим диапазоном преобразованного изображения. Смысл соляризации заключается в том, что белые участки исходного изображения или имеющие уровень яркости близкий к белому, после обработки имеют уровень черного. При этом сохраняют уровень черного и участки, имеющие его на исходном изображении. Уровень же белого на выходе приобретают участки, имеющие на входе средний уровень яркости (уровень серого).

5. КОМБИНАТОРНЫЙ МЕТОД ИЛИ МЕТОД ПОРОГОВОГО ГРАДИЕНТА

При обработке изображений в программе используется система цветовых координат RGB. Цветное изображение размером $n \times m$ задается тремя матрицами $S_R = S_R(i,j)$, $S_G = S_G(i,j)$ и $S_B = S_B(i,j)$, где $0 \le i \le n-1$, $0 \le j \le m-1$. Значения элементов матриц $S_R(i,j)$, $S_G(i,j)$ и $S_B(i,j)$ изменяются в пределах от 0 до 255.

Рассмотрим две точки на изображении $p = (i,j)$ и $p' = (i',j')$, имеющих цвета (r,g,b) и (r',g',b'), соответственно. Это означает, что $S_R(i,j) = r$, $S_G(i,j) = g$, $S_B(i,j) = b$, $S_R(i',j') = r'$, $S_G(i',j') = g'$, $S_B(i',j') = b'$. Для краткости будем писать $S_R(p)$ вместо $S_R(i,j)$, $S_G(p)$ вместо $S_G(i,j)$ и т.д.

Определим цветовое расстояние между точками:

$$cd(p,p') = \max\{\,|S_R(p) - S_R(p')|, |S_G(p) - S_G(p')|, |S_B(p) - S_B(p')|\,\}.$$

Евклидова метрика определяется следующим образом:

$$\rho(p,p') = \sqrt{(i-i')^2 + (j-j')^2}\ .$$

В ситуациях, когда время исполнения критично, используется квадрат евклидовой метрики, поскольку он всегда целочисленный, а вычисления с целыми числами выполняются намного быстрее вычислений с числами с плавающей запятой.

В процессе сканирования изображения, а также после некоторых преобразований (например, после фильтрации), некоторые цвета могут измениться. Цвета, которые прежде были идентичны, станут разными. В то же время обычно эти цветовые изменения невелики. Поэтому в дальнейшем мы будем использовать специальную константу, которая называется цветовой константой и обозначается C_V. Цветовая константа определяет порог

цветового расстояния между двумя цветами, ниже которого эти цвета считаются идентичными. На языке формул, если

$$cd(p, p') \le C_V ,$$

то считается, что точки p и p' имеют один и тот же цвет.

Будем обозначать через $B_n(p) = B_n(i, j)$ квадрат размером $n \times n$ с центром в точке $p = (i, j)$, где n – нечетное.

6. ВЫДЕЛЕНИЕ КОНТУРОВ

Пусть p', p'' – две точки исследуемого изображения,

$$S(p') = (r', g', b') ,\ S(p'') = (r'', g'', b'') .$$

Пусть теперь

$$cd[B_m(p)] = \max\left\{cd(p', p'') : p', p'' \in B_m(p)\right\}$$

– максимальное цветовое расстояние в окрестности исследуемой точки.

При каждой итерации алгоритма вычисляются цветовые расстояния для всех возможных пар точек исследуемой окрестности, затем выбирается максимальное значение расстояния.

Теперь

$$f(p) = \begin{cases} 0, \ if\ cd[B_m(p)] \ge C_V , \\ 1, if\ cd[B_m(p)] < C_V . \end{cases}$$

– если цветовое расстояние превышает заданный порог, то через рассматриваемую точку проходит контур. Заключение о том, проходит контур через точку или нет, фиксируется соответствующими значениями функции f.

7. МЕТОД ВЫДЕЛЕНИЯ КОНТУРА ПУТЕМ ПРИМЕНЕНИЯ ОПЕРАТОРА ЛАПЛАСА И ФИЛЬТРА ГАУССА

Основная идея состоит в следующем: размытие реализуется посредством применения операторов на основе распределения Гаусса. Выделение контура объекта производится с помощью дискретного аналога оператора Лапласа.

Композиция вышеуказанных операторов может быть представлена в форме скалярного произведения с некоторой матрицей. Такое скалярное произведение называется сверткой, а матрица называется маской.

8. ОПЕРАТОР СОБЕЛЯ

Оператор Собеля производит измерение градиента на двумерном пространстве – изображении. Чаще всего он используется для нахождения приблизительного абсолютного значения градиента в каждой точке входного полутонового изображения. В детекторе краев Собеля используется пара матриц (масок) размером 3 на 3 пикселя. Одна из них используется для подсчета градиента в горизонтальном направлении, другая – по вертикали. Матрица поочередно действует на квадратную область текущего окна на изображении $\boldsymbol{A}$.

Вышеупомянутые матрицы:

$$\mathbf{G_x} = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * \mathbf{A}, \qquad \mathbf{G_y} = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \mathbf{A}$$

Значение градиента затем вычисляется по формуле:

$$\mathbf{G} = \sqrt{\mathbf{G_x}^2 + \mathbf{G_y}^2}.$$

9. ВЫДЕЛЕНИЕ ОПОРНЫХ ТОЧЕК

Выделяя контур объекта, можно получить информацию о его геометрии. Используя большее или меньшее число точек для разметки, мы осуществляем более или менее детальный анализ изображения.

Пусть есть точка с координатами $p = (i, j)$ $0 \le i \le n-1$, $0 \le j \le m-1$. Считаем, что мы имеем полутоновое изображение, заданное с помощью функции яркости $S : M^2 \to R$, которая ставит в соответствие каждой точке изображения значение её яркости. R – в данном случае целые числа от 0 до 255

На первом этапе производится преобразование полутонового изображения в логическую матрицу $T = (t_{ij})$.

Полагаем

$$t_{ij} = \begin{cases} 0, & S(i,j) \ge \tau \\ 1, & S(i,j) < \tau \end{cases}$$

где τ – некоторый порог яркости.

Далее выделяем контур, оставляя только граничные точки и стирая все внутренние. Если точка $\langle i, j \rangle$ является граничной, то

$$t_{i,j+1} + t_{i,j-1} + t_{i+1,j} + t_{i-1,j} < 4 .$$

Для того чтобы описать методы выделения характерных точек на контуре, введем ряд определений.

Пусть дана точка $\langle i, j \rangle$, тогда возникают различные типы окрестностей:

$S_4(i,j) = \{\langle i \pm 1, j \rangle, \langle i, j \pm 1 \rangle\}$ – 4–окрестность,

$S_D(i,j) = \{\langle i \pm 1, j \pm 1 \rangle\}$ – D–окрестность,

$S_8(i,j) = S_4(i,j) \cup S_D(i,j)$ – 8–окрестность.

Пронумеруем вокруг данной точки все элементы. Предположим, что они расположены на окружности с центром в этой точке, находятся на одинаковом угловом расстоянии $\pi/4$ и последовательно пронумерованы по часовой стрелке, начиная с точки, являющейся верхней точкой этой окружности. Вместо t_{kl} будем писать t_p, где p – соответствующий номер, $1 \le p \le 8$ в случае, когда $\langle k, l \rangle$ попадает в данное окружение точки.

Введем функции:

1) количество единичных точек в S_8 и S_4:

$$A_8(i,j) = \sum_{k=1}^{8} t_k, \quad A_4(i,j) = \sum_{k=1}^{4} t_{2k-1} ;$$

2) количество единичных троек в S_8:

$$C_8(i,j) = \sum_{k=1}^{8} t_{2k-1} t_{2k} t_{2k+1} ,$$

индексы суммируются по $\bmod 8$;

3) число 8-связности:

$$N_{C_8} = A_8(i,j) - C_8(i,j) ;$$

4) число 4-связности:

$$N_{C_4} = A_4(i,j) - C_4(i,j) .$$

Тогда через введенное число связности определяются характерные точки контура:

$$N_C(i,j) = \begin{cases} 0, & \textit{изолированная точка}; \\ 1, & \textit{конечная точка}; \\ 2, & \textit{связующая точка}; \\ 3, & \textit{точка ветвления}; \\ 4, & \textit{точка пересечения}, \end{cases}$$

где под $N_C(i,j)$ понимается N_{C_4}, N_{C_8} в зависимости от выбранного типа связности.

Ясно, что точки $\langle i,j \rangle$ такие, что $N_C(i,j) \neq 2$, являются информативными, т.е. их необходимо выделить при разметке объекта. Условие $N_C(i,j) = 2$ ничего не дает. Здесь необходимы другие критерии. Если контур имеет излом в данной точке, то ее целесообразно выделить. В противном случае это делать нежелательно.

Для разметки контура используем только те элементы, в которых меняется направление движения. Нетрудно видеть, что если мы имеем, например, прямоугольник, то после применения алгоритма останутся всего четыре угловых точки. Однако для геометрически сложных кривых может оказаться слишком много точек, в которых меняется направление. В этом случае ряд точек может быть отсеян, исходя из различных критериев.

10. СТАТИСТИЧЕСКИЙ АНАЛИЗ ИЗОБРАЖЕНИЙ. ЦВЕТОВЫЕ ГИСТОГРАММЫ

Метод цветовых гистограмм – наиболее популярный из методов, использующих цветовые характеристики для индексирования изображений. Возможно также использование таких показателей, как средний или основной цвета, а также множества цветов; эти характеристики имеет смысл использовать для локального индексирования областей изображения

Идея метода цветовых гистограмм для индексирования и сравнения изображений сводится к следующему. Все множество цветов разбивается на набор непересекающихся, полностью покрывающих его подмножеств V_i, $0 \le i < N$. Будем называть такое разбиение множества цветов базовой палитрой. Для изображения формируется гистограмма, отражающая долю каждого подмножества цветов в общей цветовой гамме изображения – массив

$H_i = N_i / \Sigma N_i$, где N_i – число точек с цветом из множества V_i. Для сравнения гистограмм вводится понятие расстояния между ними. Известны различные способы построения и сравнения цветовых гистограмм, отличающиеся между собой изначальной цветовой схемой (RGB, CMY, HSV, grayscale и т. д.), размерностью гистограммы и определением расстояния между гистограммами.

В данной работе реализовано несколько модификаций метода, использующих разные способы квантования множества цветов и вычисления расстояния между гистограммами. Используются две базовые палитры и, следовательно, два метода построения гистограммы.

11. РАЗБИЕНИЕ RGB-ЦВЕТОВ ПО ЯРКОСТИ.

В базовой палитре V_i ($0 \le i < N$) определяется как множество цветов C : $C \in V_i$, где $i / N * I_{max} \le I(C) < (i+1) / N * I_{max}$, где I(C) – интенсивность цвета C, нормализованная так, что $0 \le I(C) < I_{max}$. Интенсивность вычисляется по классической формуле:

$$I(C)=0.3*R(C)+0.59*G(C)+0.11*B(C),$$

где R, G и B – красная, зеленая и синяя компоненты цвета C. Imax=256; $0 \le I(C) < 256$. В частности, для черно-белых полутоновых изображений на N подмножеств разбивается исходное множество оттенков. Значение N выбиралось практически произвольно, сейчас установлено N=16.

Для сравнения гистограмм вводится понятие расстояния между ними – сумма модулей разности соответствующих элементов гистограмм. Некоторое усовершенствование метода достигается при вычислении расстояния на основании поэлементного сравнения гистограмм с учетом соседних элементов. Для каждого элемента гистограммы первого изображения вычисляется не одна, а три разности:

$$R_1(i)=\left| H_1[i] - H_2[i-1] \right|$$
$$R_2(i)=\left| H_1[i] - H_2[i] \right|$$
$$R_3(i)=\left| H_1[i] - H_2[i+1] \right|$$

(для i = 0 и i = N вместо невычислимых разностей подставляются заведомо большие значения), итоговое же расстояние равно:

$$S = \sum_{i=0}^{N-1} \min_{1<=k<=3} (R_k(i)),$$

Этот способ не годится для произвольной базовой палитры, т. к. предполагает строгую упорядоченность множества цветов, как в случае с разбиением по яркости. Заметим, что так определенное S не является расстоянием в математическом смысле из-за несимметричности (нельзя гарантировать, что $S(H_1,H_2)= S(H_2,H_1)$). Основное преимущество алгоритма состоит в том, что он слабо чувствителен к изменению освещенности, что ощутимо улучшает результаты его применения на широком классе изображений.

Этот метод построения гистограмм наиболее эффективен для черно-белых полутоновых изображений. Для цветных RGB-изображений лучшие результаты дает другой способ.

12. РАЗБИЕНИЕ RGB-ЦВЕТОВ ПО ПРЯМОУГОЛЬНЫМ ПАРАЛЛЕЛЕПИПЕДАМ

Цветовое RGB-пространство рассматривается как трехмерный куб, каждая ось которого соответствует одному из трех основных цветов (красному, зеленому или синему), деления на осях пронумерованы от 0 до 255 (большее значение соответствует большей интенсивности цвета). При таком рассмотрении любой цвет RGB-изображения может быть представлен точкой куба. Для построения цветовой гистограммы каждая сторона делится на n (n=4) равных интервалов, соответственно, RGB-куб делится на N (N=64) прямоугольных параллелепипедов. V_i – множество цветов, все компоненты которых попадают в определенные интервалы. Гистограмма изображения отражает распределение точек RGB-пространства, соответствующих цветам пикселей изображения, по параллелепипедам.

Выбор размерности гистограммы определялся из следующих соображений. При n = 2 (N = 8) считались бы одинаковыми, например, {126, 128, 126} и {0, 255, 0}, что, естественно, недопустимо. Установка n = 8 (N = 512) приводит к тому, что базовая палитра становится более строгой, чем 8-битная. Такая точность не только автоматически дает некорректную обработку 256-цветных изображений, но и на остальных изображениях приводит к неестественным результатам. Очевидно, что при росте n ситуация только ухудшается. Поэтому было установлено n=4.

В качестве расстояния между гистограммами используется покомпонентная сумма модулей разности между ними. Несмотря на предельную

простоту подхода, он показывает довольно стабильные результаты. Распознаются схожие по цветовой гамме изображения.

13. ТЕКСТУРНЫЙ АНАЛИЗ

При анализе изображений важной их характеристикой служит текстура, которая присутствует во всех изображениях, начиная с изображений, получаемых с помощью самолетных и спутниковых устройств и заканчивая микроскопическими изображениями в биомедицинских исследованиях.

Один из аспектов текстуры связан с пространственным распределением и пространственной взаимозависимостью значений яркости локальной области изображения. Статистики пространственной взаимозависимости значений яркости вычисляются по матрицам переходов значений яркости между ближайшими соседними точками.

Матрица смежности (или матрица совместной встречаемости) уровней яркости представляет собой оценку плотности распределения вероятностей второго порядка, полученную по изображению в предположении, что плотность вероятности зависит лишь от расположения двух пикселей. Совершенно очевидно, что такие матрицы содержат информацию, характеризующую текстуру. По матрице совместной встречаемости вычисляется около двадцати признаков, такие как степень однородности, максимальная вероятность, контраст и другие.

Вычисляя признаки для различных изображений, можно получить многомерный вектор признаков текстур. Такие признаки четко связаны с визуальными особенностями текстуры и используются для поиска схожих изображений.

СПИСОК ЛИТЕРАТУРЫ

1. **Братцев С.Г., Мурзин Ф.А., Нартов Б.К., Пунтус А.А.** Конфликт сложных систем. Модели и управление. – М.: Изд-во МАИ, 1995. – 120 с.
2. **Грузман И.С., Киричук В.С., Косых В.П., Перетягин Г.И., Спектор А.А.** Цифровая обработка изображений в информационных системах: Учебное пособие. – Новосибирск: Изд-во НГТУ, 2000. – С. 22–27.
3. **Харалик Р. М.** Статистический и структурный подходы к описанию текстур – ТИИРЭ 5, 1979. – С. 98–118.
4. **Мишулина О.А., Тхей В.** Признаки корреляционного типа в системе распознавания текстурных изображений // Научная сессия МИФИ-2007. – М.: МИФИ, 2007. – Т. 2. – С. 17–18.

Р. И. Идрисов

ПРОТЯГИВАНИЕ КОНСТАНТ В ГРАФЕ IR2 ВНУТРЕННЕГО ПРЕДСТАВЛЕНИЯ ЯЗЫКА SISAL

ВВЕДЕНИЕ

Потоковый язык SISAL 3.2 [1], разрабатываемый в Институте систем информатики им. А. П. Ершова, является функциональным языком однократного присваивания. Основная часть оптимизаций в компиляторе языка SISAL выполняется на уровне представления IR2, которое является иерархическим потоковым бесконтурным графом программы. Представление состоит из вершин и портов, где вершины обозначают операции, а порты используются для обозначения начала и окончания дуги передачи значения между операциями. В данной статье пойдёт речь о протягивании значений, алгоритме оптимизации, направленном на подстановку значений, которые могут быть вычислены до исполнения программы. В контексте компилятора Sisal алгоритмы протягивания констант используются не только как оптимизирующие преобразования, а ещё и как алгоритмы анализа, которые направлены на определение возможных значений, известных на стадии компиляции программы. Эта информация используется другими оптимизирующими преобразованиями компилятора. Речь пойдёт не только о случае, когда значение может быть однозначно определено, но и когда есть некоторый набор или диапазон возможных значений. Названия глав данной статьи соответствуют характеру распространяемой информации.

1. ПРОТЯГИВАНИЕ ЗНАЧЕНИЙ

Этот алгоритм является самым простым из рассматриваемых. Он направлен на вычисление константных значений, которые могут быть получены до компиляции.

Алгоритм перебирает порты графа IR2 в порядке вычисления операндов и каждому из портов сопоставляет вычисленное значение переменной, если таковое может быть найдено. Значение может быть найдено, если:

- порт является входным, а соответствующий выходной порт принимает константное значение;

- порт является выходным, вершина соответствует простой операции, и входные порты этой вершины также определены. Под простой операцией подразумевается функция, которая, действуя на константные операнды, возвращает значение целого или булевского типа, например: сложить, вычесть, умножить.

Этот алгоритм может не найти некоторых константных значений, реально возникающих при исполнении программы.

Утверждение: подстановка значения, обнаруженного таким образом, является эквивалентным преобразованием программы.

Доказательство: если подстановка значения преобразует программу в неэквивалентную, значит, существует набор данных, для которого результат исполнения программы отличается от результата исполнения исходной программы. Пусть A_i – такой набор входных данных, B_k и B_k' – результаты программ до и после подстановки, а K_j – набор подставленных значений. Выделим подставленные значения как входные параметры программы. В таком случае мы получаем две программы с эквивалентными графами вычислений. Эти программы на наборе входных данных (A_i, K_j) должны приводить к одинаковому результату, что противоречит нашему предположению о существовании набора данных A_i, приводящего к различным результатам вычислений.

Особые случаи возникают при рассмотрении сложных вершин, имеющих неявную внутреннюю структуру (select, for). В этих случаях информация распространяется по неявным связям внутри составной вершины и объединяется в случае нескольких возможных путей исполнения, но эта особенность не влияет на доказательство рассмотренного утверждения.

Рассмотрим пример:

```
function ts_411(returns integer)
  let
 i:=1
  in
   if i>0 then 1
   else 0
   end if
  end let
end function
```

В этом случае граф IR2 будет выглядеть так:

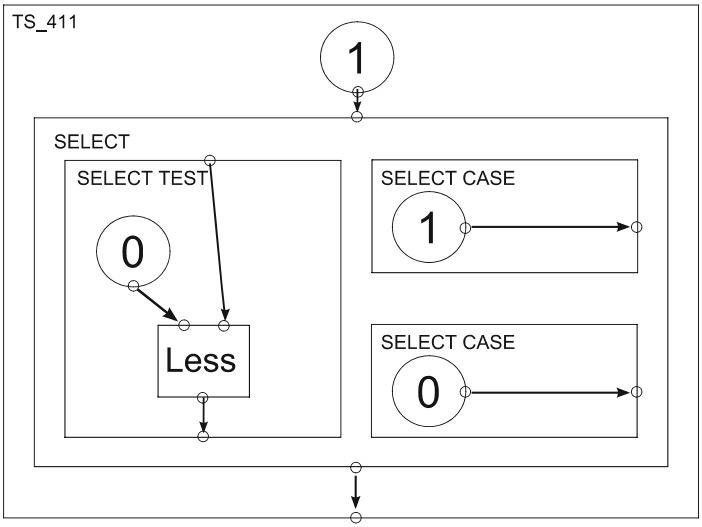


Рис. 1. Граф IR2 для функции ts_411

В данном случае. Информация о константном значении условия может показать компилятору, что в коде функции много лишнего «мёртвого» кода, рассмотрим на данном примере распространение информации о константных значениях. В процессе анализа такая информация определяется для каждой дуги подграфа. После того, как определено значение выходного порта подграфа Select Test, можно утверждать, что всегда будет выполняться только первая ветвь Select Case, а функция будет иметь константное выходное значение (рис. 1).

Информация о том, что функция принимает константное значение, должна быть выражена в терминах графа IR2. Для того, чтобы она могла быть использована любыми другими алгоритмами анализа или оптимизации. Для этого в граф добавляются вершины типа literal (константа) там, где значение постоянное (рис. 2). Исключение составляют те рёбра, которые идут из вершин типа literal, поскольку такое преобразование для них бессмысленно. Граф после подстановки констант можно увидеть на рис. 3. Константа булевского типа со значением «истина» (true) обозначена буквой «T».

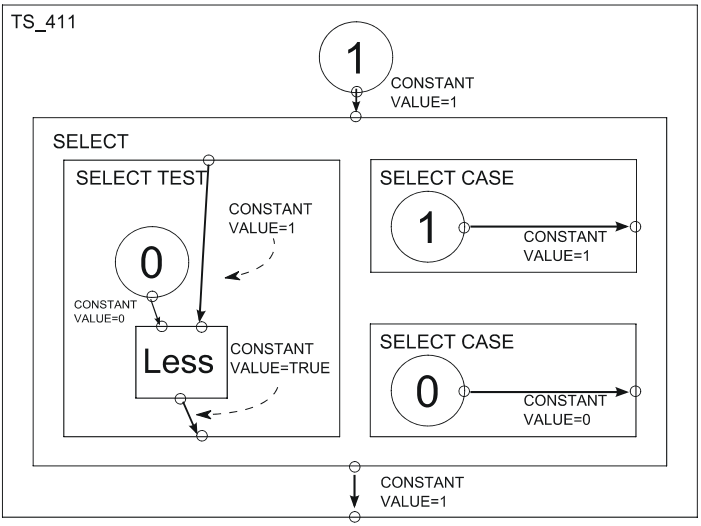


Рис. 2. Граф функции, дополненный алгоритмом анализа

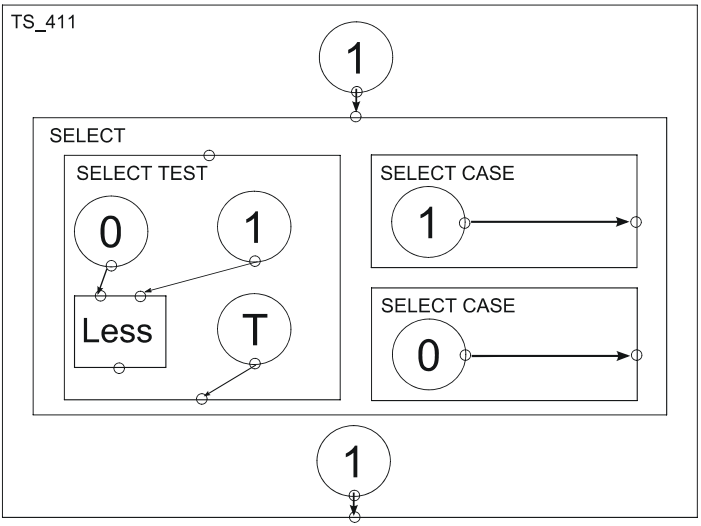


Рис. 3. Граф функции после подстановки константных значений

Этот алгоритм анализа имеет сложность, линейно зависящую от числа вершин графа функции O(N), поскольку для каждой вершины выполняется конечное количество операций, которое не зависит от общего числа вершин и других параметров программы, а является лишь особенностью реализации алгоритма.

2. ПРОТЯГИВАНИЕ МУЛЬТИЗНАЧЕНИЙ

Этот алгоритм отличается от предыдущего представлением данных. Здесь распространяется информация о множестве возможных значений, которое может быть задано при помощи перечисления. Имеется в виду, что здесь обрабатываются перечислимые конечные множества целых чисел.

Рассмотрим пример:

```
function ts_412(i : integer returns integer)
   let j := if i>0 then 1
     else 0
     end if
   in
     if j<2 then 1
     else 0
     end if
   end let
end function
```

В этом случае предыдущий алгоритм не сработает, поскольку нельзя сказать, что значение j всегда одно, но код очевидно «мёртвый», поскольку второе условие всегда истинно.

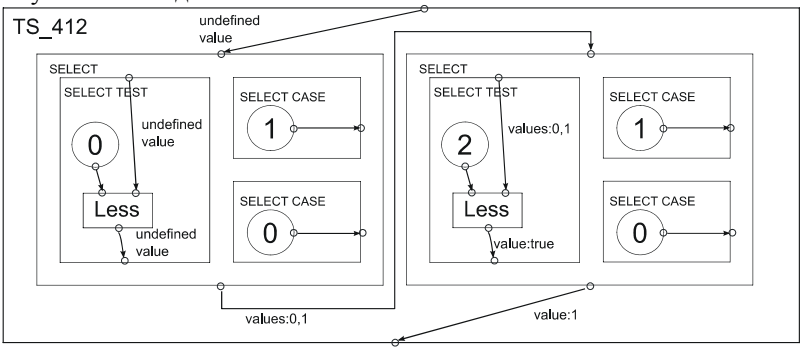


Рис. 4. Протягивание мультизначений

Подстановка производится только в том случае, когда полученное значение оказывается единственным, а записи о множестве возможных значений могут быть использованы другими оптимизирующими алгоритмами компилятора.

Количество возможных значений в данном случае ограничено количеством вершин типа literal (константа) с различным значением в графе программы, это число может стремиться к общему числу вершин графа программы. Сложность вычисления простой операции для множественных значений пропорционально их количеству. Таким образом, сложность алгоритма ограничена $O(N^2)$, где N – общее число вершин графового представления программы. Это оценка сверху; в реальных же случаях общее количество ветвлений программы ограничено константой и практически не зависит от общего числа вершин; если это предположение верно, сложность алгоритма – O(N).

3. ПРОТЯГИВАНИЕ ДИАПАЗОНОВ

Этот алгоритм отличается от предыдущего также представлением данных. Здесь, кроме константных значений, могут фигурировать интервалы значений, но не множество значений.

Информация о возможных значениях переменной представляет собой интервал [a;b] либо может быть не определена в случае, если область значений не может быть вычислена в процессе компиляции, либо не может быть записана в виде одиночного интервала такого типа.

Алгоритм анализа практически остаётся без изменений за исключением того, что вершины, порождающие множества значений (например, Scatter) теперь могут служить источником информации для алгоритма анализа. Конечно, такую возможность можно было реализовать и в рамках предыдущего алгоритма, но это не имело бы особого смысла, поскольку по точности он бы был таким же, как алгоритм, описываемый в следующем разделе (протягивание мультидиапазонов), но по скорости работы и объёму требуемых ресурсов явно уступал бы ему.

В данном случае требуется доопределение простых операций на операнды, представленные не одиночными значениями, а интервалами. Например, вычитание двух значений с возможными диапазонами [10..20] и [3..7] даст значение с возможным диапазоном [3..17].

Рассмотрим пример программы:

```
function ts_413(returns array[integer])
   for i in 1, 100 repeat
   s:=if i>0 then 1
     else 0
     end if
   returns array of s
   end for
end function
```

В этом случае предыдущие алгоритмы не нашли бы избыточный код, который здесь, очевидно, присутствует.

Сложность этого алгоритма также O(N), поскольку количество операций, выполняемое для каждой из вершин графа, ограничено константой, не зависящей от общего числа вершин (N).

4. ПРОТЯГИВАНИЕ МУЛЬТИДИАПАЗОНОВ

Протягивание мультидиапазонов даёт уточнение анализа в несколько экзотических случаях. Например, это происходит, когда циклическая переменная изменяется по набору интервалов или имеется ветвление в программе, которое различным образом определяет массив, используемый далее в программе. Сам факт уточнения данных в процессе анализа без реальной возможности это использовать не представляет пользы. Например, может встретиться условие, которое выполняется только при значениях, находящихся между определёнными диапазонами (предыдущий алгоритм из-за объединения интервалов в один не мог сохранить эту информацию).

Рассмотрим следующую программу:

```
function ts_414(returns array[integer])
 s1:=for I in 1, 10 returns stream of I;
 s2:=for I in 30,40 returns stream of I;
 s3:=s1 || s2;
 for I in s3 repeat
  s:=if I=20 then 1
        else 0
        end if
 returns sum of s
 end for
end function
```

В этом примере код не менее очевидно является избыточным, но установить это при помощи предыдущих алгоритмов анализа не представляется возможным.

Сложность этого алгоритма, как и в случае мультизначений, ограничена $O(N^2)$, здесь источниками диапазонов могут служить вершины типа scatter, количество которых ограничено сверху общим числом вершин графа программы N.

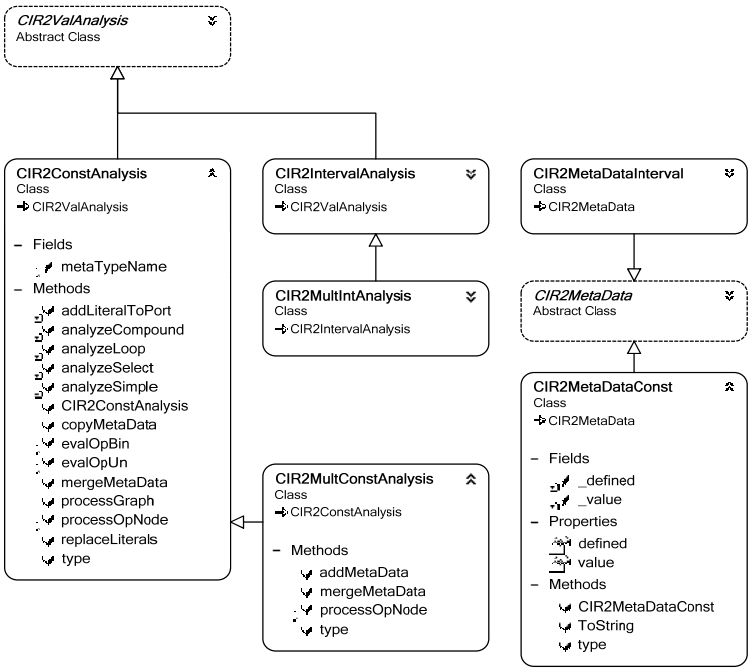


Рис. 5. Структура объектов, осуществляющих анализ

Такое большое количество алгоритмов, принципиально отличающихся только лишь представлением данных, нужно для проведения различных тестов и исследований, на которые и направлена система SFP, частью которой является компилятор Sisal.

Реализация четырёх рассмотренных алгоритмов анализа выполнена таким образом, что для добавления каждого следующего класса требовалось изменить только небольшие методы.

На рис. 5 изображены классы преобразований и классы метаданных, которые при этом используются. Классы для анализа значений и мультизначений используют один и тот же тип метаданных, только с тем отличием, что в случае одиночных значений такая запись может содержаться только одна для любого порта IR2. Для классов множественных значений и диапазонов потребовалось изменить алгоритм добавления метаданных порта IR2, алгоритм слияния значений метаданных и алгоритм обработки простой операции (преобразующей известные значения в известные).

5. ПРОТЯГИВАНИЕ ДРУГОЙ ИНФОРМАЦИИ О ЗНАЧЕНИЯХ

Когда одно из значений, составляющих выражение, является неопределённым, рассмотренные алгоритмы не будут срабатывать, но в случае сокращения неизвестного значения, общее выражение может быть вычислено до исполнения программы.

Рассмотрим следующий пример:

```
function ts_415(n: integer returns integer)
  let
 i:=n+1
 k:=n+5
  in
   if i>k then 1
   else 0
   end if
  end let
end function
```

В этом примере избыточный код можно обнаружить при помощи прямой подстановки (forward substitution). Несложно придумать пример, где такой способ не сработает:

```
function ts_4151(n: integer returns integer)
let
  i,k:=if n>0 then
     n+1,
     n+5
       else
     n+2,
     n+4
```

```
        end if
in
  if i>k then 1
  else 0
  end if
end let
end function
```

В данном случае прямая подстановка в чистом виде ничего не даст, если только не будет скомбинирована с какими-нибудь более сложными алгоритмами.

Для этого примера можно ограничиться алгоритмом протягивания значений, только в качестве значения использовать тройку *связанная переменная, операция связи, константа*. Это очень похоже на способ описания областей массивов при помощи триплетов, с той разницей, что здесь в качестве связанных переменных выступают не итераторы циклов, а просто некоторые переменные программы.

В этом случае информация распространится следующим образом:

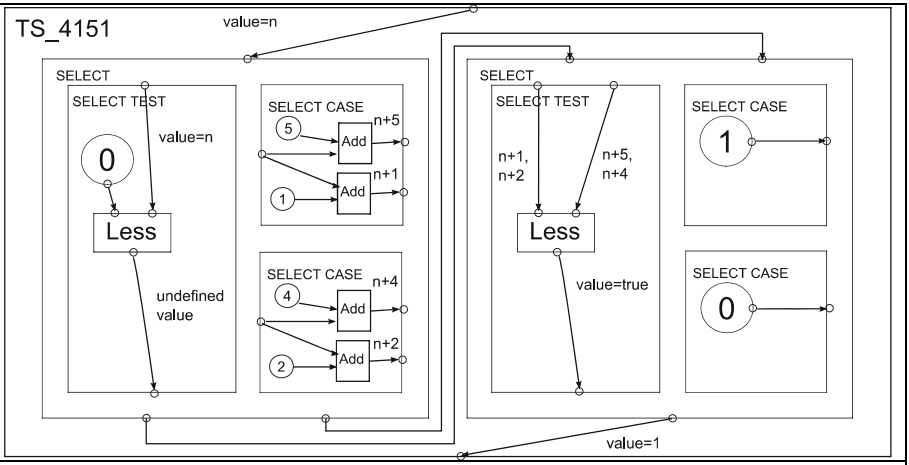


Рис. 6. Протягивание информации о связанной переменной

Оценив действие алгоритма, несложно модифицировать пример так, чтобы программа сохранила свой смысл, но алгоритм не срабатывал:

```
function ts_4151(n: integer returns integer)
let
  i,k:=if n>0 then
     n+1,
     n+5
       else
     1,
     5
       end if
in
  if i>k then 1
  else 0
  end if
end let
end function
```

В этом случае уже не получится обойтись этим же алгоритмом с модифицированным представлением данных, потребуется анализ с учётом пути исполнения. Отметим также, что этим примером неидеальности алгоритма не заканчиваются; значение может зависеть от двух или трёх переменных, и в процессе его вычисления могут встречаться произвольные функции.

Сформулируем общий критерий: подстановку константы можно совершить на стадии компиляции тогда и только тогда, когда может быть однозначно вычислен статический срез по переменной подстановки. Здесь срез – это подмножество операторов и выражений исходной программы, которые прямо или косвенно влияют на значения, вычисляемые в точке критерия среза (в нашем случае это значения, для которых требуется определить возможность их подстановки на стадии компиляции), но не обязательно составляют исполняемую программу [3].

Становится ясно, что для более точного поиска инвариантных значений можно воспользоваться алгоритмами вычисления минимального статического среза.

Для построения однопроцедурных статических срезов программ задача сводится к нахождению всех достижимых вершин из точки критерия среза в графе программных зависимостей, что является верным и для графа IR2, поскольку операторы, вычисляемые в процессе получения значения, соответствуют операторным вершинам, достижимым при прохождении графа IR2 в направлении, обратном выполнению вычислений.

Построение статического среза программы, представленной в виде графа программных зависимостей, является задачей линейной сложности. Срез должен быть вычислен для каждого значения в программе, после чего по-

требуется дать заключение о возможности его вычисления до исполнения программы. Таким образом, сложность алгоритма O(V*(N+D)), где V количество значений, для которых требуется произвести вычисление среза, а N и D количество вершин и рёбер графа IR2. В случае, когда срез не является однозначно вычислимым, представление возможных значений в виде среза не является удобным для других алгоритмов оптимизации и вряд ли может быть использовано повторно.

ЗАКЛЮЧЕНИЕ

Алгоритмы, рассмотренные в параграфах 1-4, реализованы в рамках системы функционального программирования SFP ИСИ СО РАН. Сложность и эффективность алгоритмов для протягивания множественных значений и интервалов требуется проверить на реальных программах.

Задача построения среза является более общей по отношению к анализу значений в графе программных зависимостей программы, но такой вид анализа предоставляет результаты, которые сложно использовать в других алгоритмах оптимизации. Эффективность и сложность этого метода также требуется проверить на реальном классе задач.

СПИСОК ЛИТЕРАТУРЫ:

1. Касьянов В. Н., Бирюкова Ю. В., Евстигнеев В. А. Функциональный язык Sisal 3.0 // Поддержка супервычислений и интернет-ориентированные технологии. – Новосибирск: ИСИ СО РАН, 2001. — С. 54–67.
2. Евстигнеев В. А., Серебряков В. А. Методы межпроцедурного анализа (обзор) // Программирование. – 1992. – № 3. – С. 4–15.
3. Касьянов В. Н., Мирзуитова И. Л. SLICING: Срезы программ и их использование. – Новосибирск, 2002. – 116 с.
4. Keryell R. et al. PIPS – A Workbench for Interprocedural Program Analyses and Parallelization / R. Keryell, C. Ancourt, F. Coelho, B. Creusillet, F. Irigoin, P. Jouvelot – Paris, 1996. – 24 p. – (Tech. Rep. / Centre de Recherche en Informatique. Ecole Nationale Sup'erieure des Mines de Paris)

В. Н. Касьянов

ИНТЕГРИРОВАННАЯ ВИЗУАЛЬНАЯ СРЕДА ПОДДЕРЖКИ КОНСТРУИРОВАНИЯ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ

ВВЕДЕНИЕ

Параллельные вычисления являются одной из главных парадигм современного программирования и охватывают чрезвычайно широкий круг вопросов разработки программ. Ввиду значительно более сложной природы параллельных вычислений по сравнению с последовательными большое значение приобретают методы автоматизации разработки параллельного программного обеспечения, основанные на применении техники формальных моделей, спецификаций и преобразований параллельных программ [1, 3, 9, 13].

Фундаментальными проблемами организации параллельных вычислений являются проблема увеличения производительности и эффективности использования многопроцессорных и распределенных вычислительных систем и проблема повышения уровня интеллектуализации программирования параллельных систем. Они не являются независимыми, ибо организация высокопроизводительных вычислений в многопроцессорной системе современной архитектуры оказывается слишком сложной для попыток ее решения без средств интеллектуализации программирования в такой системе. Решение вопросов программирования параллельных систем трудно потому, что вопросы организации взаимодействий и синхронизации параллельных процессов существенно усложняют разработку параллельных алгоритмов и программ по сравнению с их традиционными (последовательными) вариантами. Имеющиеся средства автоматизации программирования многопроцессорных систем, в основном для языков семейства Фортран и С, не могут удовлетворить потребности все возрастающего количества пользователей, так как имеют ограниченные возможности как по предоставляемым средствам параллельного программирования, так и по кругу решаемых задач, характеризуемых в основном регулярными схемами вычислений.

Используя традиционные языки и методы, очень трудно разработать высококачественное, переносимое программное обеспечение для парал-

лельных компьютеров. В частности, параллельное программное обеспечение нельзя разрабатывать с малыми затратами на последовательных компьютерах и потом переносить на параллельные вычислительные системы без существенного переписывания и отладки. Поэтому высококачественное параллельное программное обеспечение может разрабатываться только небольшим кругом специалистов, имеющих прямой доступ к дорогостоящему оборудованию. Однако, используя языки программирования с неявным параллелизмом, такие как функциональный язык Sisal [18], можно преодолеть этот барьер и предоставить широкому кругу прикладных программистов, не имеющих достаточного доступа к параллельным вычислительным системам, но являющихся специалистами в своих прикладных областях, возможность быстрой разработки высококачественных переносимых параллельных алгоритмов на своем рабочем месте.

Функциональная семантика языков программирования с неявным параллелизмом гарантирует детерминированные результаты для параллельной и последовательной реализации – то, что невозможно гарантировать для традиционных языков, подобных языку Фортран. Более того, неявный параллелизм языка снимает необходимость переписывания исходного кода при переносе его с одного компьютера на другой. Гарантировано, что программа с неявным параллелизмом, правильно исполняющаяся па персональном компьютере, будет давать те же результаты при ее исполнении на высокоскоростном параллельном или распределенном вычислителе.

Статья посвящена системе функционального программирования SFP, создаваемой в Институте систем информатики СО РАН при финансовой поддержке РФФИ (грант № 07-07-12050) [16]. Система SFP предназначена для поддержки разработки высококачественного переносимого программного обеспечения для параллельных вычислителей на недорогих персональных компьютерах. Разработанный в качестве входного языка системы язык функционального программирования Sisal 3.2 обладает неявным параллелизмом, гарантирует детерминированные результаты и поддерживает аннотированное программирование. Система SFP использует графовые промежуточные представления функциональных программ и предоставляет средства для написания и отладки Sisal-программ независимо от целевой архитектуры, а также для трансляции Sisal-программ в оптимизированные императивные программы, подходящие для целевых платформ исполнения.

Статья структурирована следующим образом. В разд. 1 кратко описываются особенности языка Sisal 3.2. Общее описание системы SFP приводится в разд. 2. Разд. 3 посвящен рассмотрению базовой части системы. Метатранслятор и front-end транслятор системы представлены в разд. 4.

Разд. 5 и 6 содержат описание используемых системой промежуточных представлений программ.

1. ЯЗЫК SISAL 3.2

Название языка Sisal является аббревиатурой английского выражения «Streams and Iterations in a Single Assignment Language» (потоки и итерации в языке однократного присваивания) [18]. Создание языка – результат сотрудничества Ливерморской национальной лаборатории имени Лоренца, Университета штата Колорадо, Манчестерского университета и Digital Equipment Corporation (DEC). Язык ориентирован на поддержку научных вычислений и представляет собой дальнейшее развитие языка VAL.

По сравнению с императивными языками (подобными языку Фортран) функциональные языки, такие как Sisal, упрощают работу программисту. В функциональной программе программист должен только специфицировать результаты вычислений и может переложить большую часть работ по организации вычислений на компилятор, который отвечает за отображение алгоритма на определенную архитектуру вычислителя (включая планирование команд, передачу данных, синхронизацию вычислений, управление памятью и т.д.). По сравнению с другими функциональными языками Sisal поддерживает типы данных и операторы, присущие научным вычислениям, такие как циклы и массивы. Например, Sisal-программа умножения матриц может иметь следующий вид:

```
type OneDim = array[ double_real ];
type TwoDim = array[ OneDim ];
function Main(A,B: TwoDim; M,N,L: integer returns TwoDim)
   for I in 1, M cross J in 1, L
      S := for K in 1, N
                R := A[I,K] * B[K,J]
                returns value of sum R
           end for
       returns array of S
   end for
end function
```

Sisal разрабатывается как язык функционального программирования, специально ориентированный на параллельную обработку в научных вычислениях и на замену языка Фортран на суперкомпьютерах [14]. О реальном вытеснении говорить еще рано, но Sisal как язык параллельного про-

граммирования достаточно интересен сам по себе и уже нашел свое применение в десятках организаций разных стран мира. Существует несколько реализаций языка Sisal (версии 1.2) для суперЭВМ, в частности на Denelcor HEP, Vax 11-780, Cray-1, Cray-X/MP, создан прототип оптимизирующего компилятора с языка Sisal 1.2 в распределенные программы для вычислителей, аппаратно или программно поддерживающих многонитевые вычисления, таких как, например, TERA, *T, TAM и MIDC.

Ливерморская национальная лаборатория и Манчестерский университет разработали усовершенствованную версию языка Sisal 90 [15], которая пока еще нигде не была реализована. К наиболее значительным нововведениям языка можно отнести поддержку функций высших порядков и их полиморфизма, операций над массивами и потоками разных размерностей, потенциально недетерминированных пользовательских редукций, а также введение развитых расширений для организации циклических вычислений и вызовов функций, написанных на других языках программирования.

Последние годы в Институте систем информатики СО РАН ведутся исследования по разработке входного языка создаваемой системы параллельного программирования SFP на базе языка Sisal 90 [6]. Текущим результатом этих работ является язык Sisal 3.2 [8]. Язык Sisal 3.2 был получен путём развития языка Sisal 90 в сторону поддержки расширенных межмодульных взаимодействий, мультиязыкового и объектно-ориентированного программирования, а также возможностей предварительной обработки (preprocessing) и аннотированного программирования.

Для повышения уровня абстракции алгоритмов и возможности взаимодействия с другими языками программирования в язык Sisal 3.2 были введены новые концепции пользовательских типов с параметрами, обобщенных процедур и инородных типов. В языке Sisal 3.2 впервые было дано точное описание семантики аннотаций-утверждений ([17]) – прагм, позволяющих пользователю управлять оптимизирующими преобразованиями и настройкой транслируемой программы на целевой вычислитель. Язык Sisal 3.2 поддерживает также возможность (с синтаксисом, развивающим синтаксис языка Sisal 2.0 в этой области) использования функций уже готовых программ на других языках программирования, таких как C++, C и Фортран.

Пользовательские типы с параметрами позволяют задавать составные типы со специфическими операциями. Например, можно определить тип матрицы произвольного типа и операцию перемножения этих матриц (не поэлементного). Операция перемножения матриц задаётся с помощью обобщенной процедуры. Каждой обобщенной процедуре сопоставляется

ранее определённый контракт, в котором указано, какие операции должны поддерживать типы, задаваемые параметрами обобщенной процедуры. Например, для элементов типа матрицы это операции сложения и умножения. Далее приведён пример описания матрицы и операции их перемножения.

```
contract additive[T]
   operation + (T, T returns T)
   operation * (T, T returns T)
end contract
type matrix[T] = array [..,..] of T
operation * of additive[T] (matrix[T], matrix[T] returns matrix[T])
```

Инородные типы задаются некоторым строковым представлением на другом языке программирования. Значения инородных типов конструируются с помощью инородных операций и функций, написанных на другом языке программирования и расположенных в другом модуле со специальным интерфейсом на языке Sisal 3.2. С помощью инородных типов можно задавать архитектурно-зависимые типы и определять операции неявного преобразования между ними и «машинно-независимыми» типами языка Sisal. Например, можно определить инородные типы целых и вещественных чисел фиксированного размера и определить операции неявного преобразования между ними и типами «integer» и «real» языка Sisal. Также на инородных типах основывается возможность использования языком Sisal 3.2 функций уже написанных программ на других языках программирования.

2. СИСТЕМА SFP

Система SFP разрабатывается с целью предоставить прикладному программисту на его рабочем месте удобную среду для разработки функциональных программ на языке Sisal, предназначенных для последующего исполнения на параллельных супервычислителях, доступных через телекоммуникационные сети [6, 16].

В рамках этой среды программист должен иметь возможность, с одной стороны, создавать и отлаживать Sisal-программу без учета целевой параллельной архитектуры, а с другой – производить настройку отлаженной программы на ту или другую целевую параллельную архитектуру с целью достижения высокой эффективности исполнения разработанной программы на суперЭВМ.

Процедура настройки состоит в реализации оптимизирующей кросстрансляции разработанной функциональной программы в программу на

языке супервычислителя (например, на языке C или C#), в процессе которой программа подвергается необходимым оптимизирующим и реструктурирующим преобразованиям под управлением пользователя. При этом степень участия программиста может быть различной – от предоставления дополнительной информации до прямого управления производимыми преобразованиями. В частности, программист должен иметь возможность визуальной обработки создаваемой Sisal-программы в рамках ее внутреннего представления.

Такие возможности делают супервычислители, включенные в сеть, более доступными для использования широкому кругу прикладных программистов, а также позволяют упростить работу прикладным программистам и повысить эффективность использования ими супервычислителей за счет переноса работ по конструированию и отладке программ с дорогих супервычислителей на более дешевые и привычные персональные компьютеры, а также за счет снятия необходимости выполнять эти работы для одной и той же задачи каждый раз заново при переходе с одного супервычислителя на другой.

Система SFP включает следующие компоненты: ядро, визуальный каркас, отладчик, метатранслятор, транслятор, ретранслятор, блоки промежуточных представлений IR1, IR2 и IR3 [10, 12], блоки анализа, преобразования и визуализации IR1-, IR2- и IR3-программ, конверторы промежуточных представлений, генераторы выходного кода. IR1 и IR2 – это языки иерархических графов [5], представляющие функциональные программы в виде схем над распределенной и общей памятью [4, 7], а IR3 – язык для представления императивных программ.

Общая схема процесса кросс-компиляции в системе представлена на рис. 1. Исходная Sisal-программа («Source» на схеме) поступает на вход транслятора (front-end транслятора), который строит её первое внутреннее представление IR1. Далее граф IR1 подается на вход back-end части компилятора. Back-end часть включает следующие фазы, где фазы оптимизации являются необязательными: трансляция из IR1 в IR2 («IR2 Gen» на схеме); оптимизация IR2 («IR2 Opt» на схеме); трансляция из IR2 в IR3 с генерацией параллельного кода («IR3 Gen» на схеме); оптимизация IR3 («IR3 Opt» на схеме); понижение уровня IR3 (входит в блок «IR3 Opt»); трансляция IR3 в код целевой архитектуры («CodeGen» на схеме).

Целевой платформой для существующей реализации системы является платформа .NET. В ней в качестве кодогенератора используется транслятор внутреннего представления IR3 в программу на языке C#. Полученная программа транслируется в байт-код .NET компилятором языка C#. Разработа-

на библиотека, содержащая систему классов C#, обеспечивающих поддержку периода исполнения для Sisal программ. Кроме того, пользователь может использовать эти классы для обеспечения взаимодействия Sisal-программы и кода на языке C# (например, организовывать ввод-вывод).

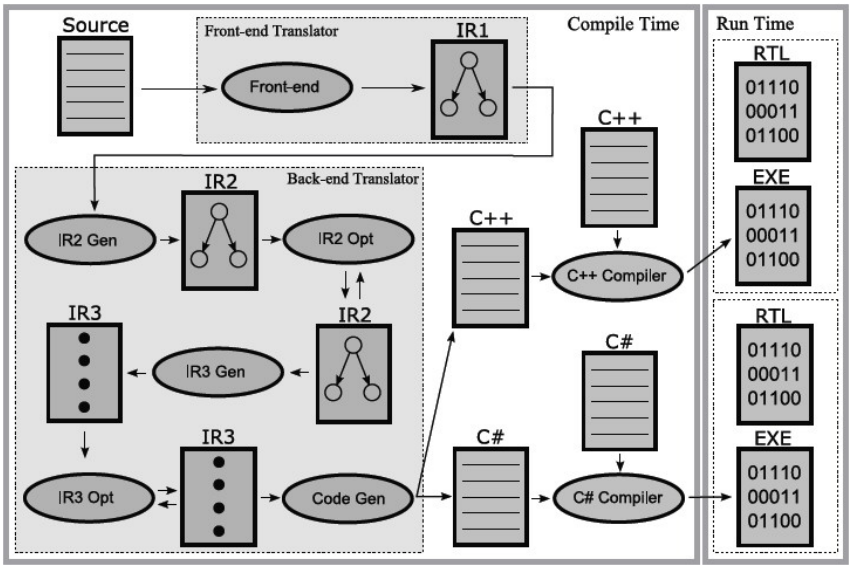


Рис. 1. Схема компиляции и исполнения Sisal программ

3. БАЗОВАЯ ЧАСТЬ СИСТЕМЫ SFP

На основе разработанного подхода к интеграции средств поддержки функционального программирования создана базовая часть системы, состоящая из загружаемого ядра и визуального каркаса (см. рис. 2) [2].

Каркас предоставляет базовый интерфейс пользователя: главное окно, меню, строку состояния, панели управления и инструментов, дочерние окна документов и другие стандартные элементы пользовательского интерфейса. Функциональность этого интерфейса пользователя зависит от загруженных компонентов и их состояния. Например, пункты меню могут меняться ди-

намически. При своей инициализации компоненты получают ссылку на интерфейс ядра, а у него можно запросить ссылку на интерфейс каркаса.

Каркас позволяет модулям создавать окна, меню и элементы управления. Кроме того, каркас может сам создавать элементы пользовательского интерфейса и предоставлять программный интерфейс модулям для работы с этим интерфейсом. В задачи каркаса входит также создание в отдельном окне нужного элемента управления. Элементами управления могут являться, например, визуализаторы.

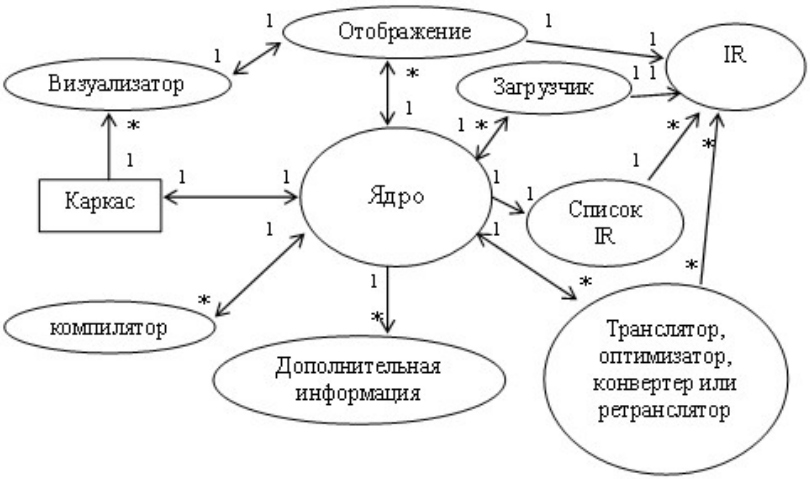


Рис. 2. Общая схема связей компонентов системы SFP

Визуализаторы позволяют отображать на экране информацию в виде графических изображений. Например, может быть визуализатор, рисующий граф представления программы. Способ задания графа визуализатору может быть любым и не обязан зависеть от представления программы, т.е. визуализатор может просто уметь рисовать графы некоторого типа. О том, как отобразить представление программы с помощью визуализатора, знает отображение. Отображения могут быть не отдельными компонентами, а лишь интерфейсами визуализаторов. Каждое отображение предназначено для связи внутреннего представления программы и визуализатора.

Компоненты представления и визуализации не обязаны знать о существовании друг друга, всё их взаимодействие осуществляется через компоненты отображений. Использование компонентов отображений позволяет увеличить повторное использование компонентов визуализации. Компоненты загрузки и сохранения позволяют хранить внутренние представления в файловом виде. Форматы хранения могут быть текстовыми или бинарными. Текстовые форматы могут быть полезны для ручного редактирования.

Компоненты взаимодействуют друг с другом и пользователем при помощи событий. Пользователь генерирует события при помощи вызова команд пользовательского интерфейса. Под событием понимается вызов одного компонента другим. События реализуются при помощи событий .NET. Вызывающий компонент имеет информацию только о сигнатуре вызова. Вызывающий компонент называется источником события, а вызываемый – подписчиком. Если компонент может обрабатывать события определённого типа, то он подписывается ко всем источникам данного события. О появлении нового источника оповещаются все компоненты, что позволяет организовать подписку. На одно событие могут подписаться сразу несколько компонентов (это не относится к пользовательским командам). Использование механизма событий позволяет упростить связи между модулями, а также интеграцию новых модулей, так как источники и подписчики могут работать независимо друг от друга.

4. МЕТАТРАНСЛЯТОР И FRONT-END ТРАНСЛЯТОР

В основу метатранслятора, предназначенного для автоматизации построения front-end трансляторов (трансляторов с входного языка системы в промежуточное представление IR1), положены предложенные понятия γ-автоматов и γ-схем, позволяющие наглядно описывать синтаксис языка в непосредственно исполняемом виде [11].

Модель γ-автомата основана на классической модели магазинного автомата и получается из неё за счет ограничений на вид функции перехода, позволяющих повысить наглядность графового представления автомата при сужении класса допустимых языков и последующих расширений модели дополнительными состояниями и возможностями их обработки, позволяющими расширить класс допустимых автоматом языков при сохранении наглядности графового представления модели. В γ-автомате используются переходы следующих трёх видов:

- переходы, не зависящие от содержимого магазина и не меняющие его содержимое;
- переходы, не зависящие от содержимого магазина, но добавляющие к нему состояния; и
- переходы в состояние, вытолкнутое из магазина.

Класс языков, распознаваемых детерминированными γ-автоматами, совпадает с классом LL_1-языков, но расширение автоматов контекстными состояниями позволяет задавать разбор более широких классов языков. Кроме того, модель γ-автомата расширяется средствами иерархической обработки неопределённостей, позволяющими осуществлять обнаружение и нейтрализацию ошибок трансляции без накладных расходов полного определения γ-автомата.

С помощью γ-схем можно реализовать весь транслятор целиком, объединяя описания синтаксиса и семантики языка программирования. Для этого к вершинам и дугам γ-схем добавляются в качестве пометок имена трансдукторов (транслирующих процедур). Реализация этих процедур в формализме γ-схем не уточняется и может иметь общий модифицируемый контекст исполнения, позволяющий задавать трансляцию произвольной сложности.

Метатранслятор предполагает разделение спецификации конкретного транслятора на три части: графическую, текстовую и интерпретирующую. Графическая часть описывает γ-схему, которая задает γ-автомат, распознающий синтаксис языка. Текстовая часть содержит описание контекстно-зависимых переходов («семантики отношений») и транслирующих процедур («операционной семантики»), использующихся в γ-автомате. Интерпретирующая часть транслятора исполняет его графическую и текстовую части. Общая схема конструируемого транслятора приведена на рис. 3.

Разделение входа метатранслятора на графическую и текстовую части позволяет сочетать сильные качества обеих форм представления. Графические спецификации больше подходят для проектирования и документирования из-за богатства способов представления, легче усваиваемых кратковременной памятью человека. Текстовые спецификации лучше подходят для реализации программы по причине сочетания строгости, гибкости, компактности записи и переносимости.

К другим преимуществам разделения входа метатранслятора на графическую и текстовую части относятся следующие свойства:

- простота модификаций входного языка путём наглядных изменений γ-схемы;
- высокая переносимость по причине интерпретируемости γ-автомата;

- упрощение реализации и тестирования процедур (проверки входных и выходных условий) благодаря их логической обособленности;
- автоматизация тестирования транслятора за счёт наличия исполняемого описания синтаксиса в виде γ-автомата;
- возможность использования динамических оптимизаций на уровне интерпретатора γ-автомата, настраивающих его на транслируемый язык или даже на конкретную программу языка в процессе её разбора;
- возможность легкой инструментации транслятора на уровне интерпретатора γ-автомата и сбора статистики программ транслируемого языка.

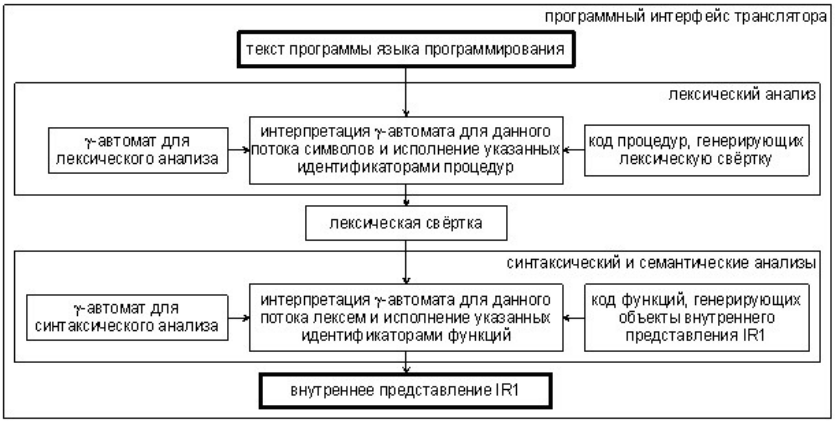


Рис. 3. Схема front-end транслятора,
реализованного с применением метатранслятора

Front-end транслятор языка Sisal 3.2 системы SFP спроектирован с использованием рассмотренного разделения на графическую, текстовую и интерпретирующую части. Front-end транслятор осуществляет лексический и синтаксический (совмещённый с семантическим) разбор текста программ языка Sisal 3.2, в процессе которого строится по входной программе ее IR1-представление. Разработанный транслятор обеспечивает простоту учёта модификаций языка, удовлетворительную скорость трансляции, качественные сообщения об ошибках и предупреждениях и развитые механизмы восстановления после ошибок разбора. Использование γ-схем позволило сократить объём текста front-end транслятора с языка Sisal 3.2 до десяти ты-

сяч строк по сравнению с тридцатью тысячами строк кода аналогичной части транслятора OSC 12.0 для более простой версии языка Sisal 1.2 [18].

5. ВНУТРЕННЕЕ ПРЕДСТАВЛЕНИЕ IR1

Первый промежуточный язык IR1 является языком помеченных иерархических графов [5, 7], построенных из (вычислительных) вершин, дуг, портов и типов (см. рис. 4). Вершины могут быть простыми или составными. Простые вершины являются вершинами основного графа и обозначают литералы или операции, такие как сложение или деление. Составные вершины являются фрагментами (или подграфами) основного графа и представляют составные конструкции, такие как структурные выражения и циклы. Порты – это другой тип вершин основного графа, которые используются для представления операндов вычислений. Они делятся на входные порты (входы) и выходные порты (выходы). Дуги, соединяют порты, и изображают передачу значений от одного операнда к другому. Они идут от выходов к входам вычислительных вершин, содержащихся в одном и том же фрагменте, либо соединяют выходы вершин или фрагментов с выходами фрагментов, непосредственно их содержащих, или входы фрагментов с входами фрагментов или вершин, в них непосредственно вложенных. Типы – это пометки дуг, которые представляют типы значений, передаваемых по этим дугам.

На рис. 4 изображено IR1-представление следующей функции, которая осуществляет быструю сортировку целочисленного массива:

```
function QuickSort(Data: array[integer] returns array[integer])
        if size(Data) <= 1 then Data else
                let Pivot := Data[1];
                        Low, Mid, High := for E in Data
                                returns array of E when E < Pivot;
                                        array of E when E = Pivot;
                                        array of E when E > Pivot
                        end for
                in QuickSort(Low) || Mid || QuickSort(High)
                end let
        end if
end function
```

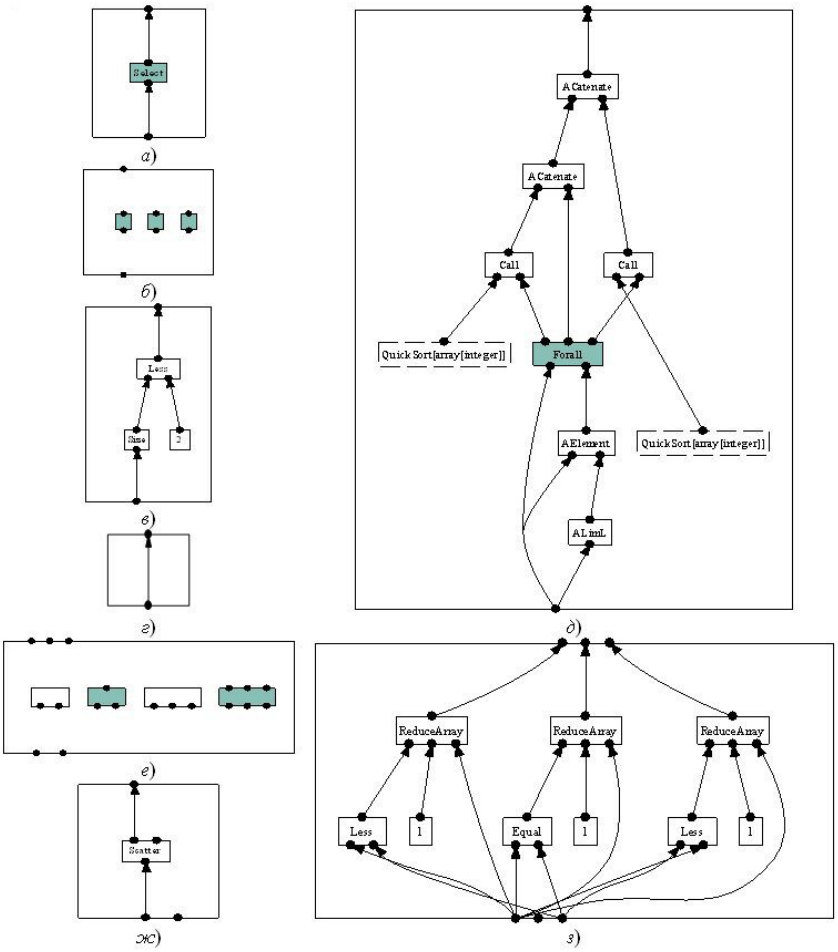


Рис. 4. Изображения графов, сгенерированные автоматически по IR1-представлению, построенному front-end транслятором для функции QuickSort:
а – граф функции QuickSort, *б* – граф составной вершины **Select**,
в – первый подфрагмент вершины **Select**, *г* – второй подфрагмент вершины **Select**,
д – третий подфрагмент вершины **Select**, *е* – граф составной вершины **Forall**,
ж – второй подфрагмент вершины **Forall**,
з – четвёртый подфрагмент вершины **Forall**

В языке IR1 альтернатива явным образом реализуется неявными зависимостями между элементами составной вершины **Select**. Первый её подфрагмент имеет один выход целого типа, значение которого определяет то, какой из других подфрагментов составной вершины **Select** будет использоваться для генерации её выходных значений. Литералы в графе IR1 задаются вершинами без входных портов и с одним выходным портом. Так как язык IR1 описывается иерархией ациклических графов, то вызовы функций задаются литералом функционального типа с именем функции, однозначно указывающим на её граф. В языке IR1 имеются также составные вершины **LoopA**, **LoopB** и **Forall**, которые представляют циклы с постусловием, предусловием и диапазоном соответственно. Данные составные вершины содержат фиксированное число элементов (подфрагментов), описывающих управление циклом, тело цикла и его предложение возврата.

6. ВНУТРЕННИЕ ПРЕДСТАВЛЕНИЯ IR2 И IR3

Внутреннее представление IR2 отличается от IR1 введением общей памяти в виде множества переменных VAR и отображения $\sigma_v : E \to VAR$, задающего привязку переменных к дугам E иерархического графа, а также явным заданием отношения частичного порядка $\leq_\beta$ на вычислительных вершинах иерархического графа, ограничивающего возможные последовательности их исполнения. Предполагается, что если $N_1 \leq_\beta N_2$, то вычисление, представленное вершиной N_1, должно обязательно происходить перед вычислением вершины N_2. Если же вершины N_1 и N_2 не связаны отношением $\leq_\beta$, то выполнять операции для этих вершин можно в любом порядке; кроме того, возможно их параллельное исполнение.

Для внутренних представлений IR2 (и IR3) определяются такие объекты, как переменная и тип. При этом тип в IR2 (и IR3) служит уже для представления типов языка Sisal на уровне back-end транслятора. Тип содержит низкоуровневую информацию об объекте, с которым ассоциирован этот тип (такую как машинное представление типа и класс памяти).

Переменная описывает объекты языка Sisal на уровне IR2 и IR3. В представлении IR2 переменные ассоциируются с дугами графа, а в IR3 они служат операндами операций IR3 и имеют следующие атрибуты: уникальный идентификатор, уникальное имя, тип и дополнительную булеву переменную, отвечающую за свойство «is error». Переменные делятся на скалярные, массивные и записи. Каждая из этих групп предполагает наличие у переменных некоторых дополнительных атрибутов. У скалярных перемен-

ных это размер в байтах. Переменные-массивы дополнительно содержат три вспомогательные переменные: переменную, описывающую элемент массива, нижнюю границу массива и его размер. Переменные-записи содержат список переменных, описывающих поля данной записи.

IR2-представление строится из IR1-графа в два этапа. Первым этапом является построение отображения σ_v, т.е. аннотирование дуг иерархического графа переменными. Сначала оно строится так, чтобы дугам, выходящим из одного порта, приписывалась одна и та же переменная, а дугам, выходящим из разных портов, – разные переменные. Этим достигается выполнение требования единственности определения каждой переменной. В дальнейшем, при оптимизации IR2, распределение переменных по дугам графа может меняться. Заключительным этапом построения IR2 является упорядочивание вершин отношением приоритета исполнения $\leq_\beta$, которое строится исходя из ограничений, накладываемых дугами, связывающими операнды операторов, и правилами вычисления элементов (вершин) составных вершин, описывающих условные выражения и циклы.

Фаза оптимизации IR2 выполняет оптимизацию распределения переменных агрегатных типов для дуг графа и вынос инвариантных вычислений из циклов [4]. Оптимизация переменных агрегатных типов заключается в таком перераспределении переменных по дугам графа, которое позволяет избежать избыточного копирования объектов. Это преобразование выполняется в два прохода: первый проход – поиск вершин, которые являются единственными использованиями агрегатных типов; второй – привязывание входу и выходу каждой из таких вершин одной и той же переменной.

После оптимизации IR2 выполняется распараллеливание. На этом этапе производится раскраска внутреннего представления с целью выявления параллельной структуры алгоритма и нахождения фрагментов кода, для которых распараллеливание будет малоэффективным. После этого производится пометка циклов, итерации которых могут быть выполнены независимо (ParDo). В зависимости от определенных условий некоторые из этих циклов назначаются для параллельного исполнения.

Представление IR3 является среднеуровневым императивным представлением программы, состоящим из операторов и выражений. Операторы IR3 соединены между собой лексически (граф потока управления не строится).

В процессе трансляции из IR2 в IR3 для графа потока данных вычислений IR2 фактически строится императивная программа (последовательность операторов), выполняющая вычисления, заданные этим графом. Построение IR3 из графа IR2 включает генерацию последовательности опера-

торов для каждой вершины и размещение полученного фрагмента в общей последовательности операторов IR3 .

Блок оптимизации внутреннего представления IR3 выполняет втягивание переменных, удаление мёртвых присваиваний и удаление бесполезных присваиваний [4]. Эффективность оптимизирующих преобразований на примере задачи умножения двух матриц показана на рис. 5.

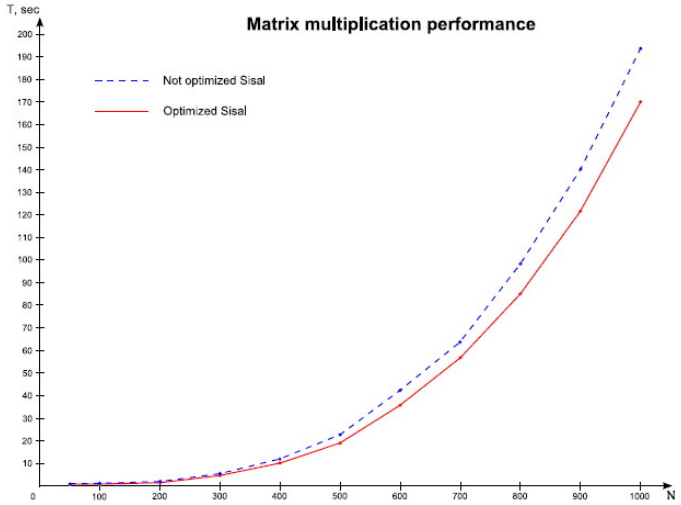


Рис. 5. Эффективность оптимизации

После выполнения оптимизирующих преобразований выполняется «понижение уровня» IR3, которое заключается в замене «функций-интринсиков» (intrinsic functions) на последовательность простых операторов или «функций-интринсиков» более низкого уровня, которые заменяются уже на фазе кодогенерации.

Распараллеливание на уровне IR3 заключается в создании вспомогательных структур, относящихся непосредственно к целевому языку трансляции. На этом шаге создаются дополнительные переменные и классы, которые требуются для описания параллельной работы программы. Эффективность распараллеливания на примере задачи умножения двух матриц показана на рис. 6. Тесты были проведены на 4-х процессорной ЭВМ с SMP-архитектурой.

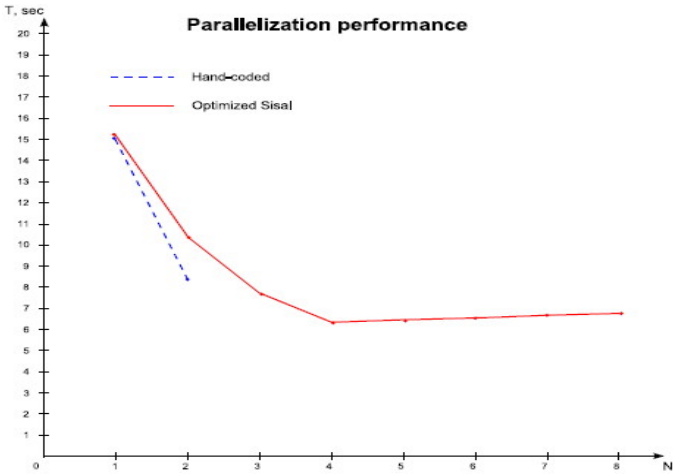


Рис. 6. Эффективность распараллеливания

ЗАКЛЮЧЕНИЕ

В статье описана система SFP, предназначенная для поддержки разработки высококачественного переносимого программного обеспечения для параллельных вычислителей на недорогих персональных компьютерах, и ее основные компоненты в существующем виде. Вкратце описаны основные черты языка Sisal 3.2. Описаны внутренние представления IR1, IR2 и IR3. Рассмотрена схема front-end трансляции из языка Sisal 3.2 в первое внутренне представление IR1. Представлены результаты исследования эффективности оптимизирующих преобразований и результаты исполнения программ на SMP архитектуре.

Благодарности. Автор благодарен всем участникам проекта SFP, работа над которым выполняется при поддержке РФФИ (грант № 07-07-12050).

СПИСОК ЛИТЕРАТУРЫ

1. Воеводин В. В., Воеводин Вл. В., Параллельные вычисления. – СПб., БХВ-Петербург, 2002. – 609 с.

2. Глуханков М. П. Интегрированная среда визуального функционального программирования SFP // Молодая информатика. – Новосибирск, 2005. – С. 21–30.
3. Евстигнеев В. А., Касьянов В. Н. Оптимизирующие преобразования в распараллеливающих компиляторах // Программирование. – 1996. – № 6. – С. 12 –26.
4. Касьянов В. Н. Оптимизирующие преобразования программ. – М.: Наука, 1988. – 336 С.
5. Касьянов В. Н. Иерархические графы и графовые модели: вопросы визуальной обработки // Проблемы систем информатики и программирования. – Новосибирск, 1999. – С. 7–32.
6. Касьянов В. Н., Бирюкова Ю. В., Евстигнеев В. А. Функциональный язык Sisal 3.0 // Поддержка супервычислений и интернет-ориентированные технологии. – Новосибирск, 2001. – С.54–67.
7. Касьянов В. Н., Евстигнеев В.А. Графы в программировании: обработка, визуализация и применение. – СПб.: БХВ-Петербург, 2003. – 1104 с.
8. Касьянов В. Н., Стасенко А.П. Язык программирования Sisal 3.2 // Методы и инструменты конструирования программ. – Новосибирск, 2007. – С. 56–134.
9. Малышкин В.Э., Корнеев В.Д. Параллельное программирование мультикомпьютеров. – Новосибирск, Изд-во НГТУ,2006. – 296 с.
10. Пыжов К.А. Внутренние представления среднего уровня для компиляторов языка Sisal // Методы и инструменты конструирования программ. – Новосибирск, 2007. – С. 174–185.
11. Стасенко А.П. Автоматная модель визуального описания синтаксического разбора // Методы и инструменты конструирования программ. – Новосибирск, 2007. – С. 186–209.
12. Стасенко А.П. Система интерфейсов транслятора во внутреннее представление IR1 // Методы и инструменты конструирования и оптимизации программ. – Новосибирск, 2005. – С. 229–238.
13. Allen R., Kennedy K. Optimizing compilers for modern architectures. A dependence-based approach – Morgan Kaufmann Publishers, 2002. – 790 p.
14. Cann D.C. Retire Fortran? A debate rekindled // Communs. ACM. – 1992. – Vol. 35, N 8. – P. 81–89.
15. Feo J.T., Miller P.J., Skedzielewski S.K. and Denton S.M. Sisal 90 user's guide. – Livermore, CA: Lawrence Livermore National Laboratory, Draft 0.96, 1995. – 80 p.
16. Kasyanov V. N., Stasenko A. P., Gluhankov M. P., Dortman P. A., Pyjov K. A., Sinyakov A. I. SFP – an interactive visual environment for supporting of functional programming and supercomputing // WSEAS Transactions on Computers. – 2006. – Vol. 5, Iss. 9. – P. 2063– 2069.
17. Kasyanov V. N., Transformational approach to program concretization // Theor.l Comput. Sci. – 1991. – Vol. 90, N 1. – P. 37–46.
18. McGraw J. R., Skedzielewski S. K., Allan S. J., Oldehoeft R. R., Glauert J., Kirkham C., Noyce B., Thomas R. Sisal: Streams and iterations in a single assignment language, Language Reference Manual, Version 1.2. – Livermore, CA, 1985. – (Tech. Rep. / Lawrence Livermore National Laboratory; M-146, Rev. 1).

В. Н. Касьянов

ВСЕМИРНЫЕ КОМПЬЮТЕРНЫЕ КОНГРЕССЫ ИФИП[1]

ВВЕДЕНИЕ

1 ноября 2008 г. исполняется 50 лет со дня образования Отдела программирования (ОП) Института математики СО АН СССР, фактически положившего начало формирования нашего коллектива, известного сегодня под названием «Сибирская школа информатики и программирования академика Андрея Петровича Ершова».

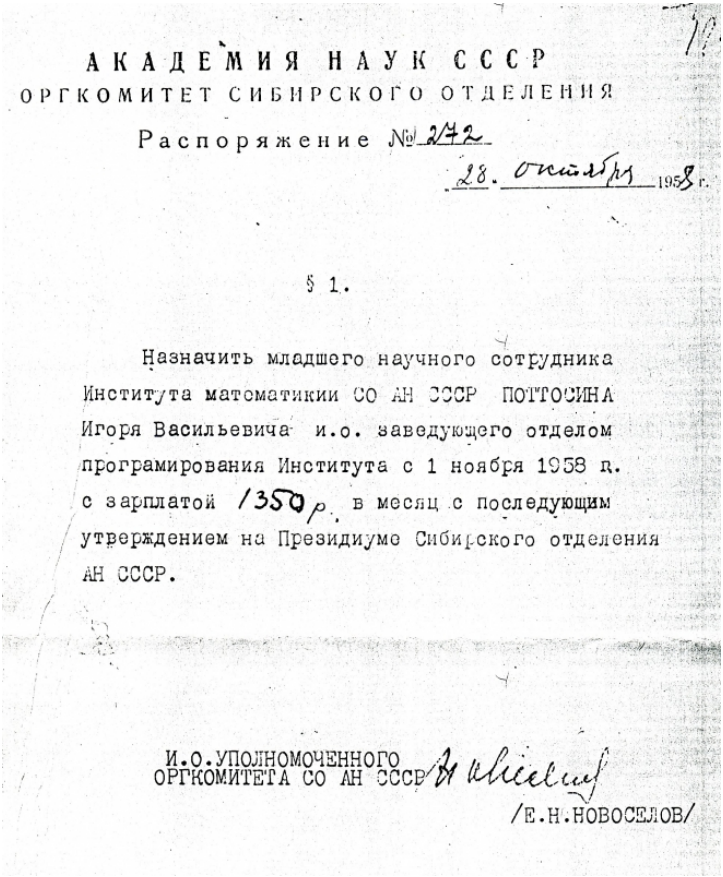
История возникновения нашего коллектива начинается в 1957 году, когда один из основателей Сибирского отделения Академии наук СССР и первый директор Института математики СО АН Сергей Львович Соболев, столетие со дня рождения которого мы также отмечаем в этом году, предложил Андрею Петровичу Ершову, ученику Алексея Андреевича Ляпунова, тогда сотруднику Вычислительного центра АН СССР (г. Москва), организовать и возглавить Отдел программирования в своем Институте.

Поскольку Ершов сразу переехать в Академгородок не смог, формально первым заведующим ОП стал Игорь Васильевич Поттосин. Приказ о его назначении был подписан 1 ноября 1958 г., и с этого дня ведет свой отсчет история Отдела программирования (Рис. 1). Однако фактическим руководителем Отдела с первых дней был А. П. Ершов. Он активно участвовал в формировании штата ОП и определял основные направления его работы, а в начале 1961 г. переехал в Академгородок и уже и формально возглавил Отдел программирования.

В 1964 г. был образован Вычислительный центр СО АН (ВЦ) под руководством Гурия Ивановича Марчука, и Отдел вошел в его состав. С течением времени расширялся круг задач, стоящих перед коллективом, менялась его организационная структура, появлялись новые направления исследований и новые структурные образования, в том числе новые отделы. Благодаря усилиям А. П. Ершова и его учеников Академгородок стал одним из

[1] Расширенный текст доклада автора на заседании N 696 от 28 октября 2008 г. Объединенного семинара ИСИ СО РАН и НГУ «Конструирование и оптимизация программ», посвященном 50-летию со дня образования Отдела программирования Института математики СО АН СССР.

главных центров развития информатики и программирования в СССР, активно взаимодействующим с Международной федерацией по обработке информации (ИФИП) [1] и другими зарубежными организациями, местом паломничества многих советских и зарубежных системных программистов. Уже в 70-е годы прошлого столетия не только результаты, полученные в Отделе программирования, но и библиотека Ершова и даже кофе-клуб Отдела программирования приобретают всемирную известность среди специалистов, занимающихся компьютерными науками.



АКАДЕМИЯ НАУК СССР
ОРГКОМИТЕТ СИБИРСКОГО ОТДЕЛЕНИЯ

Распоряжение №272

28. октября 1958 г.

§ 1.

Назначить младшего научного сотрудника Института математикии СО АН СССР ПОТТОСИНА Игоря Васильевича и.о. заведующего отделом програмирования Института с 1 ноября 1958 г. с зарплатой 1350 р. в месяц с последующим утверждением на Президиуме Сибирского отделения АН СССР.

И.О.УПОЛНОМОЧЕННОГО
ОРГКОМИТЕТА СО АН СССР
/Е.Н.НОВОСЕЛОВ/

Рис. 1

Наш семинар «Конструирование и оптимизация программ», правда, под несколько другим названием, был организован в 1979 г. как первый тематический семинар, объединяющий исследования по теоретическому и системному программированию. К этому времени в отделе существовало два семинара: «Системное программирование», руководители – А. П. Ершов и И. В. Поттосин, и «Теоретическое программирование», руководители – В. Е. Котов и В. А. Непомнящий. Надо сказать, что наш семинар возник почти одновременно с образованием в Отделе программирования ВЦ структурной группы «Теории и методов трансляции», руководитель – В. Н. Касьянов, которая в дальнейшем была преобразована в лабораторию Конструирования и оптимизации программ Института систем информатики СО РАН.

Институт систем информатики СО РАН, образованный в 1990 г. на базе нескольких отделов ВЦ, выросших из ОП, по праву считается наследником и продолжателем лучших традиций Отдела программирования. Поэтому неслучайно ему было присвоено в 1995 г. имя основателя Отдела, академика А. П. Ершова.

Андрей Петрович Ершов – один из тех ученых, которые росли вместе с Сибирским отделением АН СССР, чья деятельность создавала авторитет и научную известность работам этого отделения. При этом в отличие от стандартной ситуации для советского времени, когда высокая позиция советского ученого внутри страны давала ему право на представительство страны в международных организациях, рост известности и авторитета А. П. Ершова на международном уровне не только происходил одновременно с ростом его влияния внутри страны, но часто даже опережал его. В связи с этим можно вспомнить наш разговор с Игорем Васильевичем Поттосиным об этом времени, когда происходило становление СО РАН. В нем он рассказал об обычной для того времени ситуации, когда он, тогда совсем молодой кандидат наук из новосибирского Академгородка, на равных конкурировал с известным академиком из Москвы за место в делегации на конференцию от Академии наук. Этот разговор состоялся в 80-е годы прошлого столетия в Москве в Управлении внешних связей АН СССР (УВС) – организации, через которую в то время оформлялись все служебные выезды советских ученых заграницу. В тот раз мы с И. В. Поттосиным были приглашены выступить с заказными докладами на международной конференции и должны были за счет принимающей стороны выехать в Болгарию, где она проходила, в составе научной делегации от Академии наук. Однако по приезду в Москву, и придя в УВС, где нас ожидали уже готовые билеты и служебные паспорта с визами, мы вынуждены были изменить наши планы. Консультант, курирующий наш выезд, сообщил нам, что неожиданно

произошло уменьшение квоты Сибирского отделения в делегации на одно место (т.е. с двух мест до одного) и что нам предстоит решить, кто из нас двоих дальше не едет и возвращается в Новосибирск. Потом выяснилось, что это уменьшение квоты было вызвано тем, что один московский академик, входивший в состав делегации, в последний момент решил взять в поездку свою жену.

В своём докладе, посвященном 50-ю образования ОП, я остановлюсь на Всемирных компьютерных конгрессах ИФИП, являющихся одним из главных международных мероприятий в области информатики и программирования. Андрей Петрович не только выступал на шести конгрессах ИФИП (он участвовал во всех конгрессах, которые состоялись в период с 1965 г. по 1983 г.), но и тратил много сил на их организацию. Помимо этого, А. П. Ершов активно участвовал в деятельности Рабочей группы 2.1 по Алголу Технического комитета 2 ИФИП по программированию (ТК 2) и в работе самого ТК 2, а также в организации рабочих конференций, проводимых под эгидой ТК 2 ИФИП. В 1980 г. за плодотворную деятельность в ИФИП по организации конгрессов А. П. Ершов был награжден «Серебряным сердечником» (Silver Core) – одним из высших знаков отличия для членов ИФИП [2].

Доклад опирается на мои личные воспоминания по участию за четверть века в четырех конгрессах ИФИП и использует некоторые материалы из электронного архива Ершова [3].

Вначале я кратко остановлюсь на целях и структуре ИФИП (разд. 1). Затем опишу Рабочую конференцию ИФИП по машинно-ориентированным языкам высокого уровня (разработка качественного программного обеспечения), которая прошла в Новосибирске в 1977 г. под руководством А. П. Ершова (разд. 2). Далее я рассмотрю те Всемирные компьютерные конгрессы ИФИП, в организации которых принимал участие А. П. Ершов и на которых он выступал (разд. 3). Более подробно я расскажу о 9-м Конгрессе ИФИП-83 и о судьбе моего доклада на нем (разд. 4); ИФИП-83 стал последним в серии тех конгрессов, на которые выезжал А. П. Ершов, и первым, в котором попытался участвовать я. Затем я поделюсь воспоминаниями о тех трех конгрессах, на которых я выступал с докладами (разд. 5, 6 и 7).

1. МЕЖДУНАРОДНАЯ ФЕДЕРАЦИЯ ПО ОБРАБОТКЕ ИНФОРМАЦИИ

Международная федерация по обработке информации (IFIP или ИФИП) [1] – неправительственная некоммерческая рамочная организация национальных обществ, работающих в области информационной обработки, соз-

дана в 1960 г. под эгидой ЮНЕСКО как результат первого всемирного компьютерного конгресса, который состоялся в Париже в 1959 г.

Создание ИФИП отвечало насущным проблемам времени. В 60-е гг. прошлого столетия в мире начался существенный рост компьютерной индустрии, и стала быстро расширяться сфера применения ее продуктов. Таким образом, с началом работы ИФИП информационные технологии все в большей степени становятся эффективным инструментом, влияющим на жизнь людей, причем в разных направлениях: в науке и инженерии, в коммерции и индустрии, в образовании и управлении, а также в сферах досуга.

Основными целями ИФИП являются

- способствование международной кооперации,
- стимулирование исследований и разработок,
- поддержка образования,
- распространение информации.

Своей миссией ИФИП считает право быть лидирующей истинно международной неполитической организацией, которая поощряет и поддерживает разработку, распространение и применение информационных технологий на пользу всему человечеству.

Членами ИФИП являются более 60 общественных организаций и академий наук, представляющих страны различных регионов мира, в том числе Россию, из которых 45 являются полными членами, 4 – членами-корреспондентами, 1 – ассоциативным членом и 11 – объединенными членами.

Среди индивидуальных членов ИФИП 18.1% специалистов из индустрии, 75% – из университетов, 3.8% занимаются управлением. Женщины в ИФИП составляют 12.4%, а молодежь (до 40 лет) – 19.4%.

ИФИП поддерживает дружественные связи со многими неправительственными организациями, первой из которых является ЮНЕСКО, и тесно взаимодействует с такими международными федерациями, как IFAC, IMACS, IFORS и IMEKO.

Главным событием ИФИП является Всемирный компьютерный конгресс, который в настоящее время проводится раз в два года. Помимо конгресса, ИФИП поддерживает главные международные конференции по информационным технологиям, которые организуются техническими комитетами ИФИП и их рабочими группами. В настоящее время ИФИП включает 85 рабочих групп и состоит из 13 технических комитетов:

- основания информатики;
- программное обеспечение: теория и практика;

- образование;
- применения компьютерных технологий;
- коммуникационные системы;
- системы моделирования и оптимизации;
- информационные системы;
- отношение между компьютерами и обществом;
- технология компьютерных систем;
- секретность и защита систем информационной обработки;
- искусственный интеллект;
- человеко-машинное взаимодействие;
- досуговые вычисления.

2. РАБОЧАЯ КОНФЕРЕНЦИЯ ИФИП В НОВОСИБИРСКЕ

С 24 по 27 мая 1977 г. в Новосибирске состоялась Рабочая конференция ИФИП по машинно-ориентированным языкам высокого уровня (разработка качественного программного обеспечения). На конференции присутствовало 57 ученых из 16 стран. Это была первая столь представительная международная конференция по программированию, состоявшаяся в СССР.

Решение о проведении конференции было принято Генеральной ассамблеей ИФИП и подтверждено планом Государственного комитета Совета Министров СССР по науке и технике. Научная подготовка конференции проводилась Техническим комитетом ИФИП № 2, представителем в котором от СССР был А. П. Ершов. Делегаты Рабочих конференций ИФИП приглашаются международным комитетом по списку, утвержденному техническим комитетом. Усилиями А. П. Ершова было обеспечено представительной советской делегации, которая насчитывала 17 человек из 7 городов.

В извещении конференции ее тематика характеризовалась следующим образом. «Разнообразные критерии, входящие в определение качества программного продукта, включают

– правильность (соответствие спецификации),

– безотказность (при любых условиях применения),

– эффективность (соразмерную со сложностью задачи),

– ясность (как программы, так и её функционирования).

Давно признано, что обеспечение надлежащего баланса таких (необязательно совместимых) целей в первую очередь зависит от того, как программа конструируется. Успех зависит от методов, использованных при

разработке программы, и от инструментария, использованного в этом процессе (причем язык программирования есть лишь наиболее обычная часть этого инструментария).»

Все представленные на конференции доклады были разделены на три группы:

– методы проектирования качественного софтвера и критерии качества,

– инструментальная база,

– анализ конкретных систем и специальных вопросов.

Это разделение было во многом условным, поскольку большинство докладов затрагивало целый комплекс вопросов. Например, почти все работы по разработке инструментальной базы содержали изложение определенной методики проектирования.

В докладе В. Н. Касьянова и И. В. Поттосина «Применение методов оптимизации к проверке правильности программ» [4, 5] рассматривалось как алгоритмы потокового анализа и ряда оптимизирующих преобразований практически без изменения могут быть применены для повышения надежности за счет обнаружения в тексте программы довольно широкого класса неправдоподобностей – определенных свойств, присущих неправильным программам.

Понятие правдоподобности, рассмотренное в докладе, связано с содержательным пониманием программы как статического изображения некоторых реализаций алгоритма без учета того, какую задачу решает этот алгоритм, т.е. без знания спецификации программы, необходимой для ее верификации. Содержательно, правдоподобность некоторой программы означает, что все изображаемые ею исполнения осмысленны и согласуются с программным текстом. Более точно, правдоподобная программа в точности решает некоторую задачу, т.е. не содержит действий, избыточных по отношению к своей задаче. В правдоподобной программе все средства, которые выбраны для выражения алгоритма решения задачи, используются естественным образом, а результаты правдоподобной программы не зависят от того, как в ней будут исполнены семантически неопределенные действия.

Таким образом, понятие неправдоподобности не совпадает с понятием правильности, хотя и содержит в себе ряд свойств понятия правильности. Строго говоря, программа может быть правдоподобной и неправильной, а также неправдоподобной и правильной. Вместе с тем, последнее возможно, когда автор сознательно хочет построить неправдоподобную правильную программу, а это всегда труднее и менее выгодно, чем построить правдоподобную правильную программу. Второй реальной ситуацией, при которой мы сталкиваемся с правильной и неправдоподобной программой, является

попытка оценить на правдоподобность программы после и без учёта конкретизации алгоритма.

Ясно, что статическое доказательство правдоподобности системных программ могло бы повысить надежность математического обеспечения, и можно бы было ставить вопрос о доказательстве правдоподобности программ аналогично доказательству ее правильности. Однако, с содержательной точки зрения, полная правдоподобность, которую мы могли бы пытаться проверить, связана с исследованием всех допустимых путей в программе, проблема выделения которых алгоритмически неразрешима.

Поэтому на практике мы вынуждены рассматривать некоторое покрывающее множество в качестве множества всех возможных исполнений программы и не можем оценить близость рассматриваемого множества к точному множеству допустимых путей. За счет этого могут возникнуть некоторые неправдоподобные исполнения, которых нет среди реальных исполнений, а с другой стороны, потеряться неправдоподобности, связанные с представлением программой множества своих исполнений.

Таким образом, мы не можем доказывать полную правдоподобность программ, и не можем доверять доказательству правдоподобности программ относительно расширенного множества ее исполнений. Поэтому проверка неправдоподобности программ вместо доказательства ее правдоподобности – единственный путь сделать понятие правдоподобности достаточно конструктивным, чтобы его использовать на практике.

Следует заметить, что оптимизация программ имеет дело именно с таким пониманием программы, которое описано выше. Естественно поэтому пытаться использовать технику и методы оптимизации программ для автоматизации выявления неправдоподобности программы. Подобная автоматическая система может служить пользователю источником подсказок автору относительно того, что в программе существуют некоторые несообразности, которые могут быть следствием ошибок.

Авторы рассмотрели список основных неправдоподобностей (тех, которые наиболее часто можно ожидать в реальных неправильных программах) и показали, как указанные свойства могут быть обнаружены комбинацией обычных оптимизационных техник.

В этот список авторы включили

- незаданность переменных, когда при некотором исполнении происходит обращение за значением переменной ранее, чем эта переменная получает какое-либо значение;

- возможность бесконечного исполнения, которое возникает из-за бесконечной рекурсии, бесконечного ожидания или бесконечного цикла;
- существование неиспользуемых объектов, т.е. различных явно описанных, но не используемых в программе объектов (типов, переменных, процедур и т.д.), а также операторов, не выполнимых ни при одном исполнении программы;
- существование излишних вычислений, например, операторов, не используемых для получения результатов программы;
- несоответствие используемых конструкций и изображаемых ими действий, когда используется конструкция, которая по своему характеру гораздо универсальнее или сложнее, чем те действия, которые ею реализуются, например, когда в программе есть цикл, тело которого исполняется только раз, метка, на которую нет перехода, массив, который не используется целиком и у которого все индексные выражения по одному из измерений являются константами, или указатель, принимающий единственное значение;
- наличие побочного эффекта в совместных исполнениях;
- наличие семантически запрещенных или неопределенных конструкций.

Таким образом, содержательно неправдоподобность программы складывается из возможности неправдоподобных исполнений программ и из неправдоподобного представления программой своих исполнений. Программа становится абсолютно неправдоподобной, или, что то же самое, ошибочной, если она не содержит правдоподобных исполнений.

Авторы видят недостатки предложенного подхода в неполной достоверности обнаруживаемых ошибок, но отмечают, что этот подход позволяет обнаружить содержательные ошибки и ошибки, носящие динамический характер, статически и без содержательного знания задачи.

3. ВСЕМИРНЫЕ КОМПЬЮТЕРНЫЕ КОНГРЕССЫ ИФИП

Хотя ИФИП ведет достаточно широкий фронт работ в области информатики, в том числе ежегодно участвует в проведении многочисленных конференций и совещаний, главным событием ИФИП является Всемирный компьютерный конгресс, который первоначально проходил раз в четыре года, а в настоящее время проводится раз в два года.

Сама Международная федерация по обработке информации (ИФИП) была создана в 1960 г. под эгидой ЮНЕСКО как результат первого Всемирного компьютерного конгресса, который состоялся в 1959 г. в Париже. Традиционно в конгрессах принимает участие широкий круг ведущих специалистов различных регионов мира, и конгресс является главным мировым научным форумом в области информатики, на котором рассматриваются основные проблемы и наиболее важные новые результаты основных направлений современной информатики.

За долгое время проведение конгрессов сформировалась традиция проводить конгресс в виде ряда конференций, посвященных наиболее актуальным направлениям развития информатики и формируемых соответствующими техническими комитетами ИФИП. Программа каждой из конференций формируется своим международным программным комитетом, составленным из ведущих мировых специалистов в соответствующих областях. Каждый доклад, включенный в программу любой из конференций, проходит жесткий отбор и рассматривается не менее чем тремя рецензентами. Поэтому уровень докладов на конгрессах ИФИП, как правило, очень высок, а любое выступление всегда весьма почетно.

Конференции, составляющие конгресс, всегда работают параллельно и на одной территории. Это позволяет организаторам конгресса проводить различные общие мероприятия конгресса, а также дает возможность каждому приехавшему на конгресс участвовать в любом интересном ему заседании любой конференции и встретиться с любым его участником.

Как представитель СССР в Техническом комитете 2 ИФИП по программированию, А. П. Ершов принимал активное участие в организации всех тех конференций конгресса, которые формировались этим комитетом. Так он стал одним из активных организаторов Третьего конгресса ИФИП (ИФИП-65), который состоялся с 24 по 29 мая 1965 г. в США (г. Нью-Йорк), и выступил на нем с двумя докладами: «АЛЬФА– система автоматизации программирования высокого качества» [6] и «Система программирования, основанная на взаимодействии человека и машины» (совместно с Г. И. Марчуком) [7].

Система АЛЬФА [8], представленная в докладе Ершовым, была первой в мировой практике оптимизирующей системой программирования для языков, более сложных, чем Фортран. Существовавший в то же время английский проект (Хоукинс и Хакстейбл) для Алгола 60, аналогичный проекту АЛЬФА по функциональным возможностям, так и не был доведен до конца. Это важно отметить потому, что сама возможность существования трансляторов для языков, более сложных, чем Фортран, с приемлемой эф-

фективностью объектных программ в то время многими оспаривалась. Система АЛЬФА стала конструктивным доказательством такой возможности, и это существенно, ибо снимало преграды на пути создания новых, семантически более богатых языков.

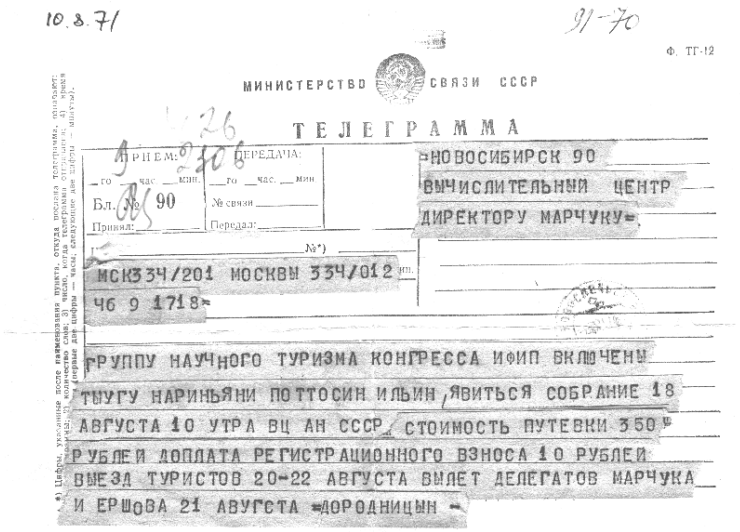
Работы по системе АЛЬФА внесли крупный вклад в методологию оптимизирующей трансляции. Была предложена и реализована многопросмотровая схема трансляции, ориентированная на оптимизацию, впервые в практику оптимизации программ были введены оптимизирующие преобразования на уровне промежуточного представления транслируемой программы, были выделены и построены промежуточные представления программы, ориентированные на алгоритмы оптимизации.

А. П. Ершов был вице-председателем программного комитета Четвертого конгресса ИФИП (ИФИП-68), который проходил со 2 по 7 августа 1968 г. в Шотландии (г. Эдинбург), и участвовал в панельной дискуссии «Разделение времени: потребность в переориентации», состоявшейся в рамках программы конгресса [9].

Осмысливая опыт проекта АИСТ и других подобных работ у нас в стране и за рубежом, в своем докладе на дискуссии Ершов выдвинул несколько насущных тезисов:

- о специализации процессов в многопроцессорном комплексе для разделения времени;
- об универсальных и специализированных системах разделения времени и областях их применимости;
- о различии требований профессионалов и непрофессионалов в программировании к общению с системами разделения времени;
- о том аспекте систем, который впоследствии будет назван дружественностью к пользователю.

На 5-м Конгрессе ИФИП (ИФИП-71), который состоялся с 23 по 28 августа 1971 г. в Югославии (г. Любляна), А. П. Ершов был членом технического комитета «Программирование и структуры систем» и выступил с известным обзорным докладом «Теория схем программ» [10]. Это был первый конгресс ИФИП, который проходил в стране «социалистического лагеря», и поэтому отличался сравнительно большой делегацией ученых из СССР. Так, только из ВЦ в конгрессе участвовало 6 ученых: два (Г. И. Марчук и А. П. Ершов) в качестве делегатов, и 4 (Э. Х. Тыугу, А. С. Нариньяни, И. В. Поттосин, В. П. Ильин) в качестве туристов (рис. 2).



10.8.71

91-70

Ф. ТГ-12

МИНИСТЕРСТВО СВЯЗИ СССР

ТЕЛЕГРАММА

ПРИЕМ: ПЕРЕДАЧА:

Бл. № 90

Принял: Передал:

№ связи

№*)

МСК334/201 МОСКВЫ 334/012 ин.

ЧБ 9 1718=

=НОВОСИБИРСК 90

ВЫЧИСЛИТЕЛЬНЫЙ ЦЕНТР

ДИРЕКТОРУ МАРЧУКУ=

ГРУППУ НАУЧНОГО ТУРИЗМА КОНГРЕССА ИФИП ВКЛЮЧЕНУ

ТНУГУ НАРИНЬЯНИ ПОТТОСИН ИЛЬИН ЯВИТЬСЯ СОБРАНИЕ 18

АВГУСТА 10 УТРА ВЦ АН СССР СТОИМОСТЬ ПУТЕВКИ 350

РУБЛЕЙ ДОПЛАТА РЕГИСТРАЦИОННОГО ВЗНОСА 10 РУБЛЕЙ

ВЪЕЗД ТУРИСТОВ 20-22 АВГУСТА ВЫЛЕТ ДЕЛЕГАТОВ МАРЧУКА

И ЕРШОВА 21 АВГУСТА =ДОРОДНИЦЫН=

*) Цифры, указанные после наименования пункта, откуда послана телеграмма, означают: первые две цифры — часы, следующие две цифры — минуты.

Рис. 2

В представленном на Конгрессе обзорном докладе «Теория схем программ» Ершовым был очерчен круг проблем теории схем программ, сопоставлены различные направления и модели этой теории, выработана общая система понятий, и связаны воедино разнообразные результаты и их применения, иначе говоря, создан фундамент теории схем программ как цельного направления теоретического программирования. Теоретические результаты в докладе рассматривались и в их практическом приложении к автоматизации программирования и к оптимизации программ. Важной для дальнейшего исследования была постановка ряда новых задач, связанных как с развитием теории, так и с ее приложениями. Текст данного приглашенного доклада Ершова был затем переиздан как одна из лучших работ по информатике за 1971 год [11].

На панельной дискуссии 6-го Конгресса ИФИП (ИФИП-74), который проходил с 5 по 10 августа 1974 г. в Швеции (г. Стокгольм), состоялось позиционное выступление А. П. Ершова «Программирование в 1980-х годах».

А. П. Ершов был организатором и председателем панельной дискуссии «Понимание естественных языков» 7-го Конгресса ИФИП (ИФИП-77), который прошел с 8 по 12 августа 1977 г. в Канаде (г. Торонто). На этом Кон-

грессе он также выступил с докладом А. П. Ершова и В. В. Грушецкого «Метод описания алгоритмических языков, ориентированный на реализацию» [12], который базировался на результатах, полученных в рамках проекта БЕТА.

На 8-м Конгрессе ИФИП (ИФИП-80), который состоялся в Японии (г. Токио) и в Австралии (г. Мельбурн) с 1 по 17 октября 1980 г., А. П. Ершов был заместителем председателя программного комитета.

4. 9-Й ВСЕМИРНЫЙ КОМПЬЮТЕРНЫЙ КОНГРЕСС ИФИП

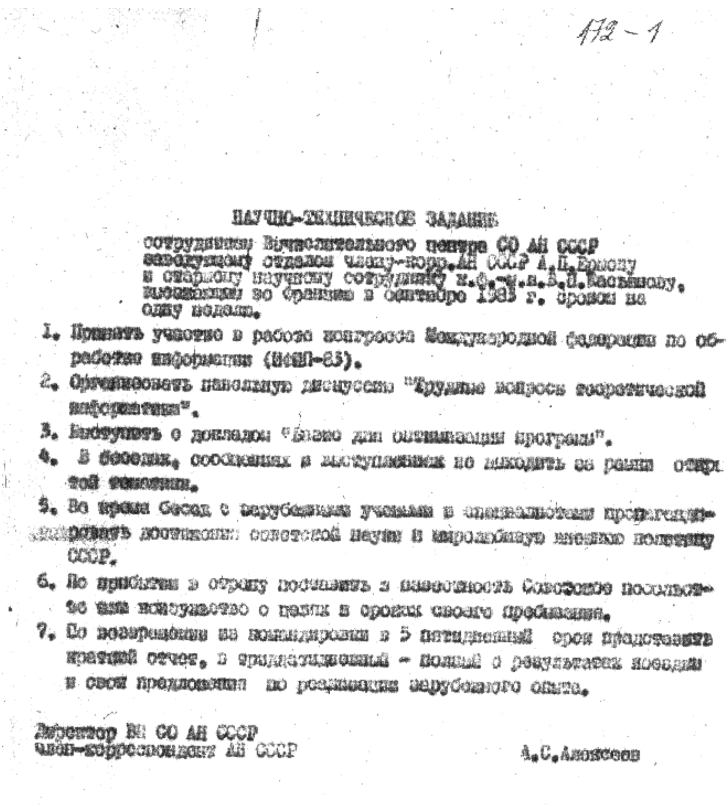
9-й Конгресс ИФИП (ИФИП-83) состоялся с 19 по 23 сентября 1983 г. во Франции (г. Париж). Это был последний из серии конгрессов ИФИП, на который выезжал А. П. Ершов. На этом Конгрессе он был организатором и участником панельной дискуссии «Крепкие орешки информатики», а также участником панельной дискуссии «Компьютерная грамотность».

Еще одним активным организатором Конгресса ИФИП-83 был другой сотрудник ВЦ, Вадим Евгеньевич Котов, в дальнейшем первый директор ИСИ и чл.-корр. АН СССР. Он был председателем международного программного комитета конференции «Теоретические основы обработки информации» – одной из десяти конференций, образующих Конгресс. Именно А. П. Ершов и В. Е. Котов стали теми сотрудниками ВЦ, которые вошли в советскую делегацию, выехавшую на Конгресс.

Надо сказать, что первоначально рассматривался в качестве кандидата на включение в формируемую советскую делегацию еще один сотрудник ВЦ – В. Н. Касьянов, доклад которого был включен в программу Конгресса. В архиве А. П. Ершова даже сохранилось научно-техническое задание чл.-корр. АН СССР А. С. Алексеева, тогда директора ВЦ, заведующему отдела чл.-корр. АН СССР А. П. Ершову и снс, к.ф.-м.н. В. Н. Касьянову, выезжающим во Францию для участия в конгрессе (Рис. 3).

Поскольку докладчик в делегацию все же не попал, возникла проблема представления доклада, которое гарантировалась советской стороной при включении доклада в труды Конгресса. Однако такая проблема возникла не в первый раз и уже имела отработанное решение: представление доклада другим участником делегации. Так, например, когда Я. М. Барздинь (в то время работавший в городе Рига) не смог приехать на Конгресс ИФИП-77 в Канаду, А. П. Ершов выступил с его докладом. Поэтому при формировании советской делегации на Конгресс ИФИП-83 статус докладчика хотя и учитывался, но уже не был определяющим. И когда А. П. Ершов сообщил мне,

что я не еду, и предложил прочитать мой доклад, я подготовил и передал ему презентацию в виде набора прозрачек, а также текст моего выступления с указанием тех мест, где нужно менять прозрачки. Однако, к сожалению, сам доклад на Конгрессе все же не состоялся. Дело в том, что, начиная с этого Конгресса, организаторы решили ужесточить требования и перестали разрешать выступать на Конгрессе с чужими докладами тем участникам, которые не имеют письменного согласия автора доклада на такую замену. Поскольку эти требования возникли неожиданно, о письменном согласии никто из нас и не подумал.



172-1

НАУЧНО-ТЕХНИЧЕСКОЕ ЗАДАНИЕ

сотрудникам Вычислительного центра СО АН СССР заведующему отделом члену-корр. АН СССР А.П. Ершову и старшему научному сотруднику к.ф.-м.н. В.Н. Касьянову, выезжающим во Францию в сентябре 1983 г. сроком на одну неделю.

1. Принять участие в работе конгресса Международной федерации по обработке информации (ИФИП-83).
2. Организовать панельную дискуссию "Трудные вопросы теоретической информатики".
3. Выступить с докладом "Знания для оптимизации программ".
4. В беседах, сообщениях и выступлениях не выходить за рамки открытой тематики.
5. Во время бесед с зарубежными учеными и специалистами пропагандировать достижения советской науки и миролюбивую внешнюю политику СССР.
6. По прибытии в страну поставить в известность Советское посольство или консульство о целях и сроках своего пребывания.
7. По возвращении из командировки в 5 пятидневный срок представить краткий отчет, в трехмесячный – полный о результатах поездки и свои предложения по реализации зарубежного опыта.

Директор ВЦ СО АН СССР
член-корреспондент АН СССР А.С. Алексеев

Рис. 3

Нельзя не упомянуть здесь и о технической проблеме, всегда возникавшей перед любым докладчиком в тот момент, когда его доклад принимался в программу Конгресса ИФИП. Это проблема подготовки в заданный весьма короткий срок оригинал-макета текста доклада. Нужно было в рамках этого срока получить по почте от издателей чистую специальную мелованную бумагу формата А3, напечатать на машинке с латинским шрифтом в две колонки текст статьи ровно на шести листах, получить разрешение областных организаций на вывоз рукописи заграницу и отослать напечатанный оригинал-макет издателям по почте так, чтобы он попал издателям вовремя. Современному ученому, в последний момент посылающему свою статью по электронной почте в виде PDF-файла и имеющему возможность послать любой документ быстрой почтой, эта проблема непонятна совсем. Мне же даже сегодня, четверть века спустя, живо вспоминаются все многочисленные (но преодоленные!) трудности, и все еще не верится, что эту проблему удалось решить.

Доклад В. Н. Касьянова «Базис для оптимизации программ» [13] был включен в программу конференции «Теоретические основы обработки информации». Он был посвящен теории крупноблочных схем, возникшей в рамках выполнения проекта многоязыковой системы программирования БЕТА и ставшей основой для создания универсального оптимизатора системы БЕТА [14].

Сегодня, когда существует .NET, никого не удивишь схемой построения трансляторов для n языков и m компьютеров, при которой за счет использования промежуточного языка вместо $n \times m$ трансляторов строится $n + m$ более простых трансляторов (рис. 4). Одной из центральных концепций такой схемы трансляции, разработанной и реализованной в рамках проекта БЕТА почти 30 лет тому назад, является концепция такого внутреннего (или промежуточного) языка, который позволяет с помощью универсальных алгоритмов оптимизации, выполняемых на уровне внутреннего языка, обеспечить получение качественной рабочей программы безотносительно ее происхождения. Таким образом, в системе БЕТА внутренний язык не только экранирует фазу генерации от конкретных входных языков, а фазу декомпозиции – от конкретных выходных, но и является входным и выходным языком фазы оптимизации (универсального оптимизатора системы БЕТА). Так как все оптимизирующие преобразования осуществляются на едином языковом представлении, возможны изменения как набора преобразований, так и порядка их выполнения. Возможно полное отключение оптимизации, а также ее повторное выполнение. При проектировании универсального оптимизатора предпочтение было отдано глобальным и уни-

версальным преобразованиям, а чтобы сохранить приемлемую сложность выполнения глобальных оптимизаций, фаза оптимизации была разбита на два последовательно работающих этапа – этап (потокового) анализа и этап преобразований. На первом этапе строится такое промежуточное представление программы, которое в дополнение к конструкциям внутреннего языка содержит специальные средства (так называемые тени), ориентированные на явное описание схемных свойств транслируемой программы и позволяющие придать преобразованиям более локальный и направленный характер. Второй этап состоит из преобразований, изменяющих как внутреннюю программу, так и ее тени (т. е. алгоритмы преобразований корректируют те схемные свойства, которые меняются при их выполнении).

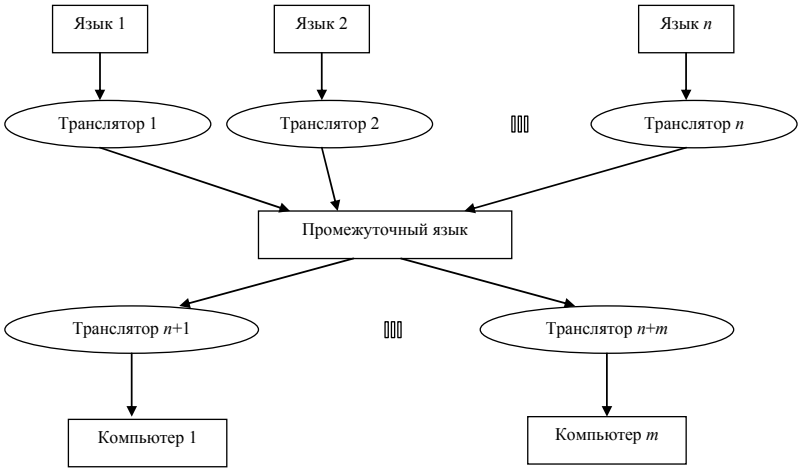


Рис. 4

Крупноблочная модель программы, положенная в основу предложенного подхода, описанного в докладе, носит комплексный и универсальный характер. Она отражает существующее разнообразие способов оптимизации и содержит в качестве подмоделей все используемые в оптимизации операторные модели программ. Класс крупноблочных схем программ вместе с концепцией крупноблочного моделирования одних схем другими, предложенной и изученной в докладе, дает единую позицию для комплексного исследования оптимизирующих преобразований программ со струк-

турами данных и действий и их совместного применения в системах конструирования программ. Существенными свойствами, отличающими класс крупноблочных схем от других моделей программ, является его универсальность в смысле широты описания класса последовательных программ и способов их оптимизации, а также его полнота – возможность построения по любой крупноблочной схеме (в частности, программе) и любому ее укрупнению операторов и переменных такой другой крупноблочной схемы, которая моделирует исходную при заданном ее укрупнении.

Модель представляет собой единый аппарат для всевозможных разноуровневых схемных представлений структуры вычислений, различающихся как по степени абстрагирования от семантики данных и операций (выполнение программы, программа, полуинтерпретированная схема программы, неинтерпретированная схема программы), так и по выбору тех объектов и действий программы, которые рассматриваются в качестве элементарных (неделимых), а также для изучения отношения моделирования, связанного с корректностью исследования эквивалентных и оптимизирующих преобразований в рамках одного схемного представления программы по отношению к другому.

5. 16-Й ВСЕМИРНЫЙ КОМПЬЮТЕРНЫЙ КОНГРЕСС ИФИП

16-й Всемирный компьютерный конгресс ИФИП (WCC-2000 или ИФИП-2000) состоялся с 21 по 25 августа 2000 г. в Китае (г. Пекин). Он прошел под лозунгом «Обработка информации. За рубежом 2000 года», и в его работе приняло участие более 2 тысяч ученых и специалистов из 70 стран мира. Впервые конгресс проводился в развивающейся стране – Китае, что было не случайным, а подтвердило особый статус Китая в области развития информационных технологий. Главным организатором конгресса ИФИП-2000 с китайской стороны являлся Китайский институт электроники (CIE) –неправительственная академическая и инженерная организация Китая с большим числом профессиональных и региональных отделений по всей стране [15]. Основан CIE в 1956 г., независимый академический статус получил в 1962 г. В 1979 г. он стал членом ИФИП, а в начале 80-х гг. прошлого столетия установил отношения со всеми 15 основными международными организациями в области информатики. Китайский институт электроники издает 12 журналов и периодических изданий, в том числе журнал «Acta Electronica Sinica» и популярный еженедельник «PC Week».

Надо сказать, что к 2000 г. такой страны как СССР уже не существовало, и хотя финансирование научных исследований в России в целом резко сократилось, и многие ученые и академические институты находились на грани выживания, для ученых из бывшего Советского Союза появились новые возможности участия в международных мероприятиях. В частности, был организован Российский фонд фундаментальных исследований (РФФИ), который на конкурсной основе стал выдавать гранты для поддержки исследований отдельных небольших коллективов ученых. В бюджетах этих грантов можно было предусматривать разного типа расходы, в том числе и на командировки. В виде отдельного направления своей работы РФФИ стал выделять так называемые тревел-гранты (travel grants) для поддержки участия российских специалистов в международных мероприятиях. Причем первоначально эти гранты предусматривали покрытие лишь транспортных расходов выезжающего, но зато покрывались все эти расходы (от места жительства ученого до места проведения конференции), правда, по самому низкому тарифу. В дальнейшем эти правила изменились, и фонд стал выдавать фиксированные суммы, разрешив использовать их на покрытие любых расходов, связанных с поездкой. Таким образом, к настоящему времени сложилась парадоксальная ситуация, при которой за счет такого гранта москвич может покрыть почти все расходы по поездке, а сибиряк или дальневосточник – лишь оплатить часть транспортных расходов, например, стоимость проезда от места жительства до Москвы.

Гранты РФФИ, а также финансовая поддержка, неизменно оказываемая мне со стороны ИФИП, позволили мне в этот раз и в дальнейшем полноценно участвовать в работе Конгрессов ИФИП. На Конгрессе ИФИП-2000 я выступил с двумя докладами: «Иерархические графовые модели и визуальная обработка» (совместно с В. А. Лисицыным) и «СИМИКС: информационная система по истории информатики».

К слову сказать, без поддержки РФФИ успешная работа нашей лаборатории вряд ли была бы возможной, особенно в 90-е годы прошлого столетия, когда вопросы бюджетного финансирования стояли особенно остро. Достаточно отметить, что все оборудование, имеющееся в лаборатории, и все расходные, комплектующие и другие материалы, используемые сотрудниками лаборатории в своих исследованиях, – все это приобреталось и до сих пор приобретается исключительно за счет средств грантов РФФИ, которых было выполнено под моим руководством более 15 [16]. Среди них – гранты на поддержку инициативных научных проектов (93-01-00576, 98-01-00748, 01-01-00794), ведущих научных школ (96-01-00134), проектов

создания и развития информационных, вычислительных и телекоммуникационных ресурсов для проведения фундаментальных исследований (95-05-19269, 01-01-14051), развития материально-технической базы (94-01-00816), участия российских ученых в международных научных мероприятиях за рубежом (97-01-00936, 98-01-10941, 00-01-10892, 01-01-10545, 02-01-10747, 05-07-93547, 06-01-10660), проведения ориентированных фундаментальных исследований (07-07-12050), а также издательских проектов (95-01-01334, 01-01-14051).

Конгресс ИФИП-2000 проходил в так называемой «деревне азиатских игр», расположенной в северной части Пекина, в виде следующих 8 отдельных конференций:

1. Международная конференция по коммуникационным технологиям (ICCT-2000).
2. Международная конференция по автоматизации проектирования чипов (ICDA-2000).
3. Международная конференция по использованию информационных и коммуникационных технологий в образовании (ICEUT-2000).
4. Международная конференция по программному обеспечению: теории и практике (ICS-2000).
5. Международная конференция по обработке сигналов (ICSP-2000).
6. Международная конференция по интеллектуальной обработке информации (IIP-2000).
7. Международная конференция по информационной технологии управления бизнесом (ITBM-2000).
8. Международная конференция по защите информации (SET-2000).

Конференциям предшествовал общий день заседаний, включающий церемонию открытия, на которой присутствовал и выступил с приветствием Председатель Госсовета КНР Цзян Цзэмин, а также ряд пленарных докладов. Помимо приветствий руководителей ИФИП и организаторов конгресса на церемонии открытия состоялось вручение премии имени Исаака Ауербаха, основателя ИФИП. Премия была вручена проф. Асберну Ролстадесу (Норвегия), он стал четвертым обладателем этой престижной премии ИФИП.

По-видимому, это был первый и, скорее всего, последний Конгресс ИФИП, на котором выступал руководитель той страны, где он проходил. По крайней мере, я помню то громадное впечатление, которое оказало на участников конгресса присутствие на нем столь высокого лица, а также те беспрецедентные меры безопасности, которые сопутствовали этому участию.

В своем выступлении на церемонии открытия конгресса Цзян Цзэмин отметил, что мир охвачен технологической революцией, для которой информационные и генные технологии будут флагманскими кораблями. Он также отметил, что физические ресурсы в мире ограничены, а потому информационные ресурсы будут играть все более важную роль. Цзян Цзэмин считает, что сектор промышленности должен интегрироваться с сектором программного обеспечения и необходимо соединение традиционной индустрии и информационных сетей. Он напомнил о все расширяющемся технологическом разрыве между развитыми и развивающимися странами и призвал развитые страны помочь развивающимся нациям преодолеть этот разрыв. Цзян Цзэмин предупредил, что бесконтрольные действия в Интернете хакеров приводят к нарушению неприкосновенности частной жизни, потерям информации и секретности, и призвал международную общественность к принятию международного соглашения по Интернету, которое позволило бы повысить уровень обслуживания пользователей Интернета и обеспечить гарантии их интересов.

Доклад В. Н. Касьянова и В. А. Лисицына «Иерархические графовые модели и визуальная обработка» [17] состоялся на Международной конференции по программному обеспечению: теории и практике (ICS-2000).

Широкая применимость графов в информатике и программировании связана с тем, что они являются естественным и наглядным средством объяснения сложных ситуаций на интуитивном уровне. Эти преимущества представления сложных структур и процессов графами становятся еще более ощутимыми при наличии хороших средств их визуализации.

Поскольку информация, которую желательно визуализировать, постоянно увеличивается и усложняется, возникает все больше ситуаций, в которых классические графовые модели перестают быть адекватными. Требуются более мощные графовые формализмы для представления информационных моделей, обладающих иерархической структурой. Иерархичность является основой многочисленных методов анализа и синтеза сложных информационных моделей в различных областях применения ЭВМ. Новые формализмы должны в большей степени поддерживать визуализацию семантических аспектов информационных моделей. Понятно, что визуальная обработка информационных моделей не сводится к проблемам визуализации их структурных свойств, а включает также широкий круг вопросов, связанных с поддержкой различных процессов анализа и синтеза графовых моделей с использованием человеко-машинного интерфейса, основанного на их наглядных представлениях.

В докладе обсуждаются предложенные авторами понятия иерархических графов и графовых моделей [18], включая вопросы их представления и визуальной обработки, а также рассматривается созданная инструментальная система HIGRES, предназначенная для поддержки визуальной обработки иерархических графов и графовых моделей. Система HIGRES обладает удобным современным графическим интерфейсом, позволяет не только быстро создавать и редактировать структуру иерархического графа, но и задавать его семантику. В системе предусмотрены возможности ее специализации, расширения библиотеки алгоритмов обработки графовых моделей и их полноценной динамической визуализации (анимации). Система работает под Windows 95/98/NT, что выгодно отличает ее от западных аналогов, которые в подавляющем большинстве ориентированы на использование графической оболочки Xwindow операционной системы UNIX, и доступна для бесплатного некоммерческого использования [19].

Доклад В. Н. Касьянова «СИМИКС: информационная система по истории информатики» [20] был представлен на Международной конференции по использованию информационных и коммуникационных технологий в образовании (ICS-2000). В нем были рассмотрены страницы системы СИМИКС, посвященные истории информатики в Сибири. Информационная система СИМИКС, созданная в ИСИ СО РАН при поддержке РГНФ (грант N 96-04-12030), ориентирована на накопление и представление знаний в области современной музыки и киноискусства, а также в области истории развития информатики, поскольку, на наш взгляд, информатика в современном обществе является важным элементом культуры.

Информатика сформировалась как наука в середине 50-х годов прошлого столетия. Это достаточно молодая область науки в сравнении с такими классическими направлениями как, например, математика, физика, химия, астрономия и прочие. Но тем интереснее проследить, изучить и отразить в структурированном виде (используя при этом методы самой информатики) историю ее становления и развития, которая происходила и происходит на наших глазах. При этом мы стремились отразить особенности ее развития именно в Сибирском регионе, поскольку после организации в конце пятидесятых годов Сибирского отделения Академии наук он стал крупным центром развития информатики.

6. 17-Й ВСЕМИРНЫЙ КОМПЬЮТЕРНЫЙ КОНГРЕСС ИФИП

17-й Всемирный компьютерный конгресс ИФИП (WCC-2002 или ИФИП-2002) проходил с 24 по 31 августа 2002 года в Канаде (г. Монреаль). В этот раз конгресс проводился в виде 11 отдельных конференций (потоков):

- Основы информационных технологий в эпоху сетевых и мобильных вычислений (TCS 2002);
- Архитектура программного обеспечения;
- Теле-обучение;
- Коммуникационные системы: текущее состояние;
- Информационные системы: вызовы е-бизнеса;
- Гуманитарный выбор и компьютеры: вопросы выбора и качества жизни в информационном обществе;
- Распределенные и параллельные встроенные системы (DIPES 2002);
- Интеллектуальная информационная обработка (IIP-2002);
- Применимость: достижение конкурентоспособной грани;
- Специальный поток по секретности;
- Индустриальный поток.

Конференции сопровождались общими заседаниями, на которых происходили церемонии открытия и закрытия конгресса, состоялся лекторий из 17 циклов лекций, было прочитано четыре ключевых приглашенных доклада, а также состоялось вручение премии Исаака Ауербаха, основателя ИФИП. Премия была вручена в пятый раз; ее был награжден проф. Вильфред Брауер (Wilfried Brauer, Германия) – известный специалист в области теоретической информатики.

На Конгрессе ИФИП-2002 состоялось четыре приглашенных доклада.

- М. Лазаридис (M. Lazaridis, президент Research in Motion Limited, Канада) выступил на тему «Образование и наука: ключи к инновациям».
- К. Тачикава (K. Tachikawa, президент NTT DoCoMo, Inc., Япония) представил свой взгляд на мобильные коммуникационные стратегии в будущем.
- А. Хан (A. Khan, зам. директора по коммуникациям и информатике ЮНЕСКО, Франция) рассказал о роли ЮНЕСКО в повышении справедливого участия в обществе знаний.

- Ж. Паютте (J. Payette, главный астронавт Canadian Space Agency, Канада) выступила на тему: «Работа в космосе: проблемы, привилегия, возможность».

Если в случае Конгресса ИФИП-80 я смог оформить визу в Китай, минуя УВС РАН, поскольку в тот же год до поездки в Китай выезжал в Барселону на Третий европейский конгресс по математике (3ECM) и на обратном пути задержался в Москве для оформления визы в Посольстве Китая, то для получения визы в Канаду на Конгресс ИФИП-2002 был вынужден обратиться в УВС РАН. Это означало оформление выездного дела по поездке и подготовку научного отчета для УВС после командировки. Надо сказать, что в то время УВС еще функционировал, но работал все с большей пробуксовкой. Ему все труднее становилось сохранять кадры при тех низких зарплатах, которые тогда были у сотрудников Академии наук РАН, на фоне тех возможностях большого заработка, которые появлялись для кадров такой квалификации в это время в Москве в связи открытием границ в новой России. Дополнительная трудность состояла также в том, что официальное приглашение с финансовой поддержкой от оргкомитета Конгресса я получил довольно поздно. Поэтому решение о покупке авиабилета до Монреаля, хотя и по самому низкому тарифу, не предусматривающему возврат билета, но весьма дорогого, я вынужден был принимать, не зная, будет ли у меня виза или нет, хотя она в этот момент уже была. Дело в том, что курьер УВС РАН отказывался узнавать о решении посольства по моей заявке, мотивируя это своей занятостью и тем, что времени от сдачи документов в посольство прошло еще недостаточно, несмотря на все указания его руководства, вызванные просьбами руководства УВС СО РАН.

Работа конференции «Теле-обучение», на которой состоялся мой доклад «Графы в информатике: методы и инструменты», велась по трем основным направлениям: подготовка преподавателей (19 докладов), пожизненное обучение – профессиональное развитие (14 докладов), технологии обучения (18 докладов). Программа конференции включала три приглашенных доклада, секционные заседания, круглые столы и панельные дискуссии. Темы секционных заседаний – это проекты обучения учителей, пересекающиеся границы, национальная политика, инструменты тестирования, моделирование, организации созидательного обучения, среды обучения, управление знаниями, изменение и реформа, человеческий выбор, развитие сред теле-обучения, разработка инструментов для теле-обучения, теле-обучение как стратегия для изменения, развитие совместной работы в теле-обучении, достижение применимости для теле-обучающих, системы теле-обучения для учителей.

Основной тезис моего доклада «Графы в информатике: методы и инструменты» состоял в том, что современное состояние информационного общества нельзя представить себе без применения теоретико-графовых методов и алгоритмов, а, следовательно, нужны адекватные средства поддержки этого применения.

Теория графов стала активно применяться на заре программирования в силу удобного выражения задач обработки информации на теоретико-графовом языке. Расширение круга задач, решаемых на ЭВМ, потребовало выхода на модели дискретной математики, что привело к подлинному расцвету теории графов и комбинаторики, которые за сорок лет трансформировались из разделов «досуговой» математики в первостепенный инструмент решения огромного числа задач. Андрей Петрович Ершов называл графы основной конструкцией для программиста и говорил, что «графы обладают огромной, неисчерпаемой изобразительной силой, соразмерной масштабу задачи программирования».

В докладе я постарался представить наши работы по созданию средств поддержки применения графов в информатике и программировании, разделив их на три направления [21].

В начале доклада я охарактеризовал наши работы по созданию серии книг «энциклопедии» алгоритмов на графах для программистов, в основе которой лежит разделение алгоритмов на классы по типам графов, используемых ими в качестве модели. В отличие от Д. Кнута в этих книгах мы ориентируемся на высокоуровневое описание алгоритмов в терминах специального псевдоязыка (лексикона) программирования, содержащего традиционные конструкции математики и языков программирования высокого уровня. Такой подход позволяет формулировать алгоритмы в естественной форме, допускающей прямой анализ их корректности и сложности, а также простой перенос алгоритмов на традиционные языки программирования и ЭВМ с сохранением полученных оценок сложности. Кроме того, подобный стиль описания алгоритмов является базой для доказательного стиля программирования: он позволяет понять алгоритм на содержательном уровне, оценить пригодность его для решения конкретной задачи и осуществить модификацию алгоритма, не снижая степень математической достоверности окончательного варианта программы.

Далее в докладе я обсудил предложенные мной понятия иерархических графов и графовых моделей, включая вопросы их представления и визуальной обработки, а также рассмотрел особенности наших инструментальных систем HIGRES и VEGRAS для поддержки визуальной обработки графов и графовых моделей. VEGRAS – это универсальный и простой в использова-

нии редактор атрибутированных графов, в том числе иерархических, ориентированный на поддержку подготовки качественных штриховых иллюстраций в рамках Windows 95/98/NT. VEGRAS поддерживает обмен изображениями с другими приложениями Windows, включая систему HIGRES.

Доклад завершился обсуждением проблемы терминологии, которая, без сомнения, является одной из основных проблем в применении теоретико-графовых методов в программировании и информатике. В этой части доклада я представил недавно опубликованный наш словарь по теории графов [22], который призван если не решить, то значительно облегчить эту проблему. В нем впервые собраны наиболее употребительные термины по теории графов и ее приложениям в информатике и программировании. Статьи словаря снабжены иллюстрациями, перекрестными ссылками и ссылками на доступную литературу. Наличие английских эквивалентов терминов позволяет использовать словарь при переводе с русского на английский и обратно. Рассказал я также о наших работах по созданию электронной версии словаря, которая получила название GRAPP [23].

7. 20-Й ВСЕМИРНЫЙ КОМПЬЮТЕРНЫЙ КОНГРЕСС ИФИП

20-й Всемирный компьютерный конгресс ИФИП (WCC-2008 или ИФИП-2008) стал событием, посвященным наукам, информационным технологиям и коммуникациям (ICT). Это был первый Конгресс ИФИП, который прошел в Италии.

В течение 4 дней конгресса, почти 2000 делегатов, приехавших из 70 стран, обсуждали главные актуальные проблемы и перспективы области информационных технологий и коммуникаций в обществе знаний 21 столетия.

Основная работа Конгресса ИФИП-2008 проходила в виде 12 технических конференций:

- Биологически инспирированные кооперативные вычисления;
- Распределенные и параллельные встроенные системы;
- 1-й симпозиум по представлению вычислений;
- Обучение для жизни в обществе знания;
- 3-я Международная конференция по истории информатики и образования;
- Человеко-машинное взаимодействие;
- Искусственный интеллект 2008;

- 23-я Международная конференция по информационной безопасности;
- Успехи в исследовании, обучении и практике информационных систем;
- Управление знаниями в действии;
- Системы открытого кода 2008;
- 5-я Международная конференция по теоретическим компьютерным наукам.

Помимо технических конференций программа Конгресса ИФИП-2008 включала сессии открытия и закрытия, различные тематические сессии, многочисленные выставки, а также так называемые кросс (или междисциплинарные) сессии и конференции. Отдельные междисциплинарные сессии имели дело с различными интересными темами, такими как электронное включение, электронное правительство, ICT для культурного наследия, ICT для окружающей среды, ICT профессионализм и компетентность, ICT для здоровья, Интернет второго поколения и ICT для образования.

В частности, междисциплинарная сессия и конференция по электронному включению фокусировались на рассмотрении хорошей практики и планируемых инициатив как Европейской комиссии, так и Итальянского правительства. Междисциплинарная сессия по электронному правительству представила наилучшую практику и результаты, достигнутые Итальянской администрацией как на центральном, так и местном уровнях.

Конгресс ИФИП-2008 дал возможность участникам и гостям встретиться и послушать всемирно признанных экспертов в области ICT. Кроме того, он предоставил возможности для установления сотрудничества между исследователями, прикладниками, промышленниками и преподавателями. Таким образом, в нем проявился новый итальянский подход, направленный на поддержку соединения специфических предметов технических конференций и междисциплинарных предметов итальянской реальности.

На Конгрессе ИФИП-2008 я выступил с двумя докладами: «WAPE – система дистанционного обучения программированию» (совместно с Е В. Касьяновой) и «Открытый адаптивный виртуальный музей по истории информатики в Сибири».

Доклад «Открытый адаптивный виртуальный музей по истории информатики в Сибири» [24] состоялся на 3-й Международной конференции по истории информатики и образования. Он был посвящен нашим исследованиям в области виртуальных музеев, направленных на создание виртуального музея SVM истории информатики в Сибири, работа над которым велась при финансовой поддержке при финансовой поддержке Российского

гуманитарного научного фонда (грант РГНФ № 02-05-12010). Основная цель создания музея SVM – это сохранение историко-культурного наследия, связанного с созданием и развитием информационных ресурсов, являющихся важнейшим национальным богатством, а также обеспечение свободного повсеместного доступа к ним с целью повышения общеобразовательного и культурного уровня широких слоев населения.

Под виртуальным музеем нами понимается репозиторий цифровых культурных или научных ресурсов, к которым есть доступ, и которые могут использоваться в любое время и с любого места, где есть выход в Интернет. Это означает, что виртуальный музей – это сайт (цифровой музей), который может, но не обязан иметь соответствующий реальный музей и который содержит виртуальные экспонаты, являющиеся мультимедийными представлениями любых артефактов без каких либо ограничений на их природу или текущее состояние. Мы вводим понятие открытого виртуального музея как гипермедиа-системы, предназначенной быть как доступным репозиторием для коллекций артефактов, так и институтом культурного наследия, поддерживающим совместную работу многих людей (пользователей музея), заинтересованных в сборе, аннотировании, организации, исследовании, каталогизации и демонстрации этих артефактов. Другими словами, в отличие от посетителя обычного (реального или виртуального) музея пользователь открытого музея может не только участвовать в экскурсиях и посещать выставки, но и работать сотрудником музея. Как адаптивная гипермедиа система адаптивный виртуальный музей поддерживает модель целей, предпочтений и знаний каждого индивидуального пользователя музея и использует эту модель во время взаимодействия с данным пользователем для того, чтобы настраиваться на его потребности.

В докладе кратко описана сибирская школа информатики и программирования, рассматривается структура создаваемого нами открытого адаптивного виртуального музея SVM по истории информатики в Сибири и его содержимое, описывается пользовательский интерфейс музея и его пользователи.

Информатика сформировалась как наука в середине 50-х годов прошлого столетия и за прошедший полувековой период шагнула далеко вперед. С годами от нас уходят активные участники и свидетели ее первых шагов, многое забывается, становится труднодоступным или безвозвратно утерянным. Постоянное развитие информатики и сверхмощное давление зарубежной вычислительной науки усиливают этот процесс, и нужны целенаправленные действия, чтобы богатый отечественный опыт не забывался и мог быть востребован. Без понимания прошлого трудно двигаться вперед.

Нужно сказать, что исследования по истории информатики в передовых странах мира ведутся достаточно широко. Вместе с тем до недавнего времени история информатики в бывшем Советском Союзе была практически неизвестна на Западе, хотя отдельные работы, посвященные этим вопросам, публиковались [25, 26], а в 1996 г. компьютерное общество IEEE Computer Society в связи с 50-ой годовщиной своего основания наградило самой престижной медалью «Computer Pioneer» Виктора Михайловича Глушкова, Сергея Алексеевича Лебедева и Алексея Андреевича Ляпунова [27]. Одновременно этой награды был удостоен и ряд ученых из стран Восточной Европы (Григоре Моисил, Иван Пландер, Антонин Свобода и другие). Медаль «Computer Pioneer» была учреждена в 1981 году, чтобы признать и представить общественности выдающихся ученых, усилиями которых создавалась и развивалась сфера вычислительной и информационной науки и техники. Среди 55 лауреатов этой награды такие замечательные личности, как Артур Беркс, Джон Маккарти, Марвин Минский, Никлаус Вирт, Хайнц Земанек. Согласно формулировке награждения В. М. Глушков «основал первый в СССР Институт Кибернетики на Украине, разработал теорию цифровых автоматов и компьютерной архитектуры, а также рекурсивный макроконвейерный процессор», С. А. Лебедев «разработал и построил первый советский компьютер и основал советскую компьютерную промышленность», А. А. Ляпунов «разработал теорию операторных методов для абстрактного программирования и основал советскую кибернетику и программирование» [27].

В настоящее время музей SVM в качестве экспонатов содержит описания ученых-информатиков, коллективов, хронологий событий, проектов, публикаций, конференций и архивных материалов. Пользователи SVM могут не только пополнять экспонатами музей и высказывать предложения и замечания, но и создавать свои авторские экскурсии и экспозиции.

Открытость виртуального музея по определению означает его доступность любым пользователям из любой точки земного шара в любое время. При этом посетитель нашего виртуального музея может не только ознакомиться с экспонатами виртуальных экспозиций, но и сам принять участие в его развитии, став зарегистрированным пользователем, и получив тем самым возможность дополнять материалы, пополнять экспозиции музея, вносить свои поправки и высказывать пожелания. Таким образом, контент нашего виртуального музея может создаваться множеством людей, которые будут приносить в виртуальный музей истории информатики в Сибири новые тексты, фотографии, видео- и аудиофайлы и любые другие документы, соответствующие заявленной тематике виртуального музея. Такая деятельность вписывается в рамки международного проекта «Музей без границ».

Поэтому наш виртуальный музей является частицей открытых информационно-коммуникационных обществ и обществ знаний в сети Интернет. Мы не ограничиваемся распространением знаний, собранных в виртуальном музее истории информатики в Сибири сугубо региональными или государственными рамками, а делаем его открытым для всех желающих из любых стран мира через международную сеть, которой является Интернет.

Доклад «WAPE – система дистанционного обучения программированию» [28] состоялся на конференции «Обучение для жизни в обществе знания». Он был посвящен проекту расширяемой среды, поддерживающей дистанционное обучение программированию в рамках проблемного подхода и соединяющей возможности адаптивных гипермедиа-систем и интеллектуальных обучающих систем. Среда ориентирована на поддержку дистанционного обучения, в процессе которого студенты (обучаемые), решая поставленные им индивидуальные задачи, действуют вполне самостоятельно, но постоянно обеспечены возможностью получения квалифицированной помощи, начиная с этапа понимания условий задач и кончая этапом оценки правильности их решения. Помимо студентов, среда поддерживает и других пользователей, вовлеченных в учебный процесс (лекторов, ассистентов и администраторов), предоставляя им свои интерфейсы и возможности. Так, например, на специально разработанном языке ЯЗП лектор может задавать одновременно все варианты нового проекта удобным и компактным образом.

Многие адаптивные системы отслеживают движение пользователя по гиперкниге. Хотя это оправданный подход, его недостатком является трудность измерения знания, приобретенного пользователем во время чтения Web-страницы. Вместо этого мы используем для обновления модели знаний только успехи и неуспехи студента, проявленные им при решении задач: тестов, заданий и упражнений. Другое важное отличие нашего подхода от традиционного состоит в том, что мы не пытаемся оценивать успех студента в изучении курса на основании уровня знаний в его модели. Поэтому в нашей системе достижение определенного уровня знаний студентом не позволяет ему завершить изучение курса с определенной оценкой, а лишь дает студенту возможность приступить к решению той или иной задачи из его индивидуального набора. Все это мотивировано проблемным подходом к обучению, поддерживаемым системой WAPE.

ЗАКЛЮЧЕНИЕ

ИФИП является удачным примером эффективно работающей общественной международной организации, объединяющей как отдельные физические лица, так и целые организации, и играющей наиболее существенную роль в мире в разработке, распространении и применении информационных технологий на пользу человечеству. Следует обратить внимание на ориентацию организации на кооперацию исследований в международном масштабе.

Всемирные компьютерные конгрессы ИФИП были и продолжают оставаться наиболее представительным научным форумом, определяющим современный уровень развития вычислительного дела. Поэтому представительное участие российских информатиков в этих конгрессах и как можно более широкое ознакомление российской научной общественности с итогами Конгрессов ИФИП следует признать весьма важным.

Представление результатов на Конгрессах ИФИП является наиболее эффективным способом сделать их достоянием мировой общественности, но не всегда доступным для российских ученых. Так, например, хотя за последние четверть века 6 моих докладов были включены в программы четырех конгрессов ИФИП, мне ни разу так и не посчастливилось войти ни в одну из делегаций, выезжающих на Конгрессы, ни в бытность СССР, ни в уже новой России. Поэтому участие в других международных конгрессах и конференциях (я выступал более чем со 110 докладами на международных научных мероприятиях [16]), а также зарубежные публикации (в частности, статьи в международных журналах и монографии [29 –33]) и статьи в переводимых на английский язык советских (российских) журналах [34– 47] все же являются для меня основными возможностями донести полученные результаты до мировой научной общественности.

СПИСОК ЛИТЕРАТУРЫ

1. Международная федерация по обработке информации (ИФИП) – http://www.ifip.org.
2. Награды Международной федерации по обработке информации (ИФИП) – http://www.ifip.org/awards.htm.
3. Электронный архив академика А. П. Ершова . – http: // www.iis.nsk.su.
4. Kasyanov V. N., Pottosin I. V. Application of optimization techniques to correctness problems // Constructing Quality Software, Ed. by P. G. Hidbard and S. A. Schuman. – Amsterdam, North-Holland, 1979. – P. 237–248. – (Proc. IFIP Working Conf.).

5. Касьянов В. Н., Поттосин И. В. Применение методов оптимизации к проверке правильности программ // Создание качественного программного обеспечения. – Новосибирск: ВЦ СО АН СССР, 1978. – Т. 1. – С. 225–237.
6. Ershov A. P. The ALFA automatic programming system // Information processing 65. – Washington-London, 1965. – Vol. 2. – P. 622–623. – (Proc. IFIP Congress 65).
7. Ershov A. P., Marchuk G.I. Man-machine interaction in solving a certain class of different equations // Information processing 65. – Washington–London, 1965. – Vol. 2. – P. 550–551. – (Proc. IFIP Congress 65).
8. АЛЬФА – система автоматизации программирования / Г. И. Бабецкий и др. – Новосибирск: Наука, 1967. – 308 с.
9. Ershov A. P. Time sharing: the need for reorientation // Information processing 68. – Amsterdam, North-Holland, 1969. – P. 1615–1616. – (Proc. IFIP Congress 68).
10. Ershov A. P. Theory of program schemata // Information processing 71. – Amsterdam, North-Holland, 1971. – P. 28–45. – (Proc. IFIP Congress 71).
11. Ershov A. P. Theory of program schemata // The Best Computer Paper of 1971. – Princeton, Auerbach, 1971. – P. 93–124.
12. Ershov A. P., Grushetsky V. V. An implementation-oriented method for describing algorithmic languages // Information processing 77. – Amsterdam, North-Holland, 1977. – P. 117–122. – (Proc. IFIP Congress 77).
13. Kasyanov V. N. Basis for program optimization // Information processing 83. – Amsterdam, North-Holland, 1983. – P. 315–320. – (Proc. IFIP Congress 83).
14. Касьянов В. Н. Оптимизирующие преобразования программ. – М.: Наука , 1988, – 336 с.
15. Китайский институт электроники CIE. – http: // www.cie-china.org.
16. Виктор Николаевич Касьянов. К 60-летию со дня рождения. – Новосибирск, ИСИ СО РАН, 2008. – 136 С. – (http: // pco.iis.nsk.su/~kvn).
17. Kasyanov V. N., Lisitsyn I. A. Hierarchical graph models and visual processing // Proc. of Intern. Conf. on Software: Theory and Practice (ICS-2000). 16th IFIP World Computer Congress. – Beijing, PHEI, 2000. – P. 179–182.
18. Касьянов В. Н. Иерархические графы и графовые модели: вопросы визуальной обработки // Проблемы систем информатики и программирования. – Новосибирск, 1999. – С. 7–32.
19. Система HIGRES – http://pco.iis.nsk.su/higres.
20. Kasyanov V. N. SIMICS: information system on informatics history // Proc. of Intern. Conf. on Educational Uses of Information and Communication Technologies (ICEUT). 16th IFIP World Computer Congress. – Beijing, PHEI, 2000. – P. 168.
21. Касьянов В. Н. Применение графов в программировании // Программирование. – 2001. – N 3. – С. 51–70.
22. Евстигнеев В. А., Касьянов В. Н. Толковый словарь по теории графов в информатике и программировании. – Новосибирск: Наука, 1999. – 288 с.
23. Электронный словарь по теории графов GRAPP – http://pco.iis.nsk.su/grapp.
24. Kasyanov V. N. An open adaptive virtual museum of informatics history in Siberia // IFIP International Federation for Information Processing. – Boston: Springer, 2008. –

Vol. 266. History of Computing and Education 3 (HCE 3). – P. 129–146. – (Proc. of the 20th IFIP World Computer Congress).
25. Ershov A. P. A history of computing in the USSR // Datamation. – 1975. – Vol. 21, N 9. – P. 80–88.
26. Ershov A. P., Shura-Bura M.R. The early development of programming in the USSR // A History of Computing in the Twentieth Century. – New York: Acad. Press, 1980. – P. 137–196.
27. CS Recognizes Pioneers in Central and Eastern Europe. // IEEE Computer. – 1998. – N 6. – P. 79–84.
28. Kasyanov V. N., Kasyanova E. V. WAPE – a system for distance learning of programming // IFIP International Federation for Information Processing. – Boston: Springer, 2008. – Vol. 261. Learning to Live in the Knowledge Society. – P. 355–357. – (Proc. of the 20th IFIP World Computer Congress).
29. Kasyanov V. N. Some properties of fully reducible graphs // Information Processing Letters. – 1973. – Vol. 2, N 4. – P. 113–117.
30. Kasyanov V. N. Loop cleaning // Information Processing Letters. – 1984. – Vol. 18, N 1. – P. 1–6.
31. Kasyanov V. N. Transformational approach to program concretization // Theoretical Computer Science. – 1991. – Vol. 90, N 1. – P. 37–46.
32. Kasyanov V. N., Evstigneev V. A. Graph theory for programmers. Algorithms for processing trees. – Dordrecht/Boston/London, Kluwer Academic Publishers, 2000. – 432 p.
33. Kasyanov V. N., Stasenko A. P., et all. SFP – an Interactive Visual Environment for Supporting of Functional Programming and Supercomputing // WSEAS Transactions on Computers. – 2006. – Vol.5, Iss. 9. – P. 2063–2069.
34. Касьянов В. Н. Выделение гамаков в ориентированном графе // ДАН СССР. – 1975. – Т. 221, № 5. – С.1020–1022.
35. Касьянов В. Н. К оценке частоты выполнения операторов и переходов в программе // Программирование. – 1975. – № 5. – С. 64–72.
36. Касьянов В. Н. О нахождении аргументов и результатов в схемах с косвенной адресацией // Программирование. – 1976. – № 1. – С. 6–15.
37. Вальковский В. А., Касьянов В. Н. Крупноблочная сегментация и распараллеливание схем программ // Программирование.– 1976. – № 1. – С. 16–26.
38. Касьянов В. Н. К вопросу о реализации схем над распределенной памятью // Кибернетика. – 1977. – N 1. – С. 63–68.
39. Касьянов В. Н. Анализ структур программ // Кибернетика. – 1980. – № 1. – С. 48–61.
40. Касьянов В. Н., Поттосин И. В. Технологические возможности оптимизации программ // Программирование. – 1980. – № 2. – С. 27–31.
41. Касьянов В. Н. Смешанные вычисления и оптимизация программ // Кибернетика. – 1980. – № 2. – С. 51–54.
42. Касьянов В. Н. К обоснованию алгоритмов преобразования крупноблочных программ // Программирование. – 1981. – № 3. – С. 15–25.

43. Касьянов В. Н. Эквивалентные преобразования кратных схем // Программирование. – 1982. – № 2. – С. 3–8.
44. Касьянов В. Н. Аннотирование программ и их преобразование // Программирование. – 1989. – № 4. – С. 3–16.
45. Касьянов В. Н. Трансформационный подход к конкретизации программ // Кибернетика. – 1989. – № 6. – С. 28–32.
46. Касьянов В. Н. Трансформационные методы и средства конструирования эффективных и надежных программ // Кибернетика и системный анализ. – 1993. – № 2. – С.30–39.
47. Евстигнеев В. А., Касьянов В. Н. Оптимизирующие преобразования в распараллеливающих компиляторах // Программирование. – 1996. – № 6. – С.12–26.

С. Н. Касьянова

ИСПОЛЬЗОВАНИЕ КЛАСТЕРОВ ПРИ ВЫЧИСЛЕНИИ ПРЕОБРАЗОВАНИЯ МЕЛЛИНА ДЛЯ ФУНКЦИЙ В ЗАДАЧАХ ТОМОГРАФИИ

ВВЕДЕНИЕ

С математической точки зрения задача рентгеновской компьютерной томографии заключается в нахождении функции по её интегралам на некотором семействе прямых. В [1] показано, что если носитель интегрируемой функции двух переменных $f(x,y)$ сосредоточен в полосе $-1 \le x \le 1$, то для восстановления этой функции достаточно знать интегралы только вдоль тех прямых, которые пересекают отрезок $[-1,1]$ на оси x. В работах [2, 3] построен численный алгоритм этой задачи. В алгоритме восстановления существенно используются двумерное прямое и обратное преобразования Меллина. Большие размерности сеток приводят к большим затратам машинного времени. В задачах подобного рода естественно использовать распараллеливание вычислений.

В [1] решена следующая задача.

Требуется найти функцию $\psi(\eta_1,\eta_2)$ по заданным от неё вдоль всех прямых, пересекающих отрезок [–1, 1] на оси $\eta_2 = 0$, если носитель функции $\psi(\eta_1,\eta_2)$ содержится в полосе $-1 \le \eta_1 \le 1, \quad -\infty < \eta_2 < \infty$ интегралам

$$u(x_1, p_1) = \int_{-\infty}^{\infty} \psi(x_1 + p_1 s, s) ds .$$

Таким образом, при восстановлении функции, сосредоточенной в полосе, можно обойтись, например, без интегралов вдоль прямых, параллельных оси $\eta_1 = 0$, а также без интегралов вдоль прямых, пересекающих эту ось за пределами отрезка [–1, 1]. Источник излучения может двигаться только вдоль этого отрезка и не обходить объект со всех сторон.

В данной работе описывается эксперимент, в котором в задачах томографии для параллельного вычисления преобразования Меллина использовался кластер Новосибирского государственного университета. На мастере и узлах кластера установлена операционная система Linux (Debian 2.2r3) с

ядром версии 2.4.9-SMP (symmetric multi-processing support). Загрузка узлов кластера осуществляется через сетевой интерфейс мастера. В качестве системы распределенного вычисления используется MPICH. Запуск задач на счет происходит через систему постановки задач SUN GridEngine. На мастере существует среда разработки и отладки программ под Fortran, в которой и осуществлялась работа. Счет выполнялся на 8 процессорах.

1. ВЫЧИСЛЕНИЕ ФУНКЦИИ $\psi(\eta_1,\eta_2)$ ПО ФУНКЦИИ $u(x_1,p_1)$

При построении численного алгоритма в [3] налагаются дополнительные условия, естественные для практических ситуаций. Предполагается, что носитель функции $\psi(\eta_1,\eta_2)$ содержится не просто в полосе $-1\le\eta_1\le 1$, а в прямоугольнике $-r\le\eta_1\le r,\quad h1<\eta_2<h2,\quad 0<r<1,\quad 0<h1$.

Шаг 1.

По функции $u(x_1,p_1)$ вычисляем функцию

$$g_\rho^{\pm}(\Gamma,t)=\Gamma^{\rho+1}[f(\Gamma,t)\pm f(-\Gamma,t)] \text{ при } \Gamma>0,\quad t>0,$$

где ρ – некоторое фиксированное число, удовлетворяющее условиям $0<\rho<1$, а функция $f(\Gamma,t)$ определяется равенством

$$f(\Gamma,t)=\frac{2t}{|\Gamma|(t^2+1)}u\left(\frac{(t^2-1)}{(t^2+1)},\frac{2t\Gamma}{(t^2+1)}\right).$$

Шаг 2.

Находим преобразование Меллина от функции $g_\rho^{\pm}(\Gamma,t)$

$$\tilde g_\rho^{\pm}(\mu,\nu)=\int_0^\infty\int_0^\infty g_\rho^{\pm}(\Gamma,t)\Gamma^{-1}t^{-1}\Gamma^{i\mu}t^{i\nu}d\Gamma dt,\qquad \mu\in R^1,\ \nu\in R^1.$$

Шаг 3.

По функции $\tilde g_\rho^{\pm}(\mu,\nu)$ находим функцию $\tilde v_\rho^{\pm}(\mu,\nu)$, используя равенство

$$\tilde v_\rho^{\pm}(\mu,\nu)=\frac{\tilde g^{\pm}(\mu,\nu)}{\lambda_\rho^{\pm}(\mu,\nu)},\ \text{где}$$

$$\lambda_\rho^{\pm}(\mu,\nu)=2^{-\rho+i\mu}\left(B(\rho-i\mu,\frac{1-\rho+i\mu-i\nu}{2})\pm B(\rho-i\mu,\frac{1-\rho-i\mu+i\nu}{2})\right),$$

B - Эйлеров интеграл первого рода.

Функция $\lambda_\rho^{\pm}(\mu,\nu)$ не зависит от исследуемого объекта и может быть вычислена заранее. Отметим также, что при малых значениях модуля $\lambda_\rho^{\pm}(\mu,\nu)$ необходимо использовать методы регуляризации.

Шаг 4.

Используя обратное преобразование Меллина, находим функцию $v_\rho^{\pm}(\gamma,\tau)$:

$$v_\rho^{\pm}(\gamma,\tau)=\int_0^\infty\int_0^\infty \tilde{v}_\rho^{\pm}(\mu,\nu)\gamma^{i\mu}\tau^{i\nu}\,d\mu d\nu\,.$$

Шаг 5.

По функции $v_\rho^{\pm}(\gamma,\tau)$ находим функцию $\psi(\eta_1,\eta_2)$, используя равенства:

$$\psi\left(\frac{\tau^2-1}{\tau^2+1},\frac{2\tau}{\gamma(\tau^2+1)}\right)=\frac{\gamma^{(1-\rho)}(\tau^2+1)^2}{8\tau^2}\left[v_\rho^{+}(\gamma,\tau)+v_\rho^{-}(\gamma,\tau)\right],$$

$$\psi\left(\frac{\tau^2-1}{\tau^2+1},\frac{-2\tau}{\gamma(\tau^2+1)}\right)=\frac{-\gamma^{(1-\rho)}(\tau^2+1)^2}{8\tau^2}\left[v_\rho^{+}(\gamma,\tau)-v_\rho^{-}(\gamma,\tau)\right].$$

Эти равенства позволяют вычислить значения искомой функции $\psi(\eta_1,\eta_2)$ для любых $\eta_1\in[-1,1]$, $\eta_2\in R^1$. При этом τ находится из равенства $\eta_1=\dfrac{\tau^2-1}{\tau^2+1}$, а γ из равенств $\eta_2=\dfrac{2\tau}{\gamma(\tau^2+1)}$ или $\eta_2=\dfrac{-2\tau}{\gamma(\tau^2+1)}$, в зависимости от знаков величин τ и η_2.

Описание алгоритма закончено.

2. ВОЗМОЖНЫЕ ПРИМЕНЕНИЯ АЛГОРИТМА

2D-томография

Если объект ограниченных размеров находится, например, в верхней полуплоскости, предложенный алгоритм позволяет восстановить плотность этого объекта при движении источника излучения вдоль некоторого отрезка на прямой, разделяющей полуплоскости.

При исследовании ряда объектов используется следующая схема сканирования. Для восстановления плотности слоя движение линейки детекторов вдоль некоторой направляющей сочетается с вращением объекта. При «односторонних» алгоритмах можно будет исключить вращение объекта.

Алгоритмы восстановления, использующие интегралы по прямым, пересекающим только одну из сторон объекта, могут быть использованы в классической двумерной томографии, однако основным стимулом для разработки таких алгоритмов является полная трехмерная томографическая реконструкция в конусе лучей при неполных данных.

3D-томография

Рассмотрим подробнее вопрос о применении изложенного алгоритма при томографической реконструкции в конусе лучей.

В классической компьютерной томографии используется двумерный веер лучей. Данные о прохождении через объект регистрируются с помощью одномерной линейки детекторов. Система «детекторы – источник излучения» дискретно вращается вокруг некоторой оси, оставаясь в плоскости, ортогональной этой оси. По полученным данным восстанавливается двумерная плотность среза, который предполагается тонким. После восстановления плотности среза система перемещается вдоль оси вращения и процесс повторяется. Схемы сканирования могут варьироваться, но существо дела не меняется: объект представляется в виде набора тонких срезов и на каждом шаге решается задача восстановления двумерной плотности среза, толщиной которого можно пренебречь.

В трехмерной томографии реконструкция ведется в трехмерных конусах лучей, и рассматривается другая схема получения и обработки данных. Источник движется по некоторой трехмерной траектории, для каждой позиции источника данные регистрируются на пленке или двумерной матрице детекторов. По системе полученных двумерных данных восстанавливается плотность трехмерного объекта в заданной точке. Наиболее распространенными траекториями являются система двух окружностей, лежащих во взаимно перпендикулярных плоскостях, и винтовая линия. Относительно винтовой линии (спирали) необходимо отметить следующее. В настоящее время выпускаются и продаются спиральные томографы. Как правило, фирмы не раскрывают деталей своих алгоритмов, однако по косвенным данным можно предположить, что там используется классическая схема. Эту схему можно использовать, если шаг спирали мал, приближенно считая

один виток спирали окружностью. Предварительные исследования показывают, что переход к трехмерной схеме восстановления может существенно повысить разрешение и понизить дозу облучения.

Наибольшее преимущество методы реконструкции в трехмерных конусах лучей перед двумерными (веерными) методами имеют в случае сложных пространственных форм исследуемых объектов и дефектов в них.

Для применения большинства формул обращения лучевого преобразования необходимо, чтобы исследуемый объект и траектория источника удовлетворяли условиям Кириллова–Смита–Туя. Главное из этих условий заключается в том, что любая плоскость, пересекающая объект, пересекает траекторию движения источника. Выше были приведены примеры кривых, удовлетворяющие условиям Кириллова–Туя (винтовая линия и совокупность двух единичных окружностей, лежащих во взаимно перпендикулярных плоскостях). Траектории, не удовлетворяющие условиям Кириллова-Туя, называются неполными. Как уже говорилось выше, для восстановления функций, имеющих финитный носитель (такие функции приходится исследовать в большинстве реальных ситуаций) не является необходимым выполнение условия Кириллова–Туя.

Так, в работе Ю. Е. Аниконова [4] показано, что для восстановления функций, сосредоточенных в шаре, достаточно одной окружности. В работе А. С. Благовещенского [1] показано, что одной окружности достаточно для восстановления интегрируемых функций, сосредоточенных в цилиндре. Это означает, что, используя траекторию из одной окружности, можно восстановить функцию, носитель которой не ограничен в одном из направлений. В реальных ситуациях и метод Ю. Е. Аниконова, и метод А. С. Благовещенского будут применяться к объектам конечных размеров, но у каждого из этих методов есть свои преимущества.

Используя свойства преобразования Фурье лучевых данных, можно вычислить интегралы вдоль прямых, проходящих через любую внутреннюю точку круга, зная интегралы вдоль прямых, проходящих через окружность. Начисляемые таким образом интегралы для внутренних точек круга мы будем называть виртуальными лучевыми (рентгеновскими) проекциями. Слово «виртуальный» здесь употребляется потому, что на самом деле источник излучения не находится во внутренних точках круга, а движется по окружности, являющейся его границей. Здесь уместно напомнить, что речь идет о трехмерном пространстве, а основной интерес представляют прямые, не лежащие в плоскости траектории источника.

Алгоритм построения виртуальных рентгеновских проекций изложен в [5,6], ряд свойств этого алгоритма исследован в [7].

Демонстрационные версии программ для построения виртуальных проекций находятся в разделе Томография сервера «Методы решения условно-корректных задач».

Адреса сервера

`http://www.iae.nsk.su/~trofimov/IPP/main.htm`
`http://a-server.math.nsc.ru/IPP/main.htm`
`http://cs.nstu.ru/ipp/`

В сочетании с методом виртуальных проекций алгоритм, изложенный выше, дает метод восстановления функции трех переменных по интегралам вдоль прямых, пересекающих единичную окружность, лежащую в плоскости $z = 0$.

Действительно, рассмотрим сечение объекта некоторой плоскостью, ортогональной плоскости $z = 0$. Если объект расположен внутри цилиндра единичного радиуса, то функция, полученная в сечении, по одной из переменных будет сосредоточена в полосе $[-1, 1]$. Интегралы по прямым, пересекающим отрезок $[-1, 1]$, можно найти, используя метод виртуальных проекций.

Ограничения, веденные нами при построении одностороннего алгоритма, в случае трехмерной томографии в конусе лучей сводятся к следующему:

– между объектом и цилиндром, в котором он находится, должен существовать «зазор», то есть объект должен находиться строго внутри цилиндра;

– объект должен быть отделен от плоскости $z = 0$, «висеть» над ней.

Если первое условие является естественным для реальных ситуаций, то второе требует пояснений. Мы предполагаем, что источник движется в плоскости $z = 0$. Второе условие означает следующее. При восстановлении функции в окрестности плоскости $z = 0$ нужно использовать методы классической двумерной томографии и (или) метод Фельдкампа. Метод виртуальных проекций тем точнее, чем больше зона, отделяющая объект от плоскости $z = 0$ [6]. Затем восстановленный слой нужно вычесть из объекта и построить новые проекции, соответствующие частям объекта, отделенным от плоскости $z = 0$.

Приведенная структура алгоритма позволяет для сокращения времени счета при вычислении преобразования Меллина использовать быстрое преобразование Фурье (БПФ). Следует отметить, что преимущества БПФ наиболее полно проявляются на равномерной сетке, что приводит к очень неравномерной сетке для искомых функций и к существенной потере точности.

3. ОПИСАНИЕ ЭКСПЕРИМЕНТА

В нашем эксперименте для вычисления преобразования Меллина использовался кластер Новосибирского государственного университета. Счет выполнялся на 8 процессорах. Все обрабатываемые матрицы делились по столбцам на равные 8 частей и каждая часть обрабатывалась на своем процессоре независимо от других.

1. Генерировались (на равномерной решетке) данные-плотности на круге радиуса 0.4 с центром в точке(0, 0.5), $\rho = 0.5$. Источник двигался по прямой. Значения Γ, τ менялись в интервале [0.01 + 9.2]. Отсчеты на решетке (Γ, τ) брались (256*256), (512*512), (1024*1024), (2048*2048), (4096*4096).

2. Затем вычислялось преобразование Меллина $\tilde{v}_\rho^{\pm}(\mu, \nu)$ для функции $v_\rho^{\pm}(\mu, \nu)$ на равномерной решетке по формуле:

$$\tilde{v}_\rho^{\pm}(\mu,\nu) = \int_0^\infty \int_0^\infty v_\rho^{\pm}(\Gamma, t)\Gamma^{-1} t^{-1} \Gamma^{i\mu} t^{i\nu} d\Gamma dt \qquad \mu \in R^1, \nu \in R^1.$$

3. После этого вычислялось обратное преобразование Меллина по формуле:

$$obrv_\rho^{\pm}(\Gamma,\tau) = \int_0^\infty \int_0^\infty \tilde{v}_\rho^{\pm}(\mu,\nu)\Gamma^{i\mu}\tau^{i\nu} d\mu d\nu$$

4. Затем сравнивалось $v_\rho^{\pm}(\Gamma, \tau)$, полученное на 1 шаге с $obrv_\rho^{\pm}(\Gamma, \tau)$, полученным на 3 шаге:

$$s = \int_{0.01}^{9.2} \int_{0.01}^{9.2} \frac{\left| v_\rho^{\pm}(\Gamma,\tau) - obrv_\rho^{\pm}(\Gamma,\tau) \right|}{\Gamma\tau} d\Gamma d\tau$$

Значения (μ, ν) менялись в интервалах: [–9.2 +9.2], [–18.4 +18.4], [–30 +30], [–36.8 +36.8], [–45 +45], [–73.6 +73.6].

Таблица оценок для разных интервалов

Границы интервала	s
9,20	0,254006
18,4	0,105358
30	0,107417
36,8	0,074831
45	0,148532
73,6	0,544064

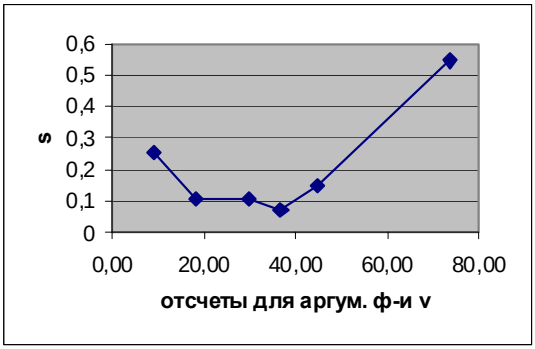


График оценок для разных интервалов

Самые хорошие оценки получились при значениях (μ,ν) на интервале [–36.8 +36.8]. Для этого интервала были получены оценки s при разных шагах дискретизации. Результаты приведены ниже.

Таблица оценок для интервала [–36.8 +36.8]

к-во отсчетов	s
256	0,074831
512	0,070392
1024	0,101108
2048	0,175747
4096	0,0107

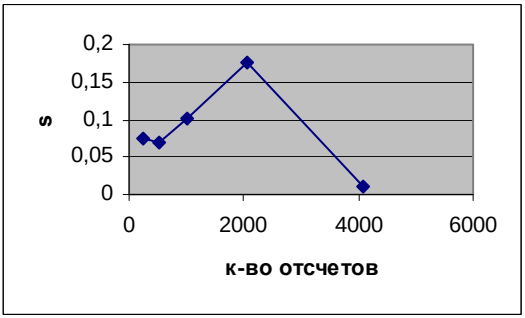


График оценок для интервала [–36.8 +36.8]

Выводы:

1. Распараллеливание алгоритма сократило время счета примерно в 8 раз, что позволило увеличить размеры обрабатываемых матриц до (4096*4096).

2. При уменьшении шага по (μ,ν) оценка

$$s = \int_{0.01}^{9.2}\int_{0.01}^{9.2} \frac{\left|v_\rho^{\pm}(\Gamma,\tau) - obrv_\rho^{\pm}(\Gamma,\tau)\right|}{\Gamma\tau} d\Gamma d\tau$$

вначале медленно растет, а затем резко убывает.

СПИСОК ЛИТЕРАТУРЫ

1. **Благовещенский А. С**. О восстановлении функции по известным интегралам от нее, взятым вдоль линейных многообразий // Математические заметки. – 1986. – Т. 39, № 6. – С. 841–849.
2. **Trofimov O. E.** Virtual beam (X-ray) projections // The VIIth International Conference on Fully 3D Reconstruction in Radiology and Nuclear, Medicine, France Saint-Malo, 2003, Abstracts. – P. 127.
3. **Трофимов О. Е.** Численный алгоритм томографии при движении источника по отрезку прямой // Автометрия. – 2005. – № 5. – С. 66–73
4. **Аниконов Ю. Е.** Некоторые методы исследования многомерных обратных задач для дифференциальных уравнений. – Новосибирск: Наука, 1978.
5. **Трофимов О. Е.** Алгоритм построения виртуальных рентгеновских проекций // Автометрия. – 2005. – № 3. – С. 64–69
6. **Trofimov O. E., Kasyanova S. N., Shaposhnikova E. V., Stukalin Yu. A., Zagoruyko A. S.** Noise stability of virtual beam (x-ray) projections // Proceedings of 4th World Congress Industrial Processes Tomography , Aizu, Japan, 5-8 September 2005. – P. 675–681.
7. **Trofimov O. E., Kasyanova S. N., Stukalin Yu. A., Zagoruyko A. S., Zhuravel F. A.** The relationship between a virtual x-ray projection quality and a distance of object to a plane in which a source is moving // 7th International Conference on Pattern Recognition and Image Analysis: New Information Technologies (PRIA-7-2004), 2004, St. Petersburg, Conference Proceedings. – P. 944–946.

Ю. В. Малинина

АВТОМАТИЧЕСКОЕ ВЫЯВЛЕНИЕ ТАКСОНОМИИ В ОБЛАСТИ ПРЕОБРАЗОВАНИЙ ПРОГРАММ НА ОСНОВЕ АНАЛИЗА СЕМАНТИЧЕСКИХ СВЯЗЕЙ В ПУБЛИКАЦИЯХ

ВВЕДЕНИЕ

В конечном счете научное знание воплощается в тексты и познается через тексты, поэтому можно допустить, что абстрактная модель научного знания является производной от множества реальных текстов и может рассматриваться как обобщающее представление этих текстов на уровне семантических моделей.

То есть задача построения классификации преобразований программ на данном уровне может быть интерпретирована как построение семантических сетей публикаций в области преобразования программ, содержащих информацию о научных терминах и их связях между собой с последующим обобщением их в единую семантическую сеть предметной области.

Следует заметить, что с 90-х годов множество проектов, посвященных автоматическому извлечению терминов, было выполнено, однако попрежнему остаются открытыми следующие вопросы [7]:

(1) идентификация сложных терминов, особенно вопрос начала и окончания терминологической фразы произвольной длины;
(2) распознавание сложных терминов, то есть решение составляет набор слов;
(3) соответствие терминологической единицы словарю предметной области.

На основе сравнения существующих систем авторы [7] заключают, что для того, чтобы улучшить извлечение терминов и сделать результат более релевантным, то есть уменьшить информационный шум и потери, следующие условия должны быть выполнены.

Во-первых, должны быть проведены лингвистически ориентированные исследования семантических связей терминов и условий ограничения терминологических единиц в пределах данной специальной области и в данном текстовом типе.

Во-вторых, программные системы должны научиться сочетать статистические и лингвистические методы и поддерживать более одной стратегии. Также должна быть полезной разработка общей шкалы тестирования и оценки/сравнения качества извлекаемых терминов.

1. ТИПЫ СЕМАНТИЧЕСКИХ СВЯЗЕЙ В ТЕКСТЕ

Сжатие информации при переходе от лексического к семантическому описанию документов приводит к ее обобщению, что эквивалентно получению некоторого знания. Ведь возможность более сжатого описания данных есть следствие скрытых в этих данных закономерностей. Сжатие информации по сути и сводится к выявлению этих закономерностей, выражающих наши знания о структуре данных.

Семантические сети с самого начала активно использовались для построения систем обработки естественного языка. В семантическую сеть включаются наиболее часто встречающиеся слова текста, которые несут основную смысловую нагрузку. Для каждого понятия формируется набор ассоциативных (смысловых) связей, т.е. список других понятий, в сочетании с которыми оно встречалось в предложениях текста. При этом считается, что чем чаще встречаются вместе два термина в предложениях текста, тем выше вероятность того, что они связаны по смыслу.

При этом нельзя сбрасывать со счета особенности стиля научного текста. Общеизвестно, что для научного стиля характерно использование следующих языковых средств:

- на уровне лексики:
  - насыщенность терминами данной предметной области;
  - использование слов с абстрактным значением: закон, число, предел, свойство; отглагольных существительных со значением действия: переработка, приземление, использование;
  - употребление слов в прямых значениях, отсутствие образности (метафор, междометий, восклицательных частиц);
  - частое использование лексических средств, указывающих на связь и последовательность мыслей: сначала, прежде всего, во-первых, следовательно, наоборот, потому что, поэтому;
- на уровне морфологии:
  - редкое использование личных местоимений я и ты и глаголов в форме 1 и 2 лица единственного числа;

- специальные приемы авторизации: авторское «мы», неопределенно-личные (считают, что...) и безличные конструкции (известно, что...; представляется необходимым...);
- использование причастий, деепричастий и оборотов с ними;

- на синтаксическом уровне:
  - употребление сложных предложений с использованием союзов, указывающих на связь явлений;
  - неупотребление восклицательных предложений, незначительное употребление вопросительных предложений;
  - частые цитаты, ссылки;
  - использование в качестве компонентов текста формул, графиков, схем.

Перечисленные выше языковые средства в свою очередь могут быть использованы как индикаторы семантических отношений.

Рассмотрим несколько основных семантических отношений, которые можно выделить как ключевые для построения классификации терминов предметной области.

1.1. Вариантность и синонимия

К синонимии относятся отношения, основанные на полном или частичном совпадении значений. Слова, связываемые отношением синонимии, называются синонимами [8].

Существует по крайней мере 2 разных подхода к определению синонимии:

- семантическая близость слов: в данном случае в качестве критерия рассматривается наличие у слов некоего общего значения [11].
- взаимозаменяемость в контексте: два слова считаются синонимами, если существует высказывание (или ряд высказываний), в котором замена этих слов друг на друга не влияет на истинность высказывания ([12, 13, 14]).

Таким образом синонимия − это отношения тождественных (или близких) семем, формально выраженные разными лексемами, например: Elimination / Removal.

Elimination − act of getting rid of; omission, act of leaving out; process of solving simultaneous equations by removing the variables (Algebra).

1) удаление; исключение; выбрасывание; 2) отсев; выбывание; 3) устранение; уничтожение, ликвидация; 4) физиол. очищение; выделение; экскреция, удаление из организма; 5) мат. исключение (неизвестного).

Removal − act of taking off, act of shedding; act of taking away; elimination; ejection, dismissal.

1) перемещение; переезд; вывоз; 2) смещение (с должности); 3) устранение, удаление; ликвидация; исключение; 4) горн. выемка перемещение.

Синонимия тесно связана с проблемой распознавания употребленных в научных текстах терминов и понятий.

Согласно [4] в терминоведении до некоторого времени преобладал подход, согласно которому в специальных текстах одно понятие должно иметь только один способ выражения. Соответственно, синонимия и полисемия терминов признавались нежелательными явлениями, и научно-технические терминологии подвергались логической и лингвистической регламентации, направленной на устранение этих явлений. Однако стремительный рост числа новых специальных терминов, сопровождавший развитие науки и техники, и неизбежное функционирование в речи большого количества синонимичных вариантов, а также исторические особенности употребления терминов различными научными школами и сообществами требуют учета вариативности специальных терминов.

Особенно хорошо это заметно в области преобразования программ.

Например, в одних работах используется термин dead code elimination (устранение неиспользуемого кода) [15, 16,17,18], а в других dead code removal (удаление неиспользуемого кода) [20, 21, 22].

1.2. Гипонимия

В лингвистической литературе принято слова, обозначающие родовые понятия, называть гиперонимами, а слова, обозначающие видовые понятия – гипонимами [8].

Гипонимия — это отношения родовидового включения, формально выраженные разными лексемами: *дом/строение*. Гипероним обозначает общее родовое понятие или совокупность, целое по отношению к составляющим его элементам, частям. Гипоним обозначает видовое понятие или название элемента, части какого-нибудь множества, целого.

Гиперонимы и гипонимы образуют гипонимические ряды, в которых гипонимы занимают подчиненное положение по отношению к гиперонимам. В гипонимический ряд входят один гипероним, занимающий ведущее место и обозначающий общее понятие, и один или несколько гипонимов, занимающие подчиненное положение по отношению к нему [8].

Например, согласно [10] мы можем выделить следующие гипонимические связи между терминами в области преобразования программ.

- Partial Evaluation (частичные вычисления) включает:
  - Constant Propagation (втягивание констант);
  - Constant Folding (свертка констант);
  - Copy Propagation (втягивание копий выражений);
  - Statement Substitution (подстановка выражений);
  - Reassociation (перегруппировка);
  - Algebraic Simplification (алгеброическое упрощение);
  - Function Cloning (клонирование функции);
  - I/O Format Compilation (компиляция формата ввода-вывода).

- Redundancy Elimination (устранение избыточности) включает:
  - Unreachable Code Elimination (устранение недостижимого кода);
  - Useless Code Elimination (устранение бесполезного кода);
  - Dead Variable Elimination (устранение мертвых переменных);
  - Common Subexpression Elimination (устранение общих подвыражений).

2. ПОДХОД К РЕАЛИЗАЦИИ

2.1. Предварительная обработка

Как уже предлагалось в [1], текст будем рассматривать, как множество соответствующих основ слов, входящих в него. Последовательность обработки текста будет следующая:

1. Построение неупорядоченного списка слов текста с учетом пропуска так называемых стоп-слов.
2. Выделение списка основ слов (stemming).

3. Подсчет частоты вхождения и частоты появления рядом слов текста.
4. Составление списков близких слов, на основе частоты появления рядом.
5. Определение меры близости для любых двух слов.

2.2. Извлечение терминов

Для автоматического извлечения многословных терминов предполагается использовать хорошо известный подход основанный на N-gram модели, которая является разновидностью вероятностной модели для определения следующего члена последовательности.

N-gram размера 1 называется «unigram»; размера 2 − «bigram» (или, менее обычно, «биграмма»); размера 3 является «триграммой»; и размера 4 или более просто «n-gram».

Согласно [5], алгоритм прямого подсчета количества пар (freq), который является модификацией метода автоматического выделения двусловий на основе bigrams, может быть использован для составления списка терминов-кандидатов в задачах полуавтоматического формирования терминологических ресурсов. Он основан на простейшем методе упорядочиния двусловий по убыванию их встречаемости в тексте

Например, для фрагмента *Strength reduction is a compiler optimization where a costly operation is replaced with an equivalent, but less expensive operation. Operator strength reduction involves using mathematical identities to replace slow math operations with faster operations,* получаем, что только словосочетание «strength reduction» встречается больше одного раза и может служить кандидатом термина.

И далее для терминов любой длинны указанный метод модифицируется наращиванием словосочетаний, если более короткие словосочетания достаточно часто встречаются в составе более длинных.

Для повышение качества алгоритмов на множестве текстов выбранной узкой специальности за счет удаления устойчивых выражений общей лексики желательно использовать дополнительный «контрастный» корпус текстов более широких областей науки. Следует также отметить, что этот способ выделения терминов эффективен только при использовании достаточно большого корпуса общенаучных текстов [6].

2.3. Выявление синонимов и вариантов терминов

Стратегия распознавания синонимов (вариантов) терминов в общем случае опирается на разбиение всего набора терминов и кандидатов в термины, полученных на предыдущем шаге, на группы синонимичных вариантов (одна группа соответствует одному понятию) на основе похожести терминов с учетом словарей синонимов общей лексики и словарей сокращений

Например,

- синонимия ключевого слова в словосочетании Dead code elimination, Dead code removal может быть определена на основе словаря;
- использование сокращений Input/Output Format Compilation, I/O Format Compilation.

2.4. Выявление гипонимов/гиперонимов

Шаблонный метод извлечения гипонимов и гиперонимов (для английского языка) описан в [9], он основан на извлечении гипонимов с помощью заранее заданных лексико-синтаксические шаблонов. Фактически утверждается что для всех семантических конструкций выделенных по шаблону

NP0 <шаблон> {NP1, NP2, …, (and|or)} NPn.

Существует связь

for all Npi, 1<= i <=n, гипоним (Npi, NP0),

где <шаблон> = “such that” (такой как), “or other” (или др.), including (включая), especially (особенно)

Например

Although related to caching, memorization refers to a specific case of this optimization, distinguishing it from forms of caching such as buffering or page replacement.

Гипоним (buffering, forms of caching); Гипоним (page replacement, forms of caching)

ЗАКЛЮЧЕНИЕ

Описанные подходы могут быть полезны для проведения терминологического анализа текста в системах литературно-научного редактирования, а также для автоматического реферирования и аннотирования документов в информационно-поисковых системах

Автоматическое извлечение гипонимических рядов терминов из текста с учетом синонимии открывает дополнительные возможности построением автоматической классификаций предметной области.

СПИСОК ЛИТЕРАТУРЫ

1. Малинина Ю.В. Семантическая сеть как формальный метод описания и обработки текстов по преобразованиям программ // Методы и инструменты конструирования и оптимизации программ. – Новосибирск, 2005. – С. 137–144.
2. Большакова Е.И., Васильева Н.Э., Морозов С.С. Лексико-синтаксические шаблоны для автоматического анализа научно-технических текстов // Тр. 10-й национальной конф. по искусственному интеллекту с международным участием – КИИ'2006. – М.: Физматлит, 2006. – Т. 3. – С. 832–840.
3. Лавренова О.А. Моделирование семантической структуры текстов научно-технического содержания в связи с автоматизацией информационных процессов. Дис. канд. филол. наук. – М.: Московский гос. университет, 1978. – 280 с.
4. Большакова Е.И., Васильева Н.Э.. Терминологическая вариантность и ее учет при автоматической обработке текстов // Тр. XI Национальной конф. по ИИ с международным участием КИИ-2008, г. Дубна. 29 сентября–3 октября 2008 г. – http://www.raai.org/cai-08/files/cai-08_paper_208.doc
5. Браславский П.И., Соколов Е.А. Сравнение пяти методов извлечения терминов произвольной длины // Тр. конф. ДИАЛОГ, 2008. – С. 67–74.
6. Митрофанова О.А., Мухин А.С., Паничева П.В. Автоматическая классификация лексики в русскоязычных текстах // Тр. междунар. конф. «Диалог'2007» (Бекасово, 30 мая–3 июня 2007 г.). – С. 413–422.
7. Castellvi M., Bagot R., Palatresi J. Automatic term detection: A review of current systems // Recent Advances in Computational Terminology. – Amsterdam: John Benjamins, 2001. – P. 53–87.
8. Новиков Л.А. Семантика русского языка. – М.: Высшая школа, 1982
9. Hearst M. Automatic Acquisition of Hyponyms from Large Text Corpora // Proc. of COLING 92, Nantes. – 1992. – Vol. 2. – P. 539–545.
10. Bacon D.F., Graham S.L., Sharp O.J. Compiler transformations for high-performance computing // ACM Computing Surveys. – 1994. – Vol. 26, № 4. – P. 345–420.

11. Словарь синонимов русского языка: В 2 т. / АН СССР, Институт русского языка; Под ред. А. П. Евгеньевой. Л.. – Наука, Ленинградское отделение, 1970.
12. Lyons J. Semantics. (2 vol.) – London; New York, 1977.
13. Miller G. et al. Five Papers on WordNet. – Princeton University, 1990. – (CSL-Report / Vol. 43).
14. Новый объяснительный словарь синонимов русского языка. / Под рук. Ю. Д. Апресяна. – М: «Языки русской культуры». Вып. 1, 1997.
15. Knoop J., Ruthing J., Steffen B. Partial dead code elimination // ACM SIGPLAN Notices. – 1994. – Vol. 29, N 6. – P. 147–158.
16. Hongwei Xi. Dead code elimination through dependent types // The First Internat. Workshop on Practical Aspects of Declarative Languages (PADL '99). – Lect. Notes Comput. Sci. – 1999. – Vol. 1551. – P. 228–242.
17. Gupta R. Partial dead code elimination using slicing transformations // Proc. of the ACM SIGPLAN '97 Conf. on Programming Language Design and Implementation (PLDI). – SIGPLAN Notices. – 1997. – Vol. 32(5). – P. 159–170.
18. Ancourt C. etc. How to add a new phase in PIPS: The case of dead code elimination // Sixth Workshop «Compilers for Parallel Computers» (CPC'96), Aachen, Konferenzen des Forschungszentrums Jülich. – P. 19–30.
19. Gold N., Harman M. An empirical study of static program slice size // ACM Transactions on Software Engineering and Methodology (TOSEM). – Vol. 16 , Iss. 2. – http://www.cs.loyola.edu/~binkley/papers/tosem-slice-size.ps
20. Chang P., Scott A. and etc. Using pro le information to assist classic code optimizations // Software Practice and Experience. – 1991. – Vol. 21(12). – P. 1301–1321.
21. Madou M., Van Put L., De Bosschere K.. Understanding Obfuscated Cod e// Proc. of the 14th IEEE Internat. Conf. on Program Comprehension (ICPC). – IEEE Computer Society, 2006. – P. 268–274.

Л.С. Мельников[1]

О СЕМИНАРЕ ЗЫКОВА В НОВОСИБИРСКЕ

Цель настоящих заметок – дать хотя бы мимолетный взгляд на семинар Зыкова с точки зрения студента Новосибирского государственного университета (НГУ), позднее стажера-исследователя Института математики СО АН (ИМ). Личные впечатления автора о семинаре относятся к периоду с 1964 по 1969. Таким образом, это взгляд из того времени.

А. А. Зыков, 1959-1960

Первое известное упоминание о семинаре Зыкова можно встретить в послесловии к русскому переводу книги Клода Бержа «Теория графов и ее приложения» [1]. Там среди участников семинара по теории графов в Институте математики СО АН названы Л. В. Канторович (будущий Нобелевский лауреат), А. И. Фет, В. Г. Визинг (будущий лауреат Большой серебряной медали ИМ), А. А. Берс, М. И. Кратко и Н. М. Сычева. По изустным сведениям к первоначальному составу семинара примыкали Г. С. Плесневич, Л. М. Витавер, Ю. М. Волошин, Б. А. Трахтенброт, А. П. Ершов (будущий академик) и Г. И. Кожухин. Первые годы существования семинара Зыкова отмечено участием Г. А. Бекишева, В. В. Матюшкова, Ю. Г. Решетняка (будущий академик) и В. А. Тюренкова. Примерное время возникновения этого семинара следует отнести к 1960–1961 годам. Вероятно,

[1] Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (коды проектов 08-01-00516 и 08-01-00673)

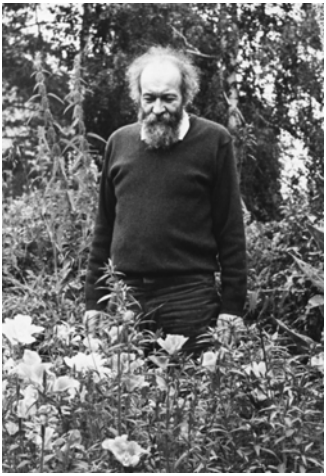
первоначально он возник при изучении книги К. Бержа[2] и носил в большей мере реферативный характер.

Основные качества моего характера с самого детства – застенчивость и любовь к наукам. С них то все и началось. Правда, теперь в мою застенчивость немногие верят.

Мои первые научные контакты начались в 1963 г. с научно-реферативного семинара, руководителем которого был Алексей Андреевич Ляпунов. Целью его было изучение относительно свежих статей по математической логике. Должен заметить, что Ляпунов в этот момент читал нам свой курс математического анализа, в котором он значительно отходил от принятых на то время канонов. Сей логический семинар проходил у Ляпунова дома – в коттедже по нынешней Терешковой, 3. Семинар был довольно серьезного уровня, – лишь замечу, что среди участников была и тогдашняя аспирантка Л. Л. Максимова. Ляпунов был человеком умным, ярким и широким; мы обсуждали и другие темы от кибернетики до изобразительного искусства, причем речь шла совсем не о «продажной девке» и «соц. реализме», чем полны были газеты. Бывали и пирожки, выносимые после завершения заседания для поддержания «голодных» студентов.

У меня твердо запечатлелся в памяти внешний вид объявления о семинаре Зыкова, висевшего на доске объявлений мехмата, где на листке из ученической тетради в клетку круглым почерком Зыкова перьевой ручкой и фиолетовыми чернилами было начертано «Семинар «Теория графов» …», причем по названию Зыков проходился повторно красным карандашом.

Думаю, что редкий студент приходит на научный семинар в первый раз один. Он обязательно старается привлечь хотя бы одного товарища, а то и группу студентов. И вот сидит такая группа сзади всех, затаится и слушает «деятелей науки». Вот точно так в на-



А. А. Ляпунов в саду 1970-1972

[2] Автор имел счастье интервьюировать К. Бержа у него дома в 1990 году. На вопрос о вышеупомянутой книге профессор Берж продемонстрировал экземпляр другой своей первой книги, также посвященной теории графов, но сказал, что она носила более инженерно-оптимизационный характер, а вот его вторая книга, освещавшая теорию графов как математическую дисциплину, получила значительно более широкое распространение.

чале весеннего семестра 1964 года в составе такой вот группки студентов и я появился на этом семинаре. Еще до этого я слышал от одного из моих «боевитых» сокурсников об этом семинаре и его руководителе.

Еще школьная любовь моя к комбинаторике толкала меня на этот семинар, а застенчивость и «мнимые» обязательства перед логикой и Ляпуновым тянули назад. Когда я скромно сказал Алексею Андреевичу об этих сомнениях, в ответ я получил добро, и такие слова как «управление» и «автоматы» были произнесены для обоснования целесообразности этого занятия. И хотя сам разговор был весьма кратким, но сама борьба этих двух противоположностей продолжалась почти год.

Из основных участников семинара я застал в то время Зыкова, Визинга, Плесневича, Зарецкого, Матюшкова и Лейфмана. Изредка заглядывали Берс, Кратко, Коршунов, Бекишев, Петрова и Трахтенброт. Буквально считанное число появлений на семинаре я помню у Ершова и Кожухина, но заглядывали они скорее для решения каких-то своих дел с одним или несколькими участниками, да и понятно – все их время уходило тогда на разного рода трансляторы. Причем Ершов запомнился более ярко – одновременно мягкий и энергичный, обаятельный и решительный. Семинар в то время не имел определенного места и времени и проходил то в Институте математики (чаще в комнате 417), то в разных аудиториях Новосибирского Университета. Место и время менялись каждый семестр, а часто, бывало, и в середине семестра. Это объяснялось тем, что многие участники семинара преподавали, а потому выкраивали время и место в соответствии со своим расписанием. Довольно часто семинар проходил и в вечернее время, начало в 18 или 19 часов. К моменту моего появления я был немного знаком с графами по книге К. Бержа, на которую обратил мое внимание в октябре-ноябре 1962 года наш преподаватель математической логики Алиев, также принимавший иногда участие в семинаре.

А. П. Ершов, апрель 1986

С. Л. Соболев, 1970 годы

Одно из заседаний в Институте математики особо запечатлелось в памяти, оно состоялось в огромном кабинете директора Института Сергея Львовича Соболева, куда нас проводила постоянный секретарь директора Л. М. Крапчан, так как 417 комнату она отдала для какой-то защиты, а нам оказалось негде приткнуться. Меня, как самого юного, усадили в директорское кресло, и чувствовал я себя на том месте весьма неловко. Конечно, не по Сеньке шапка, и тут я впервые задумался о тяжести «шапки Мономаха».

Жили мы тогда необычайно весело и с огромной пользой для науки. Нужно отметить, что с осени 1964 года большинство студентов III курса мехмата получили хоть и крохотные 7-метровые, но зато отдельные комнатки в общежитии на Пирогова, 6. Этот факт серьезно способствовал интенсификации научных исследований во всех областях математики, которыми наши студенты занимались. Параллельно с обучением в НГУ я начал свое художественное образование в Детской художественной школе СО АН. Она была создана группой молодых художников (живописцев, скульпторов, графиков и прикладников) из Новосибирска, которых Академгородок притягивал в то время своими многочисленными возможностями. Многое из того, что было категорически нельзя в городе, в Академгородке было возможно. При этой школе был создан своего рода вечерний художественный клуб, где студенты, молодые и не очень молодые сотрудники Институтов СО АН и даже рабочие и техники имели шанс чему-то научиться, многое узнать и даже позволить себе роскошь эпистолярного общения с такими мастерами, как Шагал. Позднее, общаясь со студентами разных художественных училищ, институтов и академий, я понял, какое огромное счастье нам выпало – не было рутины и надзора со стороны художественной и партийной бюрократии. «Факел» достаточно обеспечивал финансово наших педагогов и нас художественными материалами, плата за обучение хотя и существовала, но была чисто символической – особенно для студентов. Наши преподаватели не только учили нас практическим навыкам, но и допускали в свои мастерские, и мы могли видеть творческий процесс. Несмотря на постоянную нехватку времени, мне в то время удавалось прилично учиться в университете, заниматься теорией графов и художествами. Кстати, в общении с Ляпуновым, кроме математики, кибернетики и геоло-

гических коллекций появилась еще одна тема – «художники и живопись». Не только два этюда Константина Коровина и два очень хороших натюрморта Игоря Грабаря, висевших у него в гостиной, были объектами наших дискуссий. Через ГПНТБ становилось возможным знакомство с журналами «Мир искусства» и «Золотое руно». Постоянные выставки в ДК «Академия» и Доме ученых – вся атмосфера способствовала росту и развитию. Удобство того времени состояло не только в том, что можно было постучаться в практически любую дверь и найти там приятную компанию, озабоченную судьбами науки, искусств или поэзии и литературы. Одним словом – «страна непуганых идиотов», как назвал Академгородок один из партийных чиновников, навещавший нас с проверкой. На эту тему см. [2].

Да простит меня читатель за отклонение от темы. Отчетливо я помню появление на семинаре харьковчанина М. К. Гольдберга. Зыков вышел к доске и представил его как своего возможного аспиранта. Видимо, это было после длинного собеседования с Марком. Предстояло лишь пройти ряд формальностей: сдачу экзаменов и оформление документов. Не помню, кто тогда докладывал, но Марк скоро выскочил к доске со своими «оптимистическими» заявлениями. Визинг парировал какое-то из них, и у доски поднялась такая карусель, что в конце семинара оба они, донельзя измазанные мелом, вышли, если не друзьями ещё, то коллегами. Докладчик так и не пришел в себя и до конца семинара жался в стенку 417 аудитории.

Безусловным лидером по качеству сделанных докладов был в то время Вадим Визинг. Однажды ему понадобилось 15 минут на то, чтобы опровергнуть оригинальный «результат» докладчика. Я до сих пор ношу в себе то чувство восторга, охватившее меня, когда он рассказывал свои структурные теоремы о критических графах с повышенным хроматическим классом – настолько эти результаты были неожиданны. Он умело доносил до слушателей основную идею и блестяще подчеркивал неожиданные повороты доказательства, существенно используя при этом свойственную только ему ожесточенную жестикуляцию. Я так и вижу вершину у степени k (конец указательного пальца его свободной левой руки), вершину x, смежную с ней (запястье правой руки) и $\sigma - k + 1$ вершин степени σ (концы других пальцев той же руки). Иногда он при-

сутствовал на семинаре, но вовсе не слушал, а что-то прикидывал, исчеркивал целые тетради какими-то абсолютно несвязными для постороннего закорючками – это означало, что, быть может, скоро мы будем слушать его новый доклад.

Александр Александрович Зыков, будучи в роли руководителя семинара, был вынужден выслушивать и понимать всех докладчиков. Теперь, имея за плечами почти 40-летний опыт подобного руководства, я понимаю, как это иногда нелегко. Наш руководитель был сверхдоступным человеком и умел очаровывать. Поэтому на семинаре бывали очень разные доклады. Выступали алгебраисты и уходили, как правило, непонятыми. Приходили производственники-инженеры, физики, пытавшиеся поставить задачи, казавшиеся им графскими (по-моему, ни разу задачи таковыми не оказывались).

Совершенно естественно, что Зыков стал моим научным руководителем в университете, но эту обязанность фактически исполнял Визинг. На семинаре часто выступали иногородние люди, приезжавшие к Зыкову на консультацию, за советом. Среди них я помню Я. Я. Дамбита, Э. Г. Давыдова, И. Г. Илзиню, В. К. Кабулова, В. Г. Карпова, А. О. Кац, В. П. Козырева, В. К. Кузнецова, С. Э. Маркосяна, А. Мелихова, Б. С. Мордвинова (прошу прощения у тех, кого забыл). Года через два после появления Гольдберга появился целый отряд аспирантов-стажеров из Ташкента, Бухары, Самарканда, Баку. Со многими из них у нас завязались научные и дружеские связи.

С сильно-регулярными графами мне посчастливилось впервые познакомиться на выступлении американки Эстер Зейдель, приезжавшей по приглашению Зыкова (кстати, это был первый мой опыт доклада на иностранном языке). Иногда и сам Зыков рассказывал о свежих препринтах, присланных ему из-за рубежа. В те времена мы, т.е. группа Зыкова, старались охватить всю новую информацию по теории графов, о которой мы узнавали благодаря реферативным журналам, а потом с помощью отлично работавшей библиотеки ИМ. И нам это удавалось. Весьма полезной была для нас и картотека (всего несколько старых каталожных ящиков) по теории графов, начатая Зыковым.

Мое приобщение к работе на семинаре произошло в начале следующего осеннего семестра, когда Зыков решил, что настала пора испытать меня, и дал прореферировать какую-то незначительную работу из чешского журнала «Часопис про пестовани математику». Содержание работы я сейчас абсолютно не помню, да это, видимо, и не важно. Усидчивости у меня хватало, и я честно перевел всю статью, как мне казалось тогда – разобрался во всем, настолько, что мог бы ответить на любой вопрос по статье.

Пришел минут за двадцать до начала семинара, причем надел новые ботинки – они слегка жали, но я пока этого не замечал, так как сердце по дороге в институт ёкало, в груди что-то сжималось, а во рту было так сухо, что с трудом отрывал язык, присыхавший к нижним зубам. Но вот появляются и первые участники семинара. Ким Айзекович Зарецкий участливо спрашивает меня о теме моего выступления. Я что-то бормочу. Но вот Зыков объявляет мой доклад, называя меня полным именем.

Я негнущимися деревянными ногами выхожу к доске (ботинки жмут и страх сжимает горло). Чуть не пальцем отрываю присохший язык и начинаю муссировать опять название, автора, журнал и выходные данные статьи, постепенно переходя от форте к пианиссимо. Кричат – «Громче…». Я стою лицом к аудитории, но почти ничего не вижу. Только общая расплывчатая картина у меня перед глазами. Чувствую, что так нельзя, хватаю мел и ковыляю к доске. Начинаю писать малозначащие обозначения, бубня в доску первые заученные фразы моего доклада и опять переходя от форте к пианиссимо. Мне говорят: «Громче…». Напрягаюсь. Я весь мокрый и уже видимо красный, капли пота срываются с носа. Понимаю, что надо повернуться к аудитории (моей расплывчатой картине), разворачиваю левую ногу и с трудом приставляю к ней правую. Говорю свою речь дальше (мне кажется, что я почти кричу), но тут я понимаю, что каждое слово в ней отделено от других перегородкой моего непонимания. Смысл фразы теряется, ускользает прежде всего от меня, и поэтому не только мои ноги стиснуты ботинками, но и я сам привязан к бумажкам в своей руке. Понятно, никакая сила не сможет до конца выступления вырвать мои бумажки – я их держу крепко. Более того, я кидаюсь к столу и выхватываю у Зыкова журнал с реферируемой статьей.

Обратим внимание на мою жестикуляцию в это самое время. Левая моя рука прижимает к груди драгоценные бумаги вместе с журналом, а правая с мелом молотит воздух. И я думаю, что у некоторых участников, видевших это выступление, могло возникнуть подозрение, что я брошу в них мелом. Но напоминаю, я видел перед собой только общую расплывчатую картину. Я резко дергал шеей в сторону того, кто мне пытался задать вопрос или довести меня до понимания тех «правильных» слов, которые вырывались из моих уст, так как был удивлен ответной реакцией расплывчатой картины.

В перерыве я ещё и ещё просматривал статью. Второй час, как мне кажется теперь, я говорил лучше и более осмысленно. Но прием перехода от форте к пианиссимо повторялся неоднократно. Когда после семинара в коридоре я осмелился спросить Зыкова его впечатления о моем выступлении с оговоркой, что выступал я не очень хорошо, он подумал и сказал, забирая у меня журнал: «Ничего – для первого раза!»

Думаю, что читатель согласится с этим мнением.

К. Мосесян, Л. С. Мельников, А. А. Зыков, В. Г. Визинг, М. К. Гольдберг, 8 сентября 1972 г, Одесса

Я вспомнил этот эпизод с тем, чтобы ободрить моих теперешних и бывших студентов в их первых шагах на любом поприще. За эти воспоминания я взялся потому, что считаю наш семинар, который сегодня собирается в двухсотый раз, преемником семинара А. А. Зыкова в Академгородке. Надеюсь, что наш семинар «Экстремальные задачи на графах» достойно продолжает дело, начатое и поддержаное А. А. Зыковым, В. Г. Визингом, Г. С. Плесневичем, М. К. Гольдбергом, А. А. Ляпуновым, А. П. Ершовым, Л. М. Кожухиным и многими, многими другими. Приношу свои извинения тем людям, которых я здесь не упомянул. Безусловно, семинар А. А. Зыкова в Академгородке был первым и определяющим этапом для развития теории графов в нынешних странах СНГ, а может быть частично и в странах восточного блока.

Рассматривая семинар Зыкова в Новосибирске как первый этап ее развития на сибирской почве, должен заметить, что этот этап был достойно завершен публикацией обзорной статьи Визинга, посвященной нерешенным проблемам теории графов [5], а также первой монографии Зыкова «Теория конечных графов I» [6] под редакцией Визинга. И если Визинг покинул нас в октябре 1968 года ради позиции заведующего кафедрой математики в Нежинском педагогическом институте – alma mater Н. В. Гоголя, то Зыков уехал чуть позже и, встретив Новый 1969 год в Киеве вместе с отцом и подъехавшим в Киев семейством, через несколько недель начал новый семестр в стенах Одесского института инженеров морского флота,

замкнув цикл Одесса–Москва–Новосибирск–Одесса для воспитания будущих юмористов, комедиографов и драматургов.

23 декабря1975……... 6 июня 2008

P.S. Этот текст впервые был написан к левой дате и был опубликован в единственном экземпляре в [3] к 200 заседанию новосибирского семинара «Экстремальные задачи на графах». Некоторые добавления и новая редакция относится к правой дате; для этого понадобилось почти 33 года. Рисунки-шаржи принадлежат руке В. А. Аксенова, а фотографии разных лет взяты из архива автора. Желающие выступить на графскую тему обращайтесь заранее к автору статьи по адресу <omeln@math.nsc.ru>. В момент написания этого текста уже запланированы выступления на 1238–1240 заседания семинара «Теория графов».

И в самом конце два коротких стиха Во-Кинь-Лем'*а* [4], также имеющих отношение к теории графов:

* * *

Печально – печально
холодное утро.
Шумит и шумит
нескончаемый ветер…

Вперед понеслася
теория графов.
И где-то качает
её и бросает.

Да будут удачи
в делах Ваших Графы.
В начале пути
о конце позаботьтесь.

Воспользуйтесь всеми
удобствами чина,
Но чином не бейте
пред каждою дверью.

* * *

коллеге Н посвящается

Эти солнце с луной…
День помчится за днем.
Незаметно уйдет
ученичество вслед.

Счастье к нам никогда
не приходит само,
А несчастья зато
не дают себя ждать.

Так пораньше вставай
и попозже ложись.
Вот талантов тебе
я желаю – каких.

Если ж доля твоя
бесталанным прожить,
И тогда я стерплю.
Что поделаешь, Н?

СПИСОК ЛИТЕРАТУРЫ

1. Берж К. Теория графов и ее применения, пер. с франц. А.А. Зыкова. – Москва: ИЛ, 1962. – С. 320.
2. Berge C. Theorie Des Graphes et Ses Applications. – Collection Universitaire de Mathematiques, Dunod, Paris, 1958.
3. «И забыть по-прежнему нельзя…»**:** Сборник воспоминаний старожилов Академгородка. – Новосибирск: Изд-во СО РАН, 2007. – С. 336.
4. Мельников Л. С. О семинаре Зыкова // Вечерние экстремальные задачи на графах. – 1975. – Вып. 1. – С. 4–9.
5. Во-Кинь-Лем. Сдирания из Тао-Юань-Миня // Вечерние экстремальные задачи на графах. – 1975. – Вып. 1. – С. 13–14.
6. Визинг В. Г.. Некоторые нерешенные задачи в теории графов // УМН. – 1968. – Т. 23, Вып. 6(144). – С. 117–134.
7. Зыков А. А., Теория конечных графов **I,** Тр. ИМ СО АН СССР. – Новосибирск: Наука, 1969. – С. 544.

Ф. А. Мурзин, С. А. Полетаев

ИСТОРИЯ РАЗВИТИЯ СУПЕРКОМПЬЮТЕРНОЙ ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

ВВЕДЕНИЕ

На каждом этапе развития средств вычислительной техники всегда находятся задачи, требующие для своего решения вычислительных мощностей, которые превышают мощности обычных компьютеров, доступных на рынке. Поэтому во все времена развития компьютерной отрасли ведущие страны мира занимались разработкой, реализацией и внедрением суперкомпьютеров.

Вокруг попыток дать определение термина «суперкомпьютер» всегда было много споров. Мы будем придерживаться следующего: компьютеры, включенные в какую – либо редакцию мирового рейтинга пятисот самых мощных машин мира (Top500, [1]), безусловно, заслуживают применения к ним термина «суперкомпьютер». Рейтинг Top500 ведется с июня 1993 года и обновляется два раза в год (в июне и в ноябре). Для ранжирования вычислительных установок в рейтинге Top500 используется некая процедура вычисления «реальной производительности» на тестовой задаче (программе) Linpack. Получаемые при этом оценки производительности обычно называют Linpack-производительностью. Тест Linpack по своей сути сводится к решению системы линейных уравнений с большим числом переменных.

Существуют различные подходы к достижению высокой вычислительной мощности вычислительных установок. Ряд таких подходов можно найти в работе [2]. В последнее время все большую популярность завоевывают архитектуры MPP и кластерный принцип построения суперкомпьютеров:

- используются широкодоступные компоненты – самые обычные процессоры, материнские системные платы (весьма часто двухпроцессорные), модули памяти, жесткие диски;
- из этих компонентов собирают большое число вычислительных узлов;

- вычислительные узлы соединяются между собою системной сетью, для этого используются либо существующие технологии высокоскоростных локальных сетей (например, сегодня это Gigabit Ethernet), либо специализированные высокопроизводительные сетевые технологии (Myrinet, SCI, Infiniband и т.п.);
- системную сеть, как правило, используют только для интеграции вычислительной мощности вычислительных узлов; как правило, это делается за счет реализации при помощи аппаратуры системной сети примитивов MPI [3];
- часто кроме системной сети узлы связывают еще различными сетями:
  - вспомогательной сетью, как правило, с протоколом TCP/IP (используется для передачи файлов и управления узлами);
  - сервисной сетью (например, для управления электропитанием, для мониторинга и управления вычислительными узлами и т. п.)

Возможность создания и использования суперкомпьютеров во всем мире относится к факторам стратегического потенциала оборонного, научно-технического и народно-хозяйственного значения, ибо прогресс любой развитой страны в современном мире невозможен без компьютеризации во всех сферах деятельности. При этом все более проявляется связь между достижениями страны в области создания и овладения передовыми компьютерными технологиями и ее потенциальными возможностями решать ключевые проблемы научно-технического прогресса. Особенно это проявляется в отношении решения научных фундаментальных и прикладных проблем.

Создание и применение современных супер-ЭВМ стало одним из основных направлений в ведущих странах мира. Общие затраты в этой области измеряются миллиардами долларов.

Производительность супер-ЭВМ на протяжении последних 20–30 лет возрастала ориентировочно на порядок за каждое пятилетие. Это отчетливо прослеживается по всей совокупности выпускаемой продукции, и нет оснований сомневаться в продолжении этой закономерности на следующее десятилетие.

По оценке американских экспертов стоимость выпускаемой на рынок новой супер-ЭВМ на 80–85 % окупается возможностями эффективного и быстрого ее применения для решения конкретных задач. Более того, перспективы развития супер-ЭВМ в ближайшем будущем в значительной степени будут определяться успехами вычислительной математики наряду с

успехами электроники. Поэтому в числе важнейших работ по развитию параллельных вычислительных технологий следует назвать:

1. Распараллеливание вычислений, создание новых методов и алгоритмов, ориентированных на эффективное использование в многопроцессорных системах, а также модернизация существующих с реализацией возможностей широкого параллелизма.
2. Разработку систем параллельного программирования, языковых и других средств с сохранением преемственности прикладных программных комплексов по отношению к аппаратным построениям распределенной вычислительной сети.
3. Создание программного обеспечения функционирования многопроцессорных систем, в том числе коммуникационной сети вычислительных модулей (ВМ) и между ВМ и внешними абонентами.
4. Разработку архитектур многопроцессорных вычислительных систем. Инженерное конструирование ВМ и вычислительного поля в целом.
5. Построение и задействование распределенных вычислительных и информационных систем (кластеров рабочих станций, многомашинных комплексов и др.).

Наряду с расширением области применения по мере совершенствования МВС происходит усложнение и увеличение количества задач в областях, традиционно использующих высокопроизводительную вычислительную технику. В настоящее время выделен круг фундаментальных и прикладных проблем, эффективное решение которых возможно только с использованием сверхмощных вычислительных ресурсов. Этот круг, обозначаемый понятием «Grand challenges», включает следующие задачи: структурную биологию, разработку фармацевтических препаратов, генетику, астрономию, транспортные задачи, гидро- и газодинамику, управляемый термоядерный синтез, эффективность систем сгорания топлива, геоинформационные системы, распознавание и синтез речи, распознавание изображений и многое другое.

1. НАЧАЛО ЭРЫ ГОНОК В МИРЕ ЭВМ-ГИГАНТОВ

Современный этап развития науки и техники характеризуется стремительным возрастанием сложности решаемых задач. Обусловлен этот объективный процесс преимущественно двумя причинами. Первая из них состо-

ит в том, что современная наука часто идет не по пути открытия, обнаружения отдельных закономерностей, а по линии моделирования комплексных явлений. Раньше при горении разного рода материалов, к примеру, химиками учитывалось максимум до 100 происходящих при этом химических реакций. При создании же современного химического лазера оказывается необходимым принимать во внимание ход около полутора тысяч реакций. И это не предел. На очереди стоят задачи биохимии. В живой клетке параллельно и последовательно протекает несколько десятков тысяч химических реакций. Да и сложность каждой реакции чрезвычайно высока, ведь число атомов, входящих в одну органическую молекулу, может достигать сотен миллионов.

Вторая причина усложнения научных задач состоит в том, что из года в год возрастает сложность техники – создаваемой человеком «искусственной реальности». Чудо техники, которым уже можно гордиться – современный авиалайнер – содержит более ста тысяч деталей. А число элементов в одной интегральной схеме может достигать нескольких десятков миллионов. При этом размер каждого элемента может быть менее 50 нанометров.

Сложность задач растет параллельно с ростом возможностей вычислительной техники. Не будь этих возможностей, вряд ли хватило бы смелости ставить задачи подобной сложности. Мощные ЭВМ стали необходимыми буквально во всех сферах исследований и разработок: при создании новых автомашин и летательных аппаратов, в проектировании ядерных реакторов, при исследовании космоса и плазмы, в гидрометеорологии, при разведке и добыче полезных ископаемых, а также в химии, точной механике и во многих других областях науки и техники.

Широко используются ЭВМ в микроэлектронике. Для развития вычислительной техники необходима не какая-нибудь, а очень хорошая вычислительная техника. Круг полностью замкнулся: чтобы иметь мощные ЭВМ, необходимы хорошие интегральные схемы; но для того, чтобы их создать, необходимы мощные ЭВМ. Вычислительная техника используется на всех этапах создания микросхем, начиная с проектирования и изготовления фотошаблонов и кончая управлением технологическим режимом их производства и контролем качества готовой продукции. Конечно, в некоторых случаях есть возможность обойтись прямыми натурными экспериментами, но они, как правило, требуют очень большого времени и грандиозных материальных затрат.

С очевидной неизбежностью прогресс в вычислительной технике начинает сказываться на развитии вычислительной математики.

Средства вычислительной техники дополняют физический эксперимент, участвуя в сборе и оперативной обработке различных параметров эксперимента. И хотя роль физического эксперимента по-прежнему остается очень важной, роль численного эксперимента возрастает стремительно.

Некоторые даже считают, что вычислительный эксперимент должен заменить устоявшиеся принципы научных исследований.

Чтобы подтвердить столь радикальное суждение, можно привести такой пример. Численное моделирование профиля самолетного крыла на самых мощных современных ЭВМ, какими располагают сейчас очень не многие страны, можно выполнить примерно за полчаса. Согласно американским данным, стоимость такого моделирования не превышает тысячи долларов.

Много это или мало? Если попытаться выполнить такое моделирование на ЭВМ, существовавших в 1960 году и с помощью соответствующих им алгоритмов, то только за аренду машины пришлось бы уплатить 10 млн. долларов, а расчет одного варианта длился бы 30 лет [4]. В то время, вероятно, натурные испытания были сопоставимы с расчетами по затратам и были выгоднее по времени. Но сейчас расчеты и вычислительные эксперименты стоят в десятки раз дешевле, и их можно провести в сотни раз быстрее. Кроме того, только с помощью расчетов возможно дальнейшее улучшение качества основных видов промышленной продукции. Для того же самолета необходимы расчеты каждой детали на прочность. Этого вообще нельзя сделать, используя традиционные методы проектирования.

Возросшие требования НИОКР и привели к созданию ЭВМ-гигантов, суперкомпьютеров. Со времени появления первых ЭВМ на каждом этапе развития вычислительной техники существовал свой компьютер рекордной производительности, обладающий максимального объема памятью. Но сам термин появился в 1977 году, когда фирма Cray Research [5], [6] выпустила машину Cray 1. Основатель фирмы Seymour Cray ранее работал на руководящей должности в фирме IBM и пытался убедить руководство IBM в перспективности разработки и производства суперкомпьютера. Когда это ему не удалось, он уволился и основал собственную фирму, которая и создала первый суперкомпьютер в 1977 году. Спустя пять лет таких машин работало в мире не так уж много – 61, из них 42 – в США, 7 – в Англии, 6 – в ФРГ, 4 – во Франции и две в Японии. При практически индивидуальном производстве таких машин – штука в месяц, в них постоянно вносились улучшения.

Следует подробнее сказать, почему в тот момент на освоение суперкомпьютера с такими параметрами не пошла IBM, и понадобилось создавать

отдельную фирму, не имеющую технологических традиций, следовательно – инерции.

Несмотря на высокую компактность электронных компонент, суперкомпьютеры фирмы Cray имели приличные габариты и весили многие тонны. Основную часть габаритов и веса «тянула» система охлаждения, в которой используются сжиженные газы. Поэтому трудно сразу сказать, где же лежала главная составляющая успеха фирмы - в электронике или в криогенике.

В последствии позиции фирмы Cray на мировом рынке существенно были потеснены. Суперкомпьютеры начали выпускать и другие американские фирмы: Control Data начала производить Cyber-205, компания Denelcor – HEP, корпорация Floating Point Systems – FPS/164.

Однако главная угроза всем им вместе взятым исходила из «страны восходящего солнца». В 1983 году известная японская фирма Fujitsu сообщила [7] о начале выпуска ею суперкомпьютеров FACOM VP-100 и FACOM VP-200. Интересная особенность их заключалась в том, что они были совместимы с машинами фирмы IBM. Японцы как бы продемонстрировали всем, в том числе и самой фирме IBM, как следовало осваивать суперкомпьютеры, не теряя при этом преемственности. Демонстрация сопровождалась совершенно реальной угрозой распространить на эту сферу вычислительной техники широкомасштабную международную торговую войну, ведущуюся на мировом рынке. Руководство американского филиала японской корпорации Amdahl заявило в тот момент, что намерено поставить супер-ЭВМ серии FACOM на рынки США и Западной Европы. Эти машины содержали векторные процессоры с максимальным быстродействием над числами с плавающей запятой: 267 (VP-100) и 533 (VP-200) млн. операций в секунду. Но дело не столько в высоких скоростях. Очень важным было то, что суперкомпьютеры Fujitsu могли выполнять уже имеющиеся программы, написанные, например, на языке фортран без их переработки более эффективно, чем ЭВМ фирмы Cray.

Данные сравнительных испытаний супер-ЭВМ Cray X-MP и FACOM VP-200, выполненных в Японии и США (прогон тестовых гидродинамических программ), показали, что при обработке длинных векторов производительность FACOM VP-200 составляет 115 млн. операций, а Cray X-MP – 70 млн. операций, а при обработке коротких векторов (6-10 элементов) производительность VP-200 была на 10 % ниже.

Упомянутая выше корпорация Amdahl, 49 % акций которой принадлежали – компании Fujitsu участвовала в реализации программы суперкомпьютеров в течение нескольких лет. В мае 1984 года она организовала

отделение с преднамеренно расплывчатым названием: «Подразделение систем специального назначения». Руководитель отделения сообщил, что у фирмы подготовлены планы по продаже суперкомпьютеров приблизительно 75 заказчикам в разных странах. О том, что иметь суперкомпьютер могли позволить себе очень немногие, говорят следующие цифры: минимальный комплект VF – 100 стоил 9,2 млн. долл., а VP-200 – 13,7 млн. долл., лицензия на пользование необходимым программным обеспечением обходилась пользователю в 16 тыс. долл. в месяц. За техническое обслуживание нужно было платить 35,9 тыс. долл. – для VP-100 и 51 тыс. долл. – для VP-200 в месяц.

Свой суперкомпьютер с производительностью 300 млн. операций над вещественными числами стала изготавливать фирма Hitachi. По некоторым сведениям, она создала также экспериментальную модель с быстродействием 1,3 млрд. операций. Свои типы суперкомпьютеров изготавливала японская компания Nippon Electric. Один из них также имел быстродействие 1,3 млрд. операций [8].

Характеристики основных супер-ЭВМ, построенных и планируемых на тот момент времени, приведены ниже в таблице. Максимальная производительность дана в млн. операций над вещественными числами в сек., а объем памяти – в млн. байтов.

Сразу же отметим, что приведенная таблица очень неполная. К примеру, фирма CDC создала семь моделей машины cyber 205 серии 600, а именно 611, 612, 622, 642, 644, 682, 684. В таблице приведены данные лишь для старшей модели. Имелись три модели Cray Y-MP. Как уже отмечалось выше, корпорация Floating Point Systems производила суперкомпьютеры серии FPS. Максимальная производительность FPS-164/MAX достигала 341 млн. операций, но объем ее памяти был невелик – 120 млн. байтов. Фирма Burroughs производила ILLIAC-IV, имеющий быстродействие 300 млн. операций. Кроме того, в ряде фирм и организаций использовались другие специализированные вычислительные системы высокой производительности.

Характеристики супер-ЭВМ

Фирма	Название ЭВМ и дата поставки	Максимальная производительность	Максимальная емкость ОЗУ
Fujitsu	FACOM VP-100 1983 FACOM VP-200 1983	267 533	128 256
Hitachi	8-810/20 1983 1984/85	630 1.300	256 -
Nippon Electronic Company	SX-1 1984 SX-2 1984	570 1.300	256 256
Японский супер-ЭВМ	1989	10.000	1.000
Cray Research	Cray 1 M 1983 Cray X-MP/ 22,24 1983 Cray X-MP/48 1984 Cray 2 1984 Cray 3 985/86	250 400-630 1.000 1.000-1.200 10.000	32 32+256
CDC	Cyber 205/684 1984 Cyber 2XX 1986/86	800 2.000	544 256
CDC-ETA Systems	GF-10 1986 GF-40 -	10.000 40.000	-
Denelcor	HEP-2 1985/86	1.000	2.000
Программа стратегических вычислений, реализуемая управлением DARPA	- 1985/86 - 1989	1.000 1.000.000	-

Некоторые литературные источники по данной теме были недоступны ранее и остаются недоступными в настоящее время, а некоторые содержат неточности или даже ошибки разного рода. Последнее связано с тем, что фирмы не выдерживали сроки первых поставок своих изделий, о которых они заявили или поставляли изделия с другими параметрами, нежели предполагалось первоначально. Некоторые модели не представляют интереса.

Все ранние модели суперкомпьютеров характеризовались прямым, «лобовым» решением задачи повышения быстродействия. Можно сказать, что главный лозунг разработчиков суперкомпьютеров в тот момент звучал так: «Максимум быстродействия любой ценой». Как бы само собой подразумевалось, что увеличение быстродействия окупится даже при очень больших затратах. Высокое быстродействие элементов суперкомпьютера достигалось использованием новых материалов и сверхнизких температур. Насколько далеко зашли замыслы разработчиков, можно, например, судить по такому факту: фирма Fujitsu вела исследования поведения арсенида галлия при температурах, предельно близких к абсолютному нулю.

Для создания перспективных суперкомпьютеров разрабатывались:

- корпуса для интегральных схем с матричным расположением выводов с шагом 0,5 мм, имеющих 300–500 выводов;
- технология и методы автоматического монтажа;
- многоконтактные разъемные соединители для интегральных схем на 300-500 выводов и для печатных плат на 1200–1800 контактов;
- высокоэффективные системы охлаждения, использующие жидкие газы;
- волоконно-оптические линии.

Использование новых технологий позволяло также существенно уменьшить габариты машин. К примеру, центральный процессор ЭВМ Cray 1 имел высоту 2 м и диаметр 2,7 м. А размеры Cray 2 были всего лишь 66 см и 96,5 см, соответственно. Центральный процессор Cray 2 был выполнен на бескорпусных ECL – интегральных схемах. Он помещался в специальном контейнере, через который прокачивался фторуглеродный хладагент. Таким образом, охлаждение осуществлялось посредством того, что все детали «омывались» сжиженным газом, а размеры ЦП оказывались настолько малы, что в нем не было ни одного проводника длиннее 41 см.

Естественно, что бесхитростное, «лобовое» решение приводит к возникновению новых трудностей. Использование сверхвысоких частот существенно, можно сказать, радикально повышает требования к качеству компонент и монтажа. Небрежно установленная деталь в аппаратуре такого

класса начинает работать как антенна, излучать и принимать на себя шумы. Все размеры печатных и навесных проводников должны строжайше соответствовать расчетам. В противном случае электрический сигнал не просто дойдет по назначению, а может отразиться и породить волновой процесс. Волны в проводниках в состоянии так запутать функционирование суперкомпьютера, что причину искажений практически невозможно выявить.

Трудности, на которые натолкнулись сторонники прямого повышения быстродействия, не могли не вызвать множества критических замечаний. Критика критикой, но нужно было предложить какое-либо альтернативное решение. Ведь громоздкие криогенные установки с фреоном или жидким азотом появились в составе суперкомпьютеров отнюдь не по глупости, а попросту вследствие того, что не было видно иного реализуемого технического решения.

Каков же наиболее вероятный конкурент прямому повышению скорости вычислений? Им могли быть только создание таких ЭВМ, в которых большие количества – тысяча и более процессоров – работают параллельно, в которых мощная память позволяет быстро записывать и считывать информацию. Но относительно устройства таких ЭВМ и в настоящее время полная неясность и идейный разброд. Пока принципиально важные вопросы: как должна быть организована память, каким должен быть язык программирования? Каковы должны быть, наконец, методы трансляции, если счет будет идти одновременно во многих процессорах? Может быть, и транслировать программы с языка высокого уровня нужно также на многих процессорах параллельно? Неясно даже, каким образом должны быть соединены между собою эти считающие одновременно процессоры.

Рассмотрим лишь некоторые подходы, которые развивались в то время. Какой бы ни была вычислительная система, она представляет совокупность связанных между собой устройств. В каждый момент времени эти устройства либо простаивают, либо выполняют полезную работу, то есть, заняты хранением, пересылкой или переработкой информации. Системы различаются как составом устройств, так и видом связей. Как правило, в любой системе имеется много устройств с постоянными связями, но в общем случае могут быть как редко изменяемые, так и часто изменяемые связи, причем число изменяемых связей в различных системах может колебаться в довольно больших пределах.

В используемых в тот момент суперкомпьютерах широкое распространение получили конвейерные системы. Принцип конвейерной обработки информации основан на разбиении процесса вычисления на несколько последовательных этапов. Количество этапов называется длиной конвейера.

Информация сначала попадает в первый блок. После обработки на первом этапе результаты передаются во второй блок, а в первый можно загрузить следующую порцию информации и т.д. Таким образом, одновременно выполняется количество команд, равное длине конвейера. Как правило, одновременно образуется несколько вычислительных конвейеров, и предусматривается «механизм зацепления», суть которого состоит в том, что часть продукции с одного конвейера может передаваться на другие конвейеры. При этом передаваемая продукция может браться как с конечного этапа работы конвейера, так и с его промежуточных этапов. Различие организованных таким образом компьютеров в том, что на одних конфигурация обрабатывающих конвейеров может описываться в программе, а на других устанавливается автоматически аппаратными средствами.

Кроме того, используются довольно широко, но уже значительно меньше, векторные и матричные мультипроцессоры, для которых векторы или матрицы являются элементарными объектами, в том смысле, что вектор или матрица обрабатывается сразу целиком.

В настоящее время огромный класс задач исследован на предмет «конвейеризации» и «векторизации» и практически реализован на суперкомпьютерах указанного вида. Однако считается, что такого рода подходы не обладают в достаточной мере универсальностью, имеют ряд неудобств для пользователя, а идея параллелизма не используется в них в полной мере. Поэтому идет поиск в других направлениях.

Следующий тип распараллеливания – использование мультипроцессоров на основе клеточной логики. Такой мультипроцессор представляет собой совокупность элементарных процессоров, связанных между собой в некоторую сеть. Каждый процессор способен считывать в пределах некоторой окрестности информацию у присоединенных к нему ближайших «соседей». В зависимости от нее он способен принимать решение и переходить в новое состояние. Такая сеть как бы «живет и развивается». В данный фиксированный момент процессор находится в некотором элементарном состоянии, потом все они сразу переходят в новые состояния, каждый в свое и т.д. Элементарные процессоры могут быть соединены самым различным образом: в линию, в кольцо, в виде дерева, в виде квадратной таблицы, в узлах сети, составленной из треугольников или шестиугольников, в узлах трехмерной сети иди в узлах так называемых гиперкубов. Могут использоваться также различные нерегулярные структуры. Хотя теоретически доказано, что такие мультипроцессоры универсальны в том смысле, что на них возможно выполнить любой алгоритм, однако на практике на них затруднительны программирование и трансляция. Поэтому эти устройства полу-

чили распространение в основном для решения задач обработки изображений ввиду структуры применяемых там алгоритмов.

Еще один подход – систолические мультипроцессоры. Более точно следовало бы сказать – систолическое программирование. Представим себе, что некоторый процессор находится в некотором фиксированном состоянии, можно сказать, «хранит информацию». Его сосед, «заглянув к нему», переходит в данное состояние и т.д. Наблюдается движение информации или, как еще говорят, «поток». Программирование на основе клеточной логики сводится к описанию функций перехода процессоров из одного состояния в другое. Систолическое программирование представляет собой описание потоков информации и процессов их изменения в функциональных устройствах. Такое программирование оказывается весьма удобным для записи различных алгоритмов из алгебры и геометрии.

Как уже отмечалось, поток информации может претерпевать изменения в функциональных устройствах. И здесь появляется еще одна возможность. Можно не описывать в программе действия функциональных устройств, а сами потоки данных могут активизировать необходимые устройства и «заставлять» выполнять их нужную работу. Такие машины называются «дэйта флоу компьютерс». Для них создаются специальные языки программирования, ориентированные на значения данных.

В этом направлении были сконцентрированы большие усилия. Однако до сих пор они не привели к каким-то выдающимся результатам.

Остановимся на некоторых проектах.

Машина NQN-VQN разрабатывалась в Колумбийском университете (США) под руководством D.Shaw, являлась параллельной вычислительной системой с потоковым управлением [9]. Она представляла собой «дерево», объединяющее 1 млн. процессоров, каждый из которых – миниатюрная ЭВМ с однобайтовым АЛУ, 64-байтовой памятью и 8 однобайтовыми регистрами. Имеется один центральный (в вершине «дерева») источник команд, которые поступают ко многим процессорам, обрабатывающим потоки данных. Отмечают два недостатка NON-VON – симметричная структура дерева осложняет эффективное наращивание системы, а выбор очень малых размеров ПЭ ограничивает тип решаемых задач. В частности, для большинства нечисловых применений задача не может быть разбита на миллион идентичных маленьких частей.

Другая система с потоковым управлением – булевый мультипроцессор в виде 10-мерного куба, разработанный в Калифорнийском университете (США). В нем использовались ПЭ сравнительно больших размеров (мик-

ропроцессоры 8086). Общее число ПЭ – около одной тысячи. В данной системе отсутствуют проблемы, характерные для машины NON-VON.

Проект Ultracomputer Нью-Йоркского университета был рассчитан на решение крупномасштабных вычислительных задач. По проекту предполагалось использовать тысячи однокристальных СБИС-процессоров [10].

В Массачусетском технологическом институте профессор Arvind выполнял еще один проект архитектуры для реализации тысячекратного распараллеливания. Машина строилась на микропрограммируемых процессорах, взаимосвязанных программируемой сетью с широкой пропускной способностью. Топология сети – 7 мерный гиперкуб с соединениями, обеспечивающими скорость передачи 4 млн. байт в секунду. Там же была разработана структура памяти, которая обеспечивала синхронизацию [11], [12].

Не затрагивая другие подхода, отметим, что их математический анализ средствами современной математики в значительной мере затруднялся и затрудняется в настоящее время большим разнообразием, многочисленными плохо определенными понятиями и отсутствием формализованных принципов исследования.

Первые суперкомпьютеры фирмы Cray представляли собой машины, организованные по конвейерному принципу. Немного коснемся архитектуры ЭВМ Cray X-MP. Она содержала два идентичных процессора, устройство, осуществляющее связь между ними, как уже отмечалось, центральную скоростную намять, дополнительную память, а также блок связи с устройствами ввода-вывода. Ниже на рис. 1 показана структурная схема ЭВМ Cray X-MP.

Процессоры могли работать как независимо, так и над выполнением одной программы в параллельном режиме. Каждый из них включает в себя, 13 функциональных устройств, 152 регистра пяти различных видов, 4 буфера для инструкций. Функциональные устройства условно можно разбить на 4 группы по типу выполняемых операций: адресные, скалярные, устройства с плавающей запятой и векторные. Все устройства работают по конвейерному принципу и независимо друг от друга. Регистры используются функциональными устройствами для хранения входных данных и результатов команд. Регистры разбиваются на пять групп, а именно: адресные, скалярные, векторные и буферные двух видов.

Центральная память представляет собой так называемую многопортовую память. Она имеет 4 порта: 2 для чтения, 1 для записи и 1 связанный с устройствами ввода-вывода. Все они могут функционировать параллельно. Это особенно важно для применения «механизма зацепления» конвейеров при векторных вычислениях, когда результирующий вектор ис-

пользуется как входной в последующих операциях, при этом, возможно, в другом конвейере.

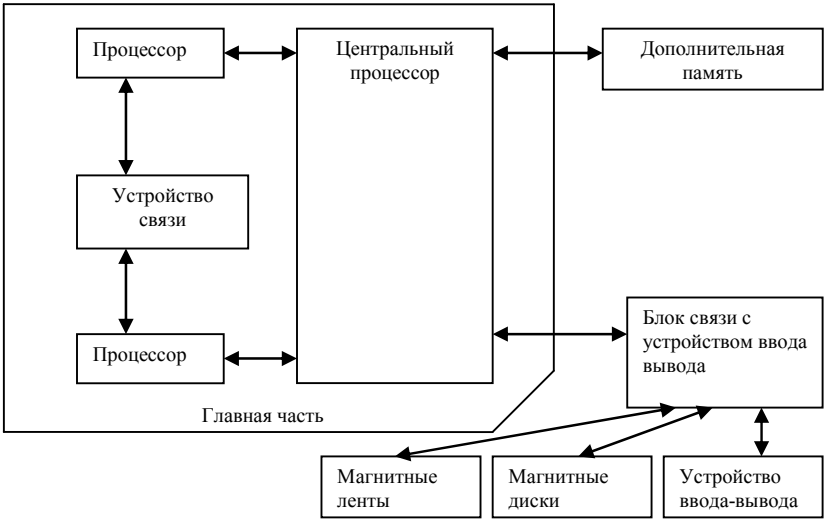


Рис. 1. Структурная схема ЭВМ Cray X-MP

Центральная память связана каналами для передачи информации с устройствами ввода-вывода, устройствами контроля и дополнительной памятью. Последний канал скоростной, скорость обмена информацией по нему – 1250 млн. байтов в секунду.

Дополнительная память имела емкость 64,128 или 256 млн. байтов (три различных варианта), состояла соответственно из 16, 32 или 64 банков. Каждый банк содержал 72 модуля. При этом последовательно расположенные коды программы или данных помещаются в разных модулях памяти. Повышение скорости обмена с памятью достигается за счет конвейеризации последовательных обращений к памяти, а также за счет параллельного вычисления адресов. Время доступа к этой памяти – 38 наносекунд. Ее вес – 1,5 тонны, и она занимала 10 м^2 площади пола.

Блок связи с устройствами ввода-вывода осуществлял связь с магнитными лентами, магнитными дисками, телетайпами, графическими и буквенно-цифровыми дисплеями и т.д. Он имел 2, 3 или 4 высокоскоростных

процессора, до 64 млн. байтов буферной памяти. Общий объем памяти подключаемых дисков – 48600 млн. байтов.

За всю историю развития отечественной суперкомпьютерной отрасли только четыре вычислительные установки – МВС-1000М, МВС-5000БМ, СКИФ К-500, СКИФ К-1000, – смогли попасть в мировой рейтинг Top500, четыре раза отечественные установки входили в первую сотню мирового рейтинга (МВС-1000М – 3 раза, СКИФ К-1000 – 1 раз). Эти суперкомпьютеры разрабатывались и изготавливались:

- МВС-1000М, МВС-5000БМ – силами ФГУП НИИ «Квант», ИПМ им. М. В. Келдыша РАН и Межведомственного суперкомпьютерного центра (МСЦ);
- СКИФ К-500, СКИФ К-1000 – в рамках суперкомпьютерной программы «СКИФ» Союзного государства силами ОИПИ НАН Беларуси (Минск), НИИ ЭВМ (Минск), компании «Т-Платформы» (Москва) и ИПС РАН (Переславль-Залесский).

Суперкомпьютер	**МВС-1000М**	**СКИФ К-500**	**МВС-5000БМ**	**СКИФ К-1000**
Разработчики и изготовители	ФГУП «Квант», ИПМ РАН, МСЦ	суперкомпьютерная программа «СКИФ» Союзного государства. ОИПИ НАН Беларуси (Минск), НИИ ЭВМ (Минск), Т-Платформы (Москва), ИПС РАН (Переславль-Залесский)	ФГУП «Квант», ИПМ РАН, МСЦ	суперкомпьютерная программа «СКИФ» Союзного государства. ОИПИ НАН Беларуси (Минск), НИИ ЭВМ (Минск), Т-Платформы (Москва), ИПС РАН (Переславль-Залесский)
Завершение изготовления	июнь 2002 года	сентябрь 2003	июнь 2004 г., модернизация: сентябрь 2004 г.	сентябрь 2004

Место установки	Межведомственный суперкомпьютерный центр (МСЦ, Москва, Президиум РАН)	Объединенный иститут проблем инфоматики (ОИПИ НАН Беларуси, Минск)	Межведомственный суперкомпьютерный центр (МСЦ, Москва, Президиум РАН)	Объединенный иститут проблем инфоматики (ОИПИ НАН Беларуси, Минск)
Пиковая / Linpack-производительность (млрд. операций в секунду)	1 024 / 564 с сентября 2002 1 024 / 734.6	716.8 / 423.6	1 075.2 / 722.1 с сентября 2004: 2 112 / 1 401	2 534.4 / 2 032
Тип процессора / Тип системной сети	Alpha EV67 667 MHz / Myrinet	Pentium IV Xeon 2.8 GHz / SCI 3D	IBM eServer BladeCen-ter JS20 PowerPC970 1.6 GHz / Myrinet	AMD Opteron 2.2 GHz / Infinivand
Число узлов / число процессоров	384 / 768	64 / 128	84 / 168 с сентября 2004: 168 / 336	288 / 576
Место / выпуск рейтинга Top500	64 / июнь 2002 г 74 / ноябрь 2002 г 95 / июнь 2003 г 189 / ноябрь 2003 г 392 / июнь 2004 г	407 / ноябрь 2003 г.	399 / июнь 2003 г 210 / ноябрь 2004 г	98 / ноябрь 2004 г

Полное название программы: «Разработка и освоение в серийном производстве семейства моделей высокопроизводительных вычислительных систем с параллельной архитектурой (суперкомпьютеров) и создание прикладных программно-аппаратных комплексов на их основе».

С одной стороны, вхождение в список TOP500 отечественных вычислительных установок говорит о том, что мы владеем технологиями, которые

доступны только весьма ограниченному списку стран:

- суперкомпьютеры из списка TOP500 установлены всего лишь в пяти десятках стран мира;
- суперкомпьютеры из списка TOP500 способны изготавливать только страны, входящие в весьма узкий (из полутора десятка стран) «элитный клуб разработчиков суперкомпьютеров»;
- суперкомпьютеры из «первой сотни» способны изготавливать только в США, Японии, Китае и России.
- ряд мощных суперкомпьютеров был изготовлен также во Франции и в Индии, но информация о них не попала в официальный список «первой сотни».

Таким образом, гонки в мире ЭВМ-гигантов продолжаются.

2. СУПЕРКОМПЬЮТЕРНЫЕ ТЕХНОЛОГИИ И ПРОЕКТЫ В США

В США развитие и применение суперкомпьютеров (*High performance computing – HPC*) за последние десять лет получило достаточно высокую отдачу в виде конкретных научно-технических проектов как в гражданской, так и военной областях. При этом основная тенденция развития данного направления заключается в широком обмене опытом, программным обеспечением, алгоритмами и специалистами между всеми участниками проектов и их заказчиками независимо от ведомственной принадлежности.

На рис. 2 представлена динамика распределения суперкомпьютеров в США по объектам их использования. Анализ диаграммы показывает, что 2000 год для высокопроизводительных вычислений в США стал решающим с точки зрения промышленного использования технологии суперкомпьютеров, а 2001 год – переломным, когда общее количество установленных компьютеров в промышленности превысило их количество в академических и научно-исследовательских организациях. Это наглядно демонстрирует переход технологий высокопроизводительных вычислений из чисто научных проектов непосредственно в производство. Наряду с этим прослеживается тенденция увеличения количества установленных суперЭВМ на секретных объектах и объектах производителя, что свидетельствует о стратегической важности данного направления развития информационных технологий для национальной безопасности США.

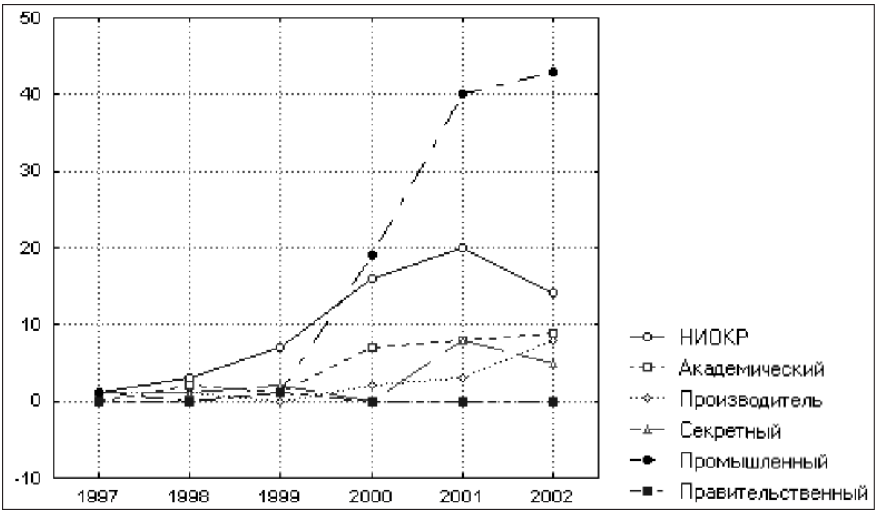


Рис. 2. Динамика распределения суперкомпьютеров, установленных в США, по объектам

Одной из первых крупномасштабных попыток совершить прорыв в использовании суперкомпьютеров в военно-технических и военно-стратегических интересах является так называемая программа «звездных войн», инициированная администрацией Рональда Рейгана в марте1983 г.

В рамках проекта Стратегической оборонной инициативы – СОИ (Strategic Defense Initiative – SDI) предполагалось создать и развернуть эшелонированную систему противоракетной обороны (на земле, в воздухе и в космосе). Среди научно-технических и инженерно-конструкторских задач, связанных с осуществлением этого проекта, особо выделялась задача селекции ложных целей на заатмосферном участке полета ядерных боеголовок баллистических ракет, для решения которой требовался колоссальный объем вычислений по тем временам – миллиарды операций в секунду с числами, представленными в формате с плавающей точкой.

Несмотря на то, что к началу 80-х годов уже существовали испытанные варианты систем ПРО наземного базирования, их эффективность ограничивалась гарантированным поражением ракет противника преимущественно на атмосферном участке, когда боеголовки при входе в плотные слои атмосферы подвергаются сильному нагреву и могут быть идентифицированы на фоне ложных целей по скорости. В условиях полета на заатмосферном уча-

стке в безвоздушном пространстве отличить боевые цели от ложных при равных скоростях (до 7 км/с.) и физических размерах практически невозможно без использования космических датчиков радиолокационного слежения. Кроме того, для управления орбитальной группировкой космических платформ, на которых предполагалось разместить средства обнаружения и поражения боеголовок и ракет (кинетических, лазерных и др.) требовалось, чтобы большая часть вычислений проводилась непосредственно на месте, т.е. в космосе в условиях ограниченного ресурса времени для принятия решения, что накладывало значительные ограничения не только на быстродействие и память, но и вес, объем, потребляемую энергию и надежность компьютеров и их программного обеспечения.

Принимая во внимания достигнутый к тому времени уровень развития элементной базы и архитектуры компьютеров, решить подобную задачу можно было только за счет использования комплекса технологий: сверхбыстродействующих и сверхбольших интегральных микросхем, параллельной и векторной обработки данных, методов искусственного интеллекта и высоконадежного программного обеспечения.

Например, суперкомпьютер «Cray-1» американской компании Cray Research Inc. в 1976 году выполнял 240·млн. арифметических операций с плавающей точкой в секунду (240 Мфлоп) и стоил от $4 до $11 млн. в зависимости от комплектации. Стоимость таких вычислительных систем становилась просто астрономической, если к цене оборудования добавлялись расходы на изготовление программного обеспечения.

С этой целью в рамках программы СОИ были открыты специальные технологические подпрограммы, получившие название «стратегические сверхпроизводительные вычисления» (Strategic Supercomputing), «сверхбольшие интегральные микросхемы» (Very Large Integrated Circuit – VLIC), «сверхскоростные интегральные микросхемы» (Very High Speed Integrated Circuit – VHSIC), «стратегическое адаптируемое и надежное программное обеспечение» (Strategic Adaptable And Reliable Software – STARS).

Результаты не замедлили себя ждать. Созданный в 1988 году суперкомпьютер «Cray-Y-MP» объединял уже до 16 процессоров, обладал пиковой производительностью 2670 Мфлоп и стоил от $2,5 до $16 млн. Тем не менее, для реализации проекта СОИ этого было явно мало: по оценкам американских специалистов бортовые вычислительные системы космического базирования должны были обладать пиковой производительностью не менее 50 Гфлоп.

И здесь не лишним будет упомянуть тот факт, что к началу открытия работ в рамках программы СОИ в Советском Союзе тоже были достигнуты

ощутимые результаты в разработке компьютеров. Убедиться в этом американцы смогли еще во время совместных космических полетов по программе «Союз-Аполлон», когда в подмосковном ЦУПе обработка телеметрической информации на БЭСМ-6 проводилась быстрее почти на полчаса, чем в Хьюстоне.

Многопроцессорный вычислительный комплекс «Эльбрус-1», выпущенный в 1979 году, включал 10 процессоров и базировался на схемах средней интеграции. В этой машине советские ученые опередили американцев, создав симметричную многопроцессорную систему с общей памятью. По принципам построения система команд «Эльбрусов» близка системе команд машин компании Burroughs, считающейся нетрадиционной. Машина «Эльбрус-1» обеспечивала быстродействие от 1,5 Мфлоп до 10 Мфлоп, а «Эльбрус-2» – более 100 Мфлоп. «Эльбрус-2», работа над которым была завершена в 1985 году, также представлял собой симметричный многопроцессорный вычислительный комплекс из 10 суперскалярных процессоров на матричных БИС, которые выпускались в Зеленограде. «Эльбрусы» вообще несли в себе ряд революционных новшеств. Суперскалярность процессорной обработки, симметричная многопроцессорная архитектура с общей памятью, реализация защищенного программирования с аппаратными типами данных – все эти возможности появились в отечественных машинах раньше, чем на Западе.

Особо следует выделить создание единой операционной системы для многопроцессорных комплексов (которым руководил Борис Арташесович Бабаян, в свое время отвечавший за разработку системного программного обеспечения БЭСМ-6). Одной из важнейших задач этой ОС было управление параллельно выполняющимися процессами и их синхронизация – одна из самых сложнейших задач в области высокопроизводительных вычислений.

К сожалению, ориентация отечественной компьютерной индустрии в основном на военные проекты привела эту отрасль к началу 90-х годов в финансовый тупик, когда экономика страны была уже не в силах содержать дорогостоящие проекты военно-промышленного комплекса. Аналогично, в США был закрыт один из самых амбициозных и дорогостоящих научно-технических проектов, сопоставимый по своей значимости только с программой полетов человека на Луну. Тем не менее «звездные войны», при всей своей фантастичности на грани авантюры, стали подлинным катализатором целого ряда важнейших направлений развития современных технологий, включая суперкомпьютеры.

Опыт боевых действий в Персидском заливе, Югославии, Албании, террористические акты 11 сентября 2001 г. В Нью-Йорке и Вашингтоне, наконец, операция по уничтожению баз боевиков Аль-Каиды в Афганистане активно стимулировали новые направления в использовании Пентагоном высокопроизводительных вычислений. За двадцать лет, которые прошли после памятной речи Рональда Рейгана о «звездных войнах», Пентагон вышел на качественно новый уровень в своих НИОКР, связанных с использованием суперкомпьютеров, создав собственную сеть высокопроизводительных ЭВМ, которая сегодня обслуживает свыше 4 тыс. ученых и инженеров в 100 ведущих американских университетах, исследовательских центрах и лабораториях, занятых в 600 крупных военных проектах.

Сегодня можно с большой долей уверенности говорить о том, что это направление развития информационных технологий, которое в значительной степени повлияло на судьбу СОИ, заставив отказаться от идеи размещения лазерного оружия в космосе, достигло или во всяком случае очень близко к тем рубежам, о которых в начале 80-х годов могли только мечтать.

Развитие технического прогресса по спирали находит свое отражение в новой программе Пентагона создания общенациональной высокоскоростной научно-исследовательской и инженерно-конструкторской сети суперкомпьютеров DREN (Defense Research Engineering Network), которая должна сделать США неоспоримым лидером в практическом применении целого комплекса стратегических технологий, основанных на высокопроизводительных параллельных вычислениях с высокоточной визуализацией объектов и математическим моделированием физических процессов.

На сегодняшний день высокоскоростная сеть суперкомпьютеров DREN объединяет 4 главных вычислительных центра коллективного пользования и 17 региональных распределенных вычислительных центров, размещенных по всей территории США. На ее развитие и модернизацию в период с 1997 по 2007 гг. выделено в общей сложности $1,73 млрд. В составе этой самой мощной в мире сети суперкомпьютеров насчитывается 64 объекта электронно-вычислительной техники (ЭВТ) общей производительностью 12 Тфлоп.

При этом 96% всей вычислительной мощности сети DREN сосредоточено в 4 главных вычислительных центрах коллективного пользования: Центре исследования аэрокосмических систем на военно-воздушной базе Райт-Паттерсон (шт. Огайо) – ASC, Исследовательской лаборатории сухопутных войск на полигоне армии США в Абердине (шт. Мэриленд) – ARL-APG, Центре исследований и разработок сухопутных войск в Виксбурге

(шт. Миссисипи) – ERDC, Океанографическом управлении ВМС США в Космическом центре им. Джона Стениса (шт. Миссисипи) – NAVO.

Распределенные центры сети DREN, выполняющие роль региональных вычислительных центров, на долю которых приходится до 75% всех проектов Пентагона в области высокопроизводительных вычислений, включают в себя: Центр авиационных систем вооружения ВВС США на авиабазе в Эглин (шт. Флорида) – AAC, Испытательный центр ВВС США на авиабазе Эдвардс (шт. Калифорния) – AFFTC, Исследовательская лаборатория ВВС США в Риме (шт. Нью-Йорк) – AFRL/IF, Исследовательская лаборатория ВВС США на авиабазе Райт-Паттерсон (шт. Огайо) – AFRL/SN, Вычислительный центр арктического региона в Фэйрбэнкс (шт. Аляска) – ARSC, Исследовательский центр высокопроизводительных вычислений сухопутных войск США в Минеаполисе (шт. Минесота) – AHPCRC, Центр инженерных разработок на авиабазе Арнольд (шт. Теннеси) – AEDC, Национальный объединенный центр интеграции на авиабазе Шривер (шт. Колорадо) – JNIC, Центр высокопроизводительных вычислений в Мауи (шт. Гавайи) – MHPCC, Центр воздушной войны ВМС США в Патаксен-Ривер (шт. Мэриленд) – NAWCAD, Центр воздушной войны ВМС США в Чайна-Лейк (шт. Калифорния) – NAWCWD, Исследовательская лаборатория ВМС США в Вашингтоне (федеральный округ Колумбия) – NRL, Технический испытательный центр арсенала в Редстоун (шт. Алабама) – RTTC, Штаб командования космических сил и противоракетной обороны в Хантсвилле (шт. Алабама) – SMDC, Космический центр ВМС США в Сан-Диего (шт. Калифорния) – SSCSD, Центр исследований и разработок бронетанковой техники в Уоррен (шт. Мичиган) – TARDEC, Ракетный полигон в Уайт-Сэндс (шт. Нью-Мексика) – WSMR.

География размещения вычислительных центров сети DREN говорит сама за себя – охват значительной территории континентальной и островной части США (13 штатов и один федеральный округ) обеспечивает глобальный доступ к ее ресурсам в любое время суток с максимальной загрузкой вычислительных мощностей четырех главных центров коллективного пользования. Удаленный доступ к ресурсам сети DREN осуществлялся по высокоскоростным каналам передачи данных со скоростью до 1,5 Гбит в с. (к 2004 г. до 10 Гбит в с.). Пользователями сети могут быть как военные, так и гражданские научно-исследовательские организации, и университеты штатов (Алабама, Вирджиния, Гавайи, Джорджия, Иллинойс, Калифорния, Миссисипи, Мичиган, Мэриленд, Огайо, Пенсильвания, Северная Каролина, Теннеси, Техас, Флорида), участвующие в совместных проектах в области высокопроизводительных вычислений.

С этой целью Пентагон выдвинул достаточно смелую, с точки зрения военной бюрократии, и революционную в технологическом отношении инициативу, получившую название «Единой поддержки программного обеспечения высокопроизводительных вычислений» (Common High Performance Computing Software Support Initiative).

Суть этой инициативы заключается в доступе пользователей не только к вычислительным ресурсам центров (процессорам, каналам, оперативной и внешней памяти), но и алгоритмам, их программной реализации. Характерно, что 97% прикладного программного обеспечения для суперкомпьютеров на сегодняшний день написано на современных версиях классического Фортрана (77,90, HPF), хорошо известная и непревзойденная математическая мощность которого при решении задач численными методами (линейные и дифференциальные уравнения, преобразование Фурье и др.) в сочетании с практически неограниченными возможностями в области системного программирования языка Си++ позволяют быстро разрабатывать полноценные приложения с удобным для пользователя графическим интерфейсом.

Особое место в этой инициативе занимают вопросы информационной безопасности, связанные с распределенным доступом к вычислительным ресурсам и программам сети DREN. Предусмотрены как несекретные, так и секретные анклавы ресурсов и пользователей сети. Создана единая распределенная по всем объектам сети система датчиков обнаружения несанкционированного вторжения NIDS, монтирование и испытание которой проводилось под непосредственным наблюдением и контролем со стороны специалистов АНБ. При этом особое внимание уделено высокоскоростному трафику в режиме ATM, надзор за которым осуществляется круглосуточно специальной группой немедленного реагирования. Кроме того, в плановом порядке Управление информационных систем Пентагона осуществляет так называемый аудит информационных ресурсов сети DREN и консультации персонала по предотвращению, как перегрузки ресурсов, так и несанкционированного вторжения. Но не только военные, ученые и инженеры используют в США суперкомпьютеры в своих профессиональных интересах. Бизнес уверенно прогрессирует в этом наукоемком секторе применения высоких технологий, отвоевывая шаг за шагом ведущие позиции у пионеров высокопроизводительных вычислений: свыше 52% всех суперкомпьютеров, входящих в список 500 самых быстродействующих ЭВМ на земном шаре, сегодня заняты в маркетинге, торговле, финансах, телекоммуникациях и других секторах частного предпринимательства.

Лидер складской индустрии на мировом рынке услуг в сфере снабжения, американская компания Staples Inc., доходы которой превышают $9 млрд. в год, закупила мощный суперкомпьютер IBM серии SP для повышения эффективности и снижения себестоимости своих торгово-закупочных операций. Основу суперкомпьютера составляет 64-х процессорная вычислительная система, с помощью которой распределенная база данных (DB2), организованная на основе механизма репликации, ежедневно обновляется на общем жестком магнитном носителе объемом 4 Тбайт. Тем самым, пользователи (50 тысяч сотрудников) получают оперативный доступ к глобальному информационному ресурсу для получения сведений о ценах, объемах и номенклатуре поставок, сроках отгрузки по всем филиалам компании (свыше 1200) во всем мире (США, Канаде, Великобритании, Германии, Нидерландах, Португалии и др. странах), на основе которых они могут делать краткосрочные и долгосрочные прогнозы деловой активности клиентов, планировать транспортные операции, отслеживать прохождение грузов.

3. СТАТИСТИКА И АНАЛИЗ ТЕНДЕНЦИЙ РАЗВИТИЯ СУПЕРКОМПЬЮТЕРОВ В США

Для того, чтобы лучше понять суть происходящего в такой бурно развивающейся области информационных технологий, как высокопроизводительные вычисления, специалисты пользуются богатым арсеналом современных методов математической статистики или, как их принято называть сейчас, технологией многомерного анализа данных. Статистика позволяет компактно описать данные, понять их структуру, провести классификацию, увидеть закономерности в хаосе случайных явлений. Даже простейшие методы визуального анализа данных позволяют существенно прояснить сложную ситуацию, первоначально поражающую нагромождением цифр.

При проведении исследований ограничимся выборочными объектами электронно-вычислительной техники (ЭВТ) – вычислительными центрами коллективного пользования (ВЦКП) уже известной нам сети суперкомпьютеров DREN, данные о которой приведены в серии ежегодно публикуемых отчетов Управления перспективных исследований Пентагона DARPA. Следует иметь в виду, что с позиций теории математической статистики такого рода исследования всегда носят выборочный характер, поскольку генеральная совокупность объектов или, как ее еще называют, популяция достаточно велика: список 500 самых мощных суперкомпьютеров мира явля-

ется далеко не полным в силу того, что многие объекты ЭВТ этого класса засекречены.

Кроме того, очерченный нами круг объектов ЭВТ является во всех отношениях репрезентативным, поскольку и по своему объему, и по своим характеристикам отражает те тенденции, которые происходят в области высокопроизводительных вычислений, как в военных, так и в гражданских проектах США. Например, суммарная пиковая производительность 22 суперкомпьютеров, установленных в 4 вычислительных центрах коллективного пользования сети DREN, составляла на начало 2002 г. почти 12 Тфлоп или 22% от общей производительности всех суперкомпьютеров США. Номенклатура фирм производителей, собиравших суперкомпьютеры для этих вычислительных центров, также говорит сама за себя – IBM, Cray, SGI, Compaq, Sun. Вместе с тем, полученные в ходе исследований данные с достаточной степенью надежности можно будет распространить и на другие объекты ЭВТ данного класса.

Итак, ограничив область и круг объектов исследования, перейдем непосредственно к тем вопросам, которые больше всего сейчас волнуют экономистов и бизнесменов, ученых и инженеров, политиков и военных, словом тех людей, кто так или иначе заинтересован в суперкомпьютерах и их применении в конкретных проектах. А вопросы более чем очевидны: какой суммарной пиковой производительностью, оперативной и внешней памятью будут обладать суперкомпьютеры той или иной страны через несколько лет и за счет чего?

Следует заметить, что поставленные вопросы далеко не так банальны с точки зрения ответов на них, поскольку прогноз, который нам предстоит дать, носит, во-первых, стратегический, а, во-вторых, вероятностный характер.

Вначале рассмотрим, в каких пределах изменяются основные интересующие нас характеристики (оперативная память, количество процессоров, пиковая производительность) суперкомпьютеров с течением времени. На диаграммах размаха или так называемых графиках «ящики-усы» представлены диапазоны вариации выбранных нами переменных, которые построены отдельно по годам. В качестве основных показателей вариации выбраны следующие описательные статистики: минимальное и максимальное значения переменной, нижняя (25%) и верхняя (75%) квартили, медиана распределения. Медиана и квартили делят диапазон значений переменной на четыре равные части, показывая тем самым, где находятся 25%, 50% и 75% значений переменной.

Анализ диаграммы (Рис 3.) показывает, что к началу 2002 г. произошло резкое увеличение не только максимального значения пиковой производительности (почти в 2 раза), но и верхней квартили диапазона, в который попадают 75% всех значений производительности (в 7 раз) и соответственно медианы (в 5 раз). Заметим, что на протяжении предшествующих трех лет с 1998 по 2000 гг. значения этих показателей вариации оставались практически на одном уровне, хотя максимальное значение диапазона непрерывно увеличивалось.

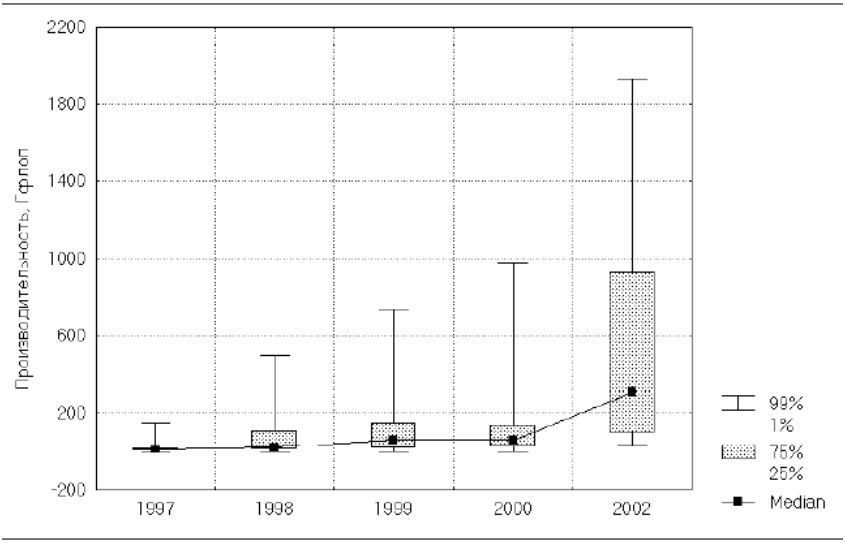


Рис. 3. Диаграмма размаха значений пиковой производительности суперкомпьютеров вычислительных центров коллективного пользования сети DREN по годам

Суммируя вышеизложенное, можно утверждать, что произошедшие к началу 2002 г. изменения в пиковой производительности отражают не только количественные, но и качественные перемены в сфере высокопроизводительных вычислений, которые были подготовлены в предшествующие три года. Ниже более подробно рассматривается этот аспект, посредством использования других методов анализа.

Существенные изменения в увеличении производительности суперкомпьютеров ВЦКП сети DREN осуществлялись в основном за счет наращивания объема оперативной памяти и количества процессоров. Однако при этом следует иметь ввиду, что имеются свои нюансы, связанные с архитектурой компьютеров.

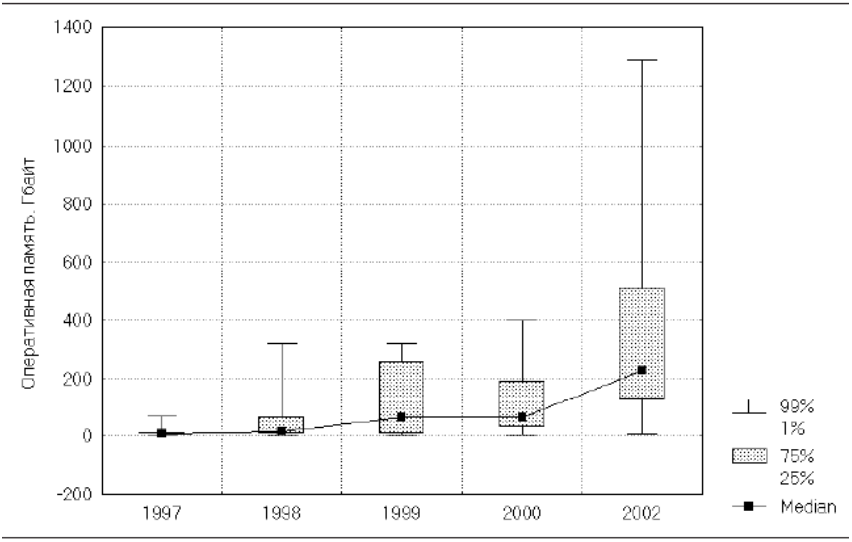


Рис. 4. Диаграмма размаха значений емкости оперативной памяти суперкомпьютеров вычислительных центров коллективного пользования сети DREN по годам

Так, например, диаграмма (рис. 4) размаха значений емкости оперативной памяти наглядно демонстрирует путь выбора архитектуры, для которого характерны колебания (взлеты и падения) всех выбранных нами статистик, в то время как на диаграмме размаха значений количества процессоров отчетливо (рис. 5) просматривается плановое начало – неуклонный рост.

Таким образом, на примере диаграмм размаха мы наглядно увидели, что достигнутый резкий скачок в производительности суперкомпьютеров ВЦКП сети DREN на рубеже конца 2001 и начала 2002 гг. стал результатом многолетних совместных усилий ученых, конструкторов и инженеров, плоды работы которых привели к наращиванию потенциальных возможностей в области высокопроизводительных вычислений (рис. 6).

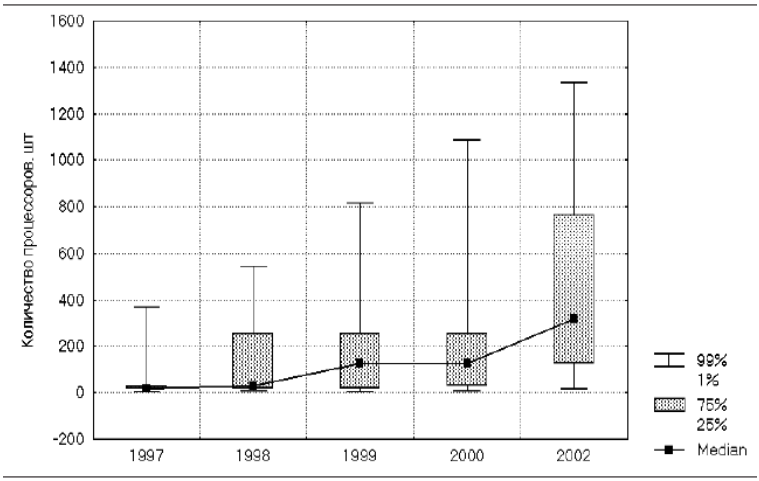


Рис. 5. Диаграмма размаха значений количества процессоров суперкомпьютеров вычислительных центров коллективного пользования DREN по годам

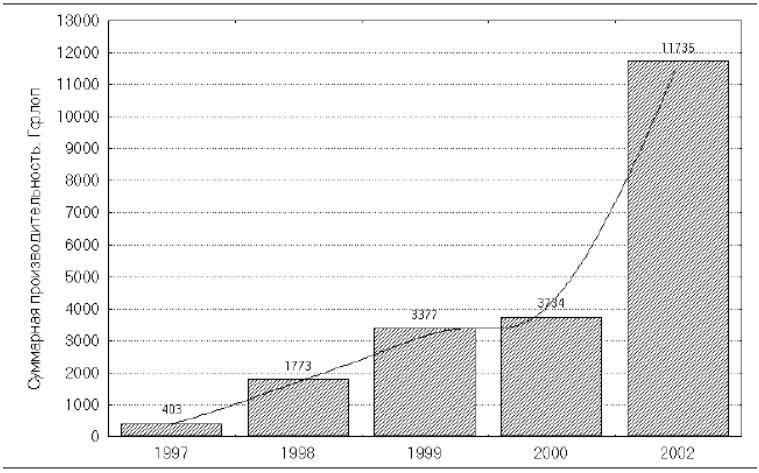


Рис. 6. Динамика увеличения суммарной производительности суперкомпьютеров вычислительных центров коллективного пользования сети DREN по годам

Далее рассмотрим, как меняется зависимость пиковой производительности от объема оперативной памяти и количества процессоров в вычислительных системах по годам. С этой целью были построены соответствующие регрессионные модели, оценим их адекватность и достоверность. Таким образом, для проведения регрессионного анализа в качестве *зависимой* переменной выберем значение пиковой производительности, а в качестве *независимых (предикторов)* – значения объема оперативной памяти и количества процессоров.

Полученные с помощью автоматизированной системы анализа данных STATISTICA линейные регрессионные модели представлены на рис. 7 и 8.

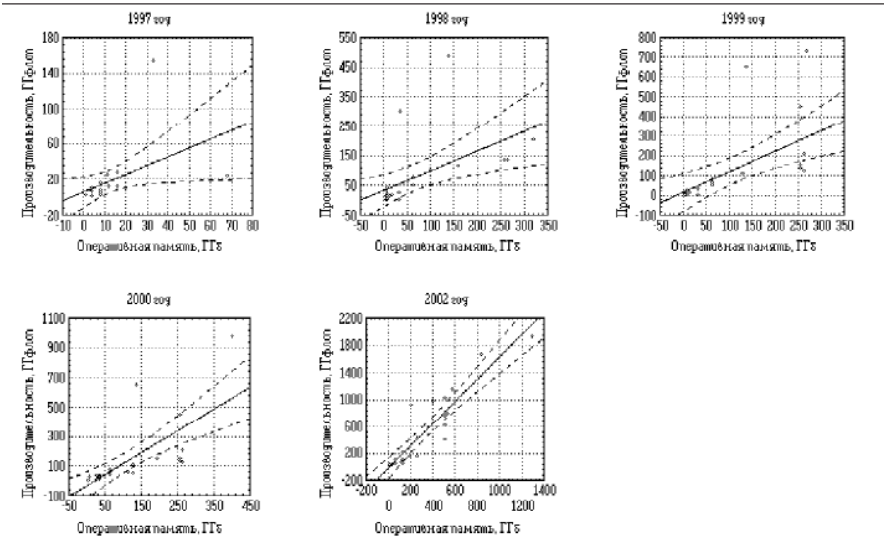


Рис. 7. Диаграммы рассеивания пиковых значения значений производительности в зависимости от объема оперативной памяти по годам

На диаграмме (рис. 7) представлены пиковые значения производительности в зависимости от объема оперативной памяти по годам, а также регрессионные прямые и их 95% доверительные интервалы. Анализ диаграмм показывает снижение разброса значений и увеличение коэффициента наклона регрессионной прямой, что дает основания говорить не о простом наращивании объема оперативной памяти, а о качественном улучшении архитектуры компьютеров. Иными словами, отдача от капиталовложений в

дорогостоящую оперативную память становится эффективной с точки зрения увеличения производительности. Аналогичная картина наблюдается и на регрессионных прямых, построенных для определения зависимости производительности от количества процессоров (рис.8), хотя они имеют и свои особенности.

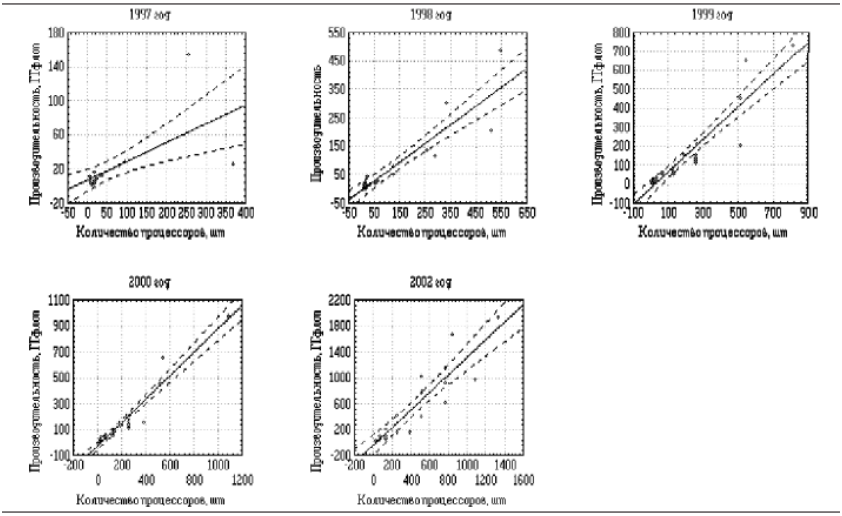


Рис. 8. Диаграммы рассеивания пиковых значений производительности в зависимости от количества процессоров по годам

Диаграммы рассеивания и регрессионные прямые, построенные для значений производительности в зависимости от количества процессоров в целом дают основания говорить о тех же тенденциях – уменьшении разброса и увеличении коэффициента наклона регрессионной прямой. Однако при этом видно что, начиная с 1999 г. наклон регрессионной прямой практически не меняется или изменяется незначительно. Это свидетельствует о том, что простое наращивание количества процессоров без увеличения оперативной памяти является эффективным только в ограниченных пределах.

Для того, чтобы убедиться в этом обратимся к анализу коэффициентов корреляции производительности с объемом оперативной памяти и количеством процессоров. В статистике корреляция определяет степень, с которой значения двух переменных пропорциональны друг другу. Диаграмма изме-

нения значений указанных коэффициентов корреляции по годам представлена на рис. 9.

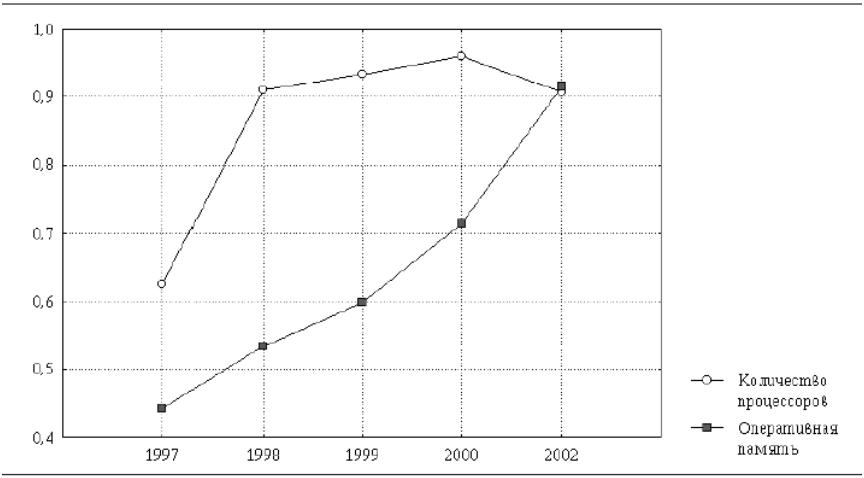


Рис. 9. Динамики изменения коэффициентов линейной парной корреляции пиковой производительности с объемом оперативной памяти и количеством процессоров

Анализ диаграммы показывает, что увеличение степени связи между производительностью и объемом оперативной памятью в указанный период было более ярко выраженной тенденцией по сравнению с ростом количества процессоров. В тоже время из диаграммы видно, что к началу 2002 гг. связь между производительностью и этими основными характеристиками архитектуры стала практически равноценной, что подтверждает достигнутый качественный рост эффективности суперкомпьютеров.

На основе нелинейной аппроксимации промежуточных значений пиковой производительности ВЦКП сети DREN и их последующей экстраполяции можно построить прогнозный тренд, достоверность которого составляет на ближайшие 5 лет 97%.

Анализ полученного прогнозного тренда изменения суммарной пиковой производительности ВЦКП сети DREN показывает, что можно ожидать повышение общей вычислительной мощности ресурсов суперкомпьютеров, входящих в их состав, с 12 до 43 Тфлоп или почти в 4 раза, что вполне согласуется с известным законом Мура. Интересно и то, что полученная прогнозная оценка не противоречит принятым еще администрацией Клинтона

в 1996 г. нормативным цифрам плана программы развития высокопроизводительных вычислений в интересах безопасного хранения ядерных вооружений, согласно которым к 2004 году предполагалось достичь производительности супер-ЭВМ в 100 Тфлоп.

4. ПРОЕКТ TOP 500

Проект TOP500 был начат в 1993 году, чтобы обеспечить надежный базис для отслеживания и обнаружения тенденций в области высокопроизводительных систем. Два раза в год собирается и выпускается список 500 самых мощных компьютерных систем в мире. Список содержит большое количество информации, включая спецификации систем и главные области их приложений. Информация во всех 30 списках TOP500, выпущенных до настоящего времени, доступна на сайте, `www.top500.org`.

С 1986 до 1992 в Университете Мангейма участникам суперкомпьютерных семинаров были представлены суперкомпьютерные статистические данные, был замечен интерес к данной области, который рос из года в год. Были представлены векторные компьютерные системы, установленные в США, Японии и Европе. Подсчет количества векторных компьютеров, установленных в мире; прежде всего зависел от данных, предоставленных изготовителями систем, и делал статистику ненадежной. Системы, существовавшие в США и Европе были достаточно хорошо известны, информацию относительно систем в Японии собрать было намного сложнее. Создатели проекта установили связь с тремя японскими изготовителями векторных компьютеров – Fujitsu, NEC и Hitachi – и использовали их данные для ежегодных оценок.

В 1992 была выпущена последняя статистика Мангейма, в которой были представлены 530 суперкомпьютеров со всего мира. Хотя на практике суперкомпьютерные статистические данные Мангейма были не очень полезны, так как им недоставало надежности данных, но в итоге, так называемые векторные компьютерные системы, такие как VP30/50 Fujitsu, стали более популярными в Японии благодаря статистике Мангейма.

Но являлись ли эти системы действительно суперкомпьютерами? Как должен быть оценен мини-суперкомпьютер, такой как Convex C1/2 из США? Необходимо было дать точное описание метода, как оценить и распознать суперкомпьютер для того, чтобы внести его в список Мангейма и обновлять его каждый год. С начала 90-х векторные компьютеры больше не были единственной суперкомпьютерной архитектурой; в широко распро-

странялись и выходили на рынок параллельные системы, такие как Интеллектуальные Машины CM2 (TMC).

Это послужило основной причиной для того, чтобы Hans Werner Meuer и Erich Strohmaier начали проект TOP500 в Университете Mannheim/Germany весной 1993. Они руководствовались следующими руководящими принципами:

- предоставлять отчет о 500 самых сильных компьютеров в мире;
- производить оценку производительности с помощью системы Linpack;
- список TOP500 обновлять и издавать два раза в год;

Все данные TOP500 открыты и доступны на сайте `www.top500.org`.

Возникает ряд вопросов:

- Почему именно «500 самых мощных компьютеров»? Одна причина – то, что в последний раз, когда были посчитаны суперкомпьютеры во всем мире в 1992, их количество было 530. И другая причина конечно – (эмоциональное) влияние 500 списков Forbes, например 500 самых богатых людей или 500 самых больших корпораций в мире.
- Почему «самый мощный» определен общей оценкой эффективности, для которой выбрана система Linpack? Потому, что данные Linpack, прежде всего максимальная производительность, известны и легко доступны для всех рассматриваемых систем. Строго говоря, TOP500 перечисляет компьютеры только по их способности решить систему линейных уравнений, Ax = b, используя произвольную плотную матрицу A.

В 7-й суперкомпьютерной статистике Мангейма, изданной на Суперкомпьютерном Семинаре Мангейма в 1992, в первом списке TOP500 ожидали гонку между США и Японией. Однако «японская опасность» была чрезвычайно завышена, поскольку первый список TOP500 показал ясно, что ведущими являются США с 45% всех установок TOP500, а Япония была далеко позади и имела всего лишь 22%. Если посмотреть на 30-ый список TOP500, изданный в ноябре 2007 в Рено/США, увидим что господство США еще больше чем было 15 лет назад: теперь у них 56.6% всех установленных систем, а Япония имеет только 4%. Даже Великобритания с 9.6% и

Германия с 6.2% оказались перед Японией, которая находится рядом с Францией 3.4%.

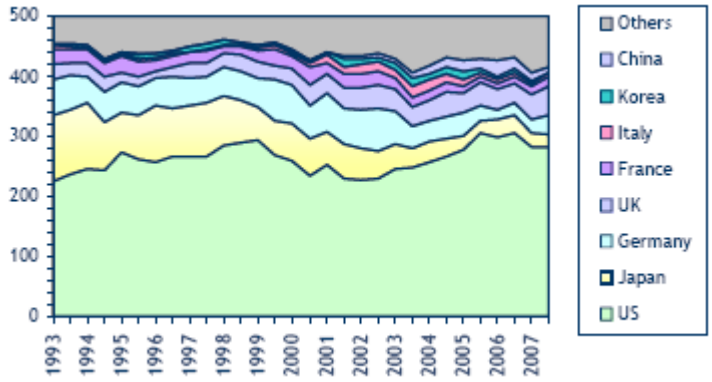


Рис. 10. Суперкомпьютерные установки в мире

В 1993, США начали с огромной цифры 45 %, которые им даже удалось немного повысить. Япония стартовала с 22%-в, но значительно отступила. В Европе – Германия, которая всегда была перед Великобританией, сейчас далеко позади.

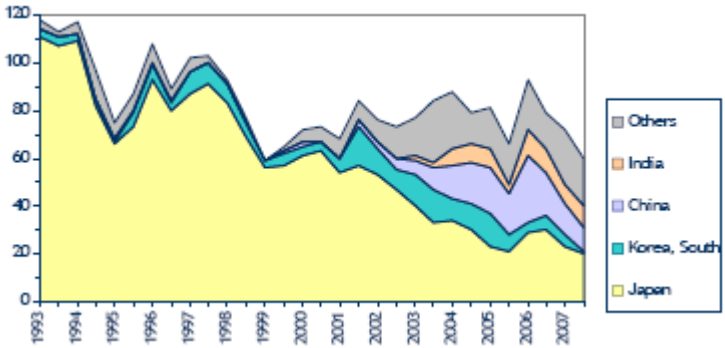


Рис. 11. Суперкомпьютерные установки в Азии

Рис. 11 показывает развитие суперкомпьютерных установок в Азии с 1993. Там виден быстрый спад в Японии, и видно, что Китай и Индия выходят на рынок HPC. Но на следующих списках TOP500 они уже теряют свои места.

Если сосредоточиться на развитии изготовителей (рис. 12), Cray был явным лидером в первом списке TOP500 с 41% перед Fujitsu с 14 %. Третье место было предоставлено TMC – невекторному суперкомпьютерному изготовителю – с 10.8 %, перед Intel с 8.8 %. Тогда у Intel все еще было свое Суперкомпьютерное Подразделение, которое также производило невекторные суперкомпьютеры. Удивительно, продвижение изготовителей HPC, IBM и Hewlett Packard, их не было представлено в первом списке TOP500 вообще. В 30-ом списке TOP500 в ноябре 2007 у IBM есть явное преимущество, а именно, 46.4 %. Второе положение принадлежит Hewlett Packard с 33.2 %, и лидер 1993 Cray Research (теперь Cray Inc) потерял позиции до 2.8 %.

1st TOP500 List, 06/1993

Manufacturer	*Count*	*Share*
Cray Research	205	41.0%
Fujitsu	69	13.8%
Thinking Machines	54	10.8%
Intel	44	8.8%
Convex	36	7.2%
NEC	32	6.4%
Kendall Square Res.	21	4.2%
MasPar	18	3.6%
Meiko	9	1.8%
Hitachi	6	1.2%
Parsytec	3	0.6%
nCube	3	0.6%
Total	**500**	**100.0%**

30th TOP500 List, 11/2007

Manufacturer	*Count*	*Share*
Cray Inc.	14	2.8%
Fujitsu	3	0.6%
Thinking Machines	–	–
Intel	1	0.2%
Hewlett-Packard	166	33.2%
NEC	2	0.4%
Kendall Square Res.	–	–
MasPar	–	–
Meiko	–	–
Hitachi/Fujitsu	1	0.2%
Parsytec	–	–
nCube	–	–
IBM	232	46.4%
SGI	22	4.4%
Dell	24	4.8%
Others	35	7.0%
Total	**500**	**100.0%**

Рис. 12. TOP 500 списки производителей

Из рис. 12 видно, что рынок HPC является очень динамичным: всего за 15 лет рынок претерпел полное преобразование. Cray потерял статус лидера на общем рынке HPC, от индустриального сегмента рынка, перешел в нишу разработок для высококачественных правительственных научно-исследовательских лабораторий и академических клиентов. IBM, с другой

стороны, которая фактически не имела достаточно важного сегмента на рынке HPC в начале 90-х, стала доминирующим лидером рынка во всех сегментах рынка, включая индустриальных и коммерческих клиентов. Hewlett Packard сначала был небольшим изготовителем HPC, представленным в первых списках TOP500, следующим за Convex, а теперь утвердился как номер два, сразу за IBM.

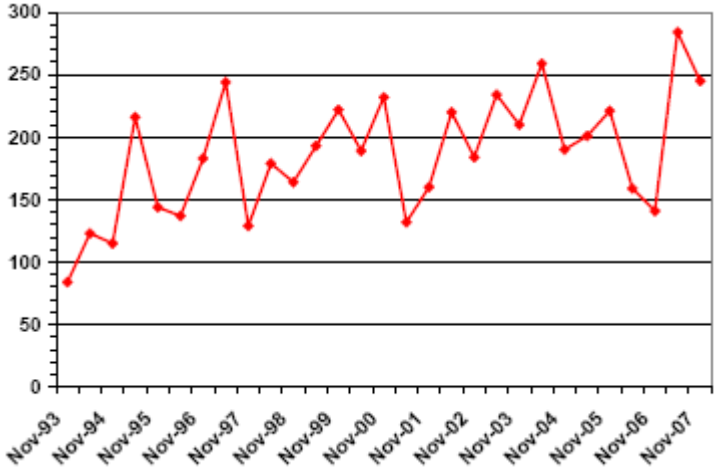


Рис. 13. Интенсивность замещения

Компания Sun Microsystems была второй среди производителей HPC, а теперь серьезно уступила свои позиции в списке TOP500. В принципе, Cray мог бы снова более активно действовать на арене HPC. Компания в настоящее время имеет три гибридных суперкомпьютера, находящихся в списке TOP10, который показывает, что она успешно развивает векторные вычисления.

Проект TOP500 компенсировал дефицит суперкомпьютерной статистики Мангейма, которую использовали в течение семи лет на конференциях ISC 1986–1992. Используется простой, но успешный подход, основанный на оценке эффективности с помощью Linpack, хотя она часто критикуется. Посредством списка TOP500 невозможно оценить размер рынка HPC (например, в US$), поскольку мы не знаем сколько систем и по какой стоимости производится и реализуется. Информация о ценах систем представляла бы большой интерес для сообщества HPC. Но в самом начале проекта

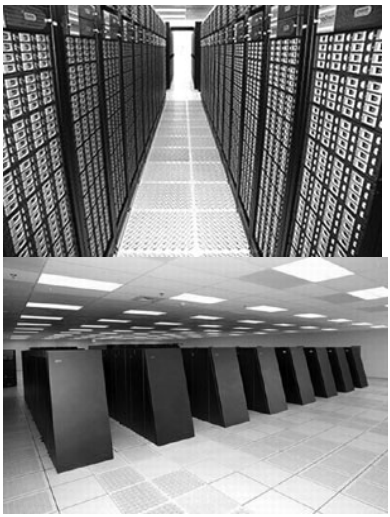
TOP500 предполагалось не включать этот вид более или менее ненадежных и неопределенных данных.

Анализируя списки TOP500, видно, что системы верхней половины списка остаются там только в течение некоторых периодов, и видны сезонные колебания. Есть случаи, когда системы находятся в списке в течение только шести месяцев, и в этот период товарооборот очень высок. Рис. 13 показывает, что в среднем, приблизительно 200 систем выбывают после шести месяцев.

5. BLUE GENE – АТЛАНТ МИРА ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

На фоне «гонки вычислительных мощностей» весьма красиво выглядит легендарный Blue Gene/L, который вот уже 2 года является “царем суперкомпьютеров”. Blue Gene/L разработан компанией IBM, а его мощность – 280,6 терафлоп (или триллионов операций с плавающей запятой в секунду) – почти в 3 раза превосходит мощность машины, занявшей в ноябре 2006 г. второе место в списке Top 500.

В 1999 г. компания IBM объявила о развертывании программы, результатом которой должен стать суперкомпьютер с производительностью в один петафлопс (тысяча терафлоп) [13, 14]. В то время десятки терафлоп были сказкой, что уж говорить о тысячах... Так вот, проект Blue Gene (голубой ген) – это один из первых шагов на пути к достижению заветной цели. Новая машина должна была затмить славу Deep Blue, суперкомпьютера обыгравшего Каспарова в шахматы. (Производительность Deep Blue чуть больше 1 терафлопа.) О строительстве новой системы, которое должно производиться в Ливерморской национальной лаборатории имени Лоуренса (США), было объявлено в сентябре 2000 г., а уже в ноябре 2004 г. она заняла первое место в рейтинге Top 500. Ее производительность тогда достигала 70,7 терафлоп, а в составе системы было 16 384 двуядерных процессора (поэтому



число процессоров умножалось на два, и в прессе фигурировала цифра 32 768). Затем, в 2005 г., количество процессоров Blue Gene/L удвоилось (65 536) и производительность повысилась почти в 2 раза (135,5 терафлоп). И наконец в 2006 г. число процессорных блоков было еще раз увеличено вдвое, а производительность достигла небывалого доселе показателя в 280,6 терафлоп.

Каким образом, возможно, измерить производительность суперкомпьютеров? Для этого используется тест Linpack, который «кормит машину» громоздкими системами линейных уравнений с множеством неизвестных. При проведении такого теста вычислениями занимаются все блоки суперкомпьютера. Необходимо заметить, что при решении реальных задач некоторые блоки заняты организацией распределения задач и коммуникаций в системе.

Для чего же нужны такие мощности? Неужели те вычислительные системы, которые существуют, не удовлетворяют потребностей науки и техники? Оказывается, что нет. Изначально Blue Gene/L, соответственно своему названию, должен был моделировать строительство белков на основании информации, хранящейся в ДНК. Эти исследования могли помочь ученым в борьбе с некоторыми болезнями. Но, как это всегда случается, проблема национальной безопасности ставится выше проблемы здоровья населения. Поэтому американские оборонщики уже в 2001 г. «переманили» многообещающую, но еще не построенную машину к себе, чтобы использовать ее для исследования ядерного оружия.

Сегодня в серии Blue Gene есть несколько машин, построенных по одному принципу. Одна из них занимается моделированием процессов, происходящих в мозге человека. Так что медицина и генетика не остались на задворках.

В 2006 г. оборонщики Соединенных Штатов заключили контракт с IBM на строительство вычислительной системы для моделирования глобальной биосистемы нашей планеты. Производительность этой машины будет не менее 1 петафлопа. Вероятно, IBM разработает новую архитектуру, хотя Blue Gene все еще обладает довольно внушительным потенциалом. Впрочем, результаты этого проекта будут известны лишь в 2010 г., поэтому Blue Gene/L пока что «царь в мире суперкомпьютеров», и, по-видимому, останется им еще какое-то время.

Blue Gene/L, как и многие суперкомпьютеры, имеет кластерную архитектуру. Суть ее состоит в том, чтобы объединить множество процессоров в единую систему и тем самым увеличить ее производительность. На сегодня этот принцип строительства суперкомпьютеров, пожалуй, самый эко-

номичный и удобный. Несмотря на это, у него есть и некоторые недостатки. Например, при использовании множества кластеров значительно увеличивается энергопотребление вычислительной системы, кроме того, она требует довольно мощного охлаждения. И конечно, невысокие скорости обмена данных с памятью снижают мощность системы, поэтому потенциал процессоров не используется полностью.

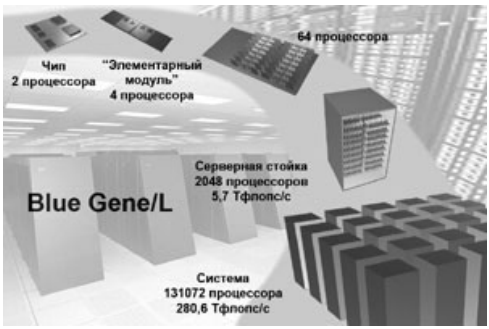
Архитектура Blue Gene/L весьма экономична. Блоки, где размещены процессор, кэш-память, оперативная память, сетевой интерфейс, обладают относительно небольшой производительностью, приравненной к скорости обмена данными. Это обеспечивает рациональное использование процессорных мощностей с минимальными потерями, а также небольшое энергопотребление. Сама параллельная архитектура суперкомпьютера хорошо организована при помощи отдельных блоков, количество которых можно регулировать в зависимости от потребности задач, решаемых системой.

«Элементарной частицей» суперкомпьютера Blue Gene/L является модуль compute card, состоящий из двух блоков, которые, в свою очередь, обладают двумя процессорами. Такие модули объединяются по 16 штук и размещаются на особой карте. В свою очередь, 16 карт также объединяются и располагаются на одной панели (ее размеры 43,18x60,96x86,36 см). Получается, что в каждой такой панели 512 блоков. Затем две панели объединяются и встраиваются в серверную стойку (в такой стойке уже 1024 блока и 2048 процессоров).

На каждом модуле compute card размещаются два процессора и по 4 Mb выделенной памяти. Здесь же располагается контроллер и девять микросхем оперативной памяти. Общий объем «оперативки» – 256 Mb, хотя теоретически модуль может поддерживать и 2 Gb памяти. Каждый модуль физически представляет собой синтез микросхем оперативной памяти SDRAM-DDR и специальной интегральной микросхемы ASIC (Application-Specific Integrated Circuit). ASIC стоит рассмотреть подробнее, так как многие достоинства Blue Gene/L определяются ею. Схема представляет собой так называемую «систему-на-чипе», включающую в себя помимо процессора сетевые интерфейсы и встроенную память. Размеры модуля достаточно малы (габариты подложки не больше 11,1x11,1 мм). Микросхема ASIC работает на частоте 700 MHz. Размеры и компактность ASIC позволили снизить энергопотребление и значительно увеличить вычислительную плотность по отношению к объемам машины.

Модули Blue Gene/L оснащены процессорами PowerPC 440. Одна панель, объединяющая 512 элементарных блоков системы, обладает производительностью в 1,4 терафлоп при условии, что на каждом узле установлен

один процессор. Но в каждом блоке присутствует еще один процессор, выполняющий функции связи с другими процессорами всей системы. Есть еще и особые процессорные узлы, которые занимаются сугубо вводом-выводом информации. Для нормальной работы всей системы необходимы хост-компьютер (он занимается анализом и компиляцией данных) и файл-серверы. Блок ввода-вывода является этаким посредником между вычислительными блоками и хост-компьютером, а также файл-серверами.



Все вычислительные и иные блоки системы связаны хитросплетением сетей. Сеть Gigabit Ethernet обеспечивает подключение к хост-компьютеру и файловым серверам. Сеть Gigabit Ethernet-JTAC предназначена для управления машиной. Кроме этого, для обмена информацией между разными блоками строится трехмерная тороидальная сеть; для выполнения общих операций – «сеть-дерево», а также присутствует сеть барьеров и прерываний, общая для всей системы. Все устройство вычислительной системы довольно компактно (в сравнении с конкурентами) и экономично. Разработчики создали хорошую систему охлаждения, состоящую из радиаторов с наклонными ребрами, благодаря которым охлажденный воздух распределяется более рационально.

Blue Gene/L работает под управлением специально разработанной версии операционной системы Linux. Благодаря тому, что программный код Linux открыт, задача создания программного обеспечения значительно упрощается и удешевляется.

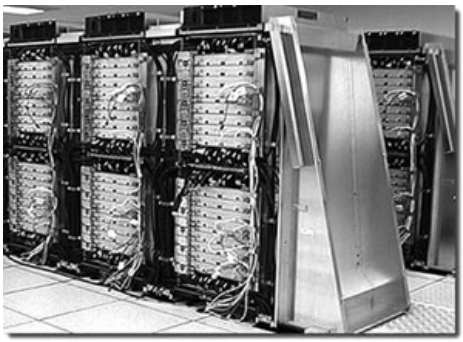
6. ПЕРСПЕКТИВЫ ТЕХНОЛОГИИ BLUE GENE

Министерство энергетики Соединенных Штатов купило Blue Gene/L за $290 млн. Для дальнейшего развития суперкомпьютерных технологий IBM начала продажу машин типа Blue Gene/L. Стоимость одной стойки (ее производительность – 5,7 терафлоп) составляет $2 млн. Некоторые научные учреждения уже оснащаются системой Blue Gene, в их числе Суперкомпьютерный центр Сан-Диего и Эдинбургский университет. В целом супер-

компьютеры, построенные на основе технологий Blue Gene, работают в разных частях света и решают различные по своему направлению задачи. Они задействованы как в сфере военной безопасности и изучении ядерных потенциалов, так и в исследовании биомолекулярных формаций, ДНК и в разработке тех или иных лекарств.

Помимо этого, IBM может предоставить суперкомпьютер, установленный в Watson Research Center, для решения каких-либо задач (естественно, за определенную плату).

Производительность этой системы – 100 терафлоп. Арендовать систему (или приобрести вычислительное время) могут как научные исследователи, так и бизнес деятели. IBM планирует внедрять суперкомпьютеры не только в научные, исследовательские центры и государственные учреждения, но и в коммерческие организации. Blue Gene – в этом отношении очень конкурентоспособная технология, так как она относительно дешева, компактна, а сотовая структура позволяет организовать конфигурацию, необходимую для решения конкретных задач с достаточной (но не избыточной) мощностью. Если произойдет массовое внедрение суперкомпьютеров в коммерческие организации, появится более широкое финансовое основание для дальнейшего развития в данной области, поэтому скорость развития высокопроизводительных машин также возрастет.



Система Blue Gene/L была способна к развитию, но IBM планировала в рамках серии Blue Gene построить еще несколько машин. Например, Blue Gene/C (C означает Cyclops64). А система BlueGene/P, релиз которой планировался выпустить в 2008 г., должна была достигнуть заветного предела в 1 петафлоп.

Реально всё произошло значительно быстрее. А именно, 26 июня 2007 года IBM представила Blue Gene/P, второе поколение суперкомпьютеров Blue Gene. То есть был разработан компьютер для постоянной работы с производительностью в 1 петафлоп. Blue Gene/P может быть сконфигурирован для достижения более чем 3 петафлоп.

Следующая система с производительностью в 3-10 петафлоп называется Blue Gene/Q. На данный момент этот проект – скорее мечта, чем реаль-

ность. И все-таки, если человечество достигло всего того, что мы уже имеем (а ведь и терафлоп когда-то был чем-то нереальным), то можно с уверенностью смотреть в будущее.

СПИСОК ЛИТЕРАТУРЫ

1. TOP500 Supercomputer Sites – мировой рейтинг пятисот самых мощных компьютеров мир. – http://www.top500.org/
2. Воеводин В.В., Воеводин Вл. В. Параллельные вычисления – СПб: БХВ-Петербург, 2002. – 609 с.
3. Message Passing Interface Forum. – http://www.mpi-forum.org/
4. Андрианов А.Н., Бабенко К.И., Забродин А.В., Задыхайло И.Б., Котов Е.И., Мямлин А.Н., Поддерюгина Н.В., Поздняков Л.А. О структуре вычислителя для решения задач обтекания. Комплексный подход к проектированию // Вычислительные процессы и системы. – М.: Наука, 1985. – С. 13–62.
5. Cray Research, Inc. CAL Assembler Version 1 Reference Manual. – Publ. 2240000, Bloomington, Minnesota. – 1977.
6. Cray Research, Inc. Computer System Hardware Reference Manual. – Publ. 2240004, Bloomington, Minnesota. – 1979.
7. Суперкомпьютеры Facom фирмы Fujitsu выходят на рынки США и Западной Европы // Электроника. – 1984. – Т. 57, № 20. – С. 28–29.
8. Towards Supermachines: NEC's Supermachine SX-2 with Operating Speeds of 1,3 Gigaflops // Data Tid, Norway. – 1984. – Vol. 6, № 1. – P. 132–135.
9. Shaw D.E. The NON-VON Project // Progress Report, Columbia Univ. – 1983.
10. Gottlieb A., Grishman R., Kruskal C., McAuliffe K., Rudolph L., Snir M. The NYU Supercomputer – Designing and MIMD Shared Memory Parallel Computer // IEEE Trans. Comput. – 1983. – Vol. C32, № 2. – P. 175–189.
11. Arvind, Dertouzos M.L., Ianucci R.A. A Multiprocessor Evaluation Facility // MIT/LCS/TR-32. Cambridge, Massachusetts. – 1983.
12. Arvind, Culler D.E., Ianucci R.A., Kathail V., Pingali K., Thomas R.E. The Tagged Token Dataflow Architecture. – MIT. Cambridge, Massachusetts. . – 1983. – (Prep.).
13. Blue Gene – IBM Research Project // http://domino.research.ibm.com/comm/research_projects.nsf/pages/bluegene.index.html
14. The IBM Blue Gene/P Solution. – http://www-03.ibm.com/systems/deepcomputing/bluegene/

Г.П. Несговорова

ПОСОБИЕ ПО НАПИСАНИЮ РАЗНОГО РОДА ДЕЛОВЫХ ТЕКСТОВ

(в помощь студентам-программистам, информатикам, математикам, а также студентам других специальностей и всем интересующимся)

I. СТИЛИСТИКА ДЕЛОВЫХ ТЕКСТОВ

Введение

Научным сотрудникам, инженерам и людям других творческих специальностей в своей профессиональной деятельности не обойтись без оформления ряда документов, таких как отчеты, статьи, разного рода описания, тексты монографий, диссертаций, авторефератов, деловые письма, запросы и т.д. При этом надо учитывать особенности деловой письменной речи, о которых и будет рассказано ниже.

Если для художественной прозы характерна многозначность, экспрессивность, эмоциональность, абстрактность, ей чужда конкретность, а присуще образное, переносное употребление слов, то деловым текстам свойственна конкретность, детальность и однозначность, четкое логическое единство и последовательность. При этом в деловых текстах используются развернутые синтаксические конструкции и точное употребление слов.

Но многие ли знают, чем отличаются деловые тексты от разговорной речи или художественной прозы? Для англоговорящих людей существует специальный курс Business English, в котором рассказывается об особенностях делового английского языка, объясняется, какие конструкции или выражения неприемлемы в деловых текстах. Есть мнение, что русские люди благодаря особенностям менталитета подсознательно настроены на нужный стиль, но жизнь подсказывает, что это далеко не всегда так, и следовало бы изучить особенности деловой разновидности родного русского языка, с которой мы сталкиваемся в разных жизненных ситуациях довольно часто.

1. Понятие информационной стилистики

Без хорошего знания языка общения (в нашем случае русского) как в быту, так и в профессиональной деятельности сложно добиться взаимопонимания, реальных результатов, овладеть изучаемым предметом и выбранной профессией, стать квалифицированным специалистом. В предлагаемом пособии за основу взят не традиционный курс русского языка, который изучается в средней школе, а специальный – ориентированный в основном на математиков, программистов, информатиков и людей других аналогичных специальностей с учетом особенностей их профессиональной деятельности и лексики соответствующей предметной области.

Эта ориентация объясняется еще и тем, что люди, избравшие объектом своего изучения точные дисциплины, часто бывают не в ладах с гуманитарными, к которым относится и русский язык. Знание последнего, в частности, так необходимо не только при написании статей и отчетов, но и спецификаций, инструкций по применению программ и другой научной документации. Для профессиональных программистов, например, отсутствие навыков составления такого рода документации может в целом значительно снизить качество и конкурентноспособность создаваемой ими программной продукции.

Исходя из всего вышесказанного, отметим, что при составлении различного рода документации используются несколько иные речевые нормы, чем в литературном или бытовом языке. Составителю документации необходимо стремиться точно и однозначно передать суть проделанной им работы. При этом лексика должна быть достаточно богатой, но лишенной синтаксической и лексической омонимии, во избежание двусмысленности. Научная дисциплина, которая устанавливает речевые нормы при написании разного рода документации, изучает письменную форму официально-делового общения с целью ее унификации и нормализации, называется документной лингвистикой. В своей основе документная лингвистика – это стилистика деловой речи. В тех случаях, когда говорится о программных системах, их разработке и применении или о моделировании с их помощью различных реальных систем, предлагается использовать новый термин – информационная стилистика. Что же вкладывается в это понятие?

Стилистика – это наука о ресурсах языка, словоупотреблении, выразительных возможностях речи [1]. Письменный текст рассматривается как разновидность речевого акта и является результатом выбора языковых форм из имеющихся в языке возможностей и их комбинаций в зависимости от его функции. В разных лингвистических школах понятие «стилисти-

ка» толкуется по-разному, но, как правило, всегда выделяются три основных раздела.

1. Лексико-грамматическая стилистика, или стилистика ресурсов, тесно связана с культурой речи и нормативной грамматикой. В этом разделе изучаются варианты слов и форм слов, выражений, словосочетаний, предложений.

2. Функциональная стилистика рассматривает вопрос о системе стилей русского языка, различающихся:

а) целью высказывания;

б) сферой применения;

в) специфическими способами использования и организации языковых средств, а иногда и их особым подбором.

3. Стилистика текста основное внимание уделяет анализу текста. Она изучает особенности и закономерности строения конкретного текста в связи с предметом и обстоятельством речи.

Вводимое здесь понятие информационной стилистики не противоречит данному выше толкованию, а лишь модифицирует, уточняет и адаптирует его к информатике как области знаний. Можно отметить следующие особенности информационной стилистики.

1. В основном используется лексика, имеющая отношение к информационной тематике и включающая общепринятые и наиболее употребляемые слова русского языка, без которых невозможно правильно построить предложение и выразить смысл любой фразы. Таким образом, мы используем все заранее данные языковые возможности, но понятие «лексичечкая возможность» сводится в основном к информационной лексике, а к ней уже применяются комбинации других языковых возможностей, а именно: грамматических, синтаксических, семантических.

2. Способы использования и организация языковых средств ориентированы на написание деловых текстов и документации. Это значит, что стиль такого рода текстов должен отличаться от литературного и утилитарного стилей строгостью и четкостью изложения с некоторыми другими ограничениями.

3. Поскольку мы имеем дело с текстами по информационной тематике и ориентируемся на обучение правильному их написанию и оформлению, то анализ текста полезно проводить именно с учебными целями, сравнивая написанный обучающимися текст с некоторым эталонным текстом.

Таким образом, основу предлагаемого толкования стилистики составляет понятие функциональной стилистики, а его разновидностью и конкретизацией и будет являться информационная стилистика.

Именно такое понимание стилистики будем использовать при обучении русскому языку, ориентированному на информационную тематику и грамотное написание документации, что включает в себя следующие основные принципы:

– достоверность и объективность содержания;
– полноту информации;
– краткость изложения, но не в ущерб смыслу;
– нейтральность тона изложения;
– средства логической оценки фактов и результатов;
– отсутствие неоднозначных толкований, т.е. омонимии на всех уровнях;
– унификацию речевых средств и стандартизацию терминологии.

Считается, что язык документации – это набор клише, штампов, стандартных терминов. В этом смысле язык документов может быть унифицирован. Унификация рассматривается как выбор одного языкового варианта из нескольких возможных способов передачи одной и той же информации и предполагает использование языковых формул – устойчивых оборотов, словосочетаний, моделей предложений, терминов и сокращений, принятых в данной области знаний. Унификация является удобным инструментом для составления документации. При этом унифицированные словесные обороты требуют минимального напряжения не только при написании документов, но и при их восприятии.

Унификация охватывает все уровни языка: графику и фонетику, словарный состав, морфологию, синтаксис, пунктуацию и стиль, но при этом не является «канцеляритом» с устоявшимися раз и навсегда, застывшими оборотами речи, а позволяет пользоваться всеми возможностями языка, не сковывая творческую инициативу составителя документации, а лишь несколько ограничивая выбор языковых норм и стиля.

Предлагаемое пособие по русскому языку предполагает не обучение ему в объеме средней школы (считается, что этот курс уже освоен), а нацелено на то, чтобы научить грамотно – стилистически и грамматически – излагать свои мысли и помочь в составлении разного рода документации высокого качества с учетом вышеизложенных требований [2].

В связи с этим полезно также ознакомиться с феноменом деловой прозы, определение которой дал академик А.П. Ершов [3].

1.1. Деловая проза

За последние годы появилось много работ по вопросам письменной речи, что свидетельствует о возросшем интересе специалистов разных профессий к языку и стилю документации.

А.П. Ершов, учитывая предельную унифицированность языка современной документации, обратил внимание на некий лингвистический феномен, который, сохраняя многие свойства естественного языка в целом и служа средством коммуникации, в то же время подготовлен для того, чтобы стать объектом автоматизации.

Этот феномен – деловая проза, которая определяется как языковой носитель производственных отношений человека. Создание информационных и программных систем является одним из таких видов производственных отношений. Таким образом, деловая проза всегда задействована в строгой модельной ситуации.

Деловая проза является подмножеством естественного языка, используемым в любой регламентированной деятельности человека. Деловой прозе свойственны жесткие средства выражения, однозначность передаваемой информации, экономность языковых средств, четкость функции каждого сообщения; она всегда внутренне формализована, даже если человек не сознает этого.

С учетом перечисленных свойств деловая проза в идеале может использоваться как инструмент в работах по созданию программных систем и стать посредником между человеком и машиной. Она, в частности, может применяться при создании интерфейса в той его части, когда текстовая информация о дальнейших действиях и разного рода подсказки высвечиваются на экране. Взаимодействие с пользователем удобно вести на языке деловой прозы, так как, с одной стороны, существует достаточная строгость и точность текста при выдаче тех или иных рекомендаций пользователю на экране, а, с другой, – сохраняются основные черты естественного языка, что создает атмосферу дружелюбного взаимодействия между машиной и человеком и снимает психологическое напряжение у последнего.

Деловая проза и унификация языка – это прежде всего установление и применение правил построения предложения по определенным моделям, соответствующим тем или иным ситуациям, к которым относятся и производственные отношения. В идеале хотелось бы свести процесс составления документации к использованию конкретных языковых моделей, на базе которых можно правильно и быстро строить предложения. При этом под моделью понимается синтаксическая конструкция предложения, охватывающая максимальное количество ситуаций, возникающих при написании документации по данной отрасли знаний.

Вопросы для самоконтроля

1. Что такое стилистика?

2. Что рассматривает функциональная стилистика?
3. Укажите особенности информационной стилистики.
4. Перечислите основные принципы информационной стилистики.
5. Что такое деловая проза?

2. Язык как средство коммуникации

Деловой текст, к которому мы будем относить не только собственно документы (приказы, сопроводительные письма, разного рода акты и пр.), но также и статьи, отчеты, тезисы, инструкции по использованию программных продуктов и компьютерных игр, спецификации и т.д., является важным составным элементом коммуникации всего современного общества. Следует определить соответствия между особенностями современных деловых текстов, перечисленных выше, и степенью участия элементов системы современного русского языка в создании и особенностях содержания и преобразования такого рода текстов.

В процессе любой деятельности (в частности, программирования) человек осуществляет процесс коммуникации (иначе – общения). Деловая коммуникация, являющаяся частью общего процесса коммуникации, – это процесс обмена информацией, осуществляемый в связи с решением разного рода производственных вопросов с использованием устной и письменной форм речи.

Документная коммуникация обозначает процесс коммуникации с использованием письменной формы речи, письменных текстов. Обязательным компонентом письменного делового текста является содержательная определенность. Под этим понимается законченность, смысловая целостность в выражении мысли, факта, доказательства. Для делового общения основным средством общения остается все же естественный человеческий язык, в нашем случае – русский.

Какие же свойства естественного языка определяют степень надежности коммуникативных процессов, для которых предназначен деловой текст? Какие трудности коммуникативных процессов существуют на разных языковых уровнях? Рассмотрим эти уровни.

1. Фонетический уровень – вариантность фонем (звучания слов), обусловленная диалектом, социальным и профессиональным жаргонами.

2. Морфологический уровень – в русском языке существует неопределенность или полисемичность (многозначие) морфологических компонентов. Например, суффикс «-ник» соотносится и со значением лица по объекту деятельности: уче**ник**, медвежат**ник**, колбас**ник**, и с устройством, вы-

полняющим определенные функции: подой**ник**, коров**ник**, курят**ник**. При этом не забываем и о других различных возможностях этого суффикса: муравей**ник**, хищ**ник**, покой**ник**.

3. Лексико-фразеологический уровень создает наибольшее количество проблем в коммуникативных процессах. К таким проблемам относится омонимия, например, лексема «коса» означает: 1) сельскохозяйственный инструмент, 2) песчаная отмель, 3) заплетенные волосы, а также полисемия: spirit по-английски переводится как «дух, алкогольный напиток, смысл, привидение, сущность» и пр. Эти проблемы высветились, например, при попытке машинного перевода текстов с одного языка на другой. При переводе с помощью компьютера устойчивого выражения английского языка: «The spirit is strong, but the flesh is weak», что означает «хотя дух силен, но плоть слаба» компьютер выдал вариант «водка держится хорошо, а мясо пропало». Полисемичность слов, составляющих английское выражение, не могла не породить ошибок в переводе, умноженных языковой комбинаторикой: spirit, кроме приведенных выше значений, переводится еще и как «душа, мнение, настроение, водка» и т.д., а flesh – как «мясо, тело, плоть, мякоть». И это еще не все значения этих слов.

С учетом того, что в русском языке такие явления, как омонимия, полисемия, паронимия (явление частичного звукового сходства при частичном или полном различии значений: «поиск – происк, здравица – здравница, скрытый – скрытный») довольно распространены, то создаются благоприятные условия для снижения коммуникативной надежности, а это особенно следует избегать в деловых текстах.

4. Синтаксический уровень – степень синтаксической сложности конструкций на естественном языке значима как при восприятии, так и при порождении фраз. Трудно понять смысл сложного, длинного предложения, ровно как и правильно построить его. На синтаксическом уровне, особенно в деловых текстах, следует также избегать синтаксической омонимии, ведущей к разным смыслам одного словосочетания, например, словосочетание «парень в зеленой рубашке с голубыми глазами» имеет два синтаксических разбора: 1) (парень в зеленой рубашке) с голубыми глазами и 2) парень (в зеленой рубашке с голубыми глазами), т. е. рисунком на зеленой рубашке при таком синтаксическом разборе являются «голубые глаза».

5. На семантическом уровне должен четко выражаться общий смысл делового текста, чтобы не допустить сематической омонимии, такой как, например, в словосочетании «портрет Тани», которое имеет минимум три значения: 1) какой-то портрет, принадлежащий Тане; 2) портрет, автором которого является Таня; 3) портрет, на котором нарисована сама Таня.

Рассматривая язык как основное средство общения, нельзя не обратиться к проблеме разграничения базисных понятий лингвистики: **язык**, **речь**, **текст**. Классическая лингвистика пытается установить характер взаимосвязей между этими важными понятиями как с содержательных, так и с функциональных позиций.

Язык создает мощную знаковую базу для дальнейшего его применения в различных формах, соответствующих данным обстоятельствам. Изменение основы языка влечет за собой изменение характера речи.

Речь является реализацией языка и в конкретных условиях приобретает конкретную форму, во многом зависящую от обстоятельств, в которых реализуется язык. Образно говоря, язык – это молоток, а речь – стук молотка.

Текст является письменной формой речи.

Когда мы имеем дело с грамотным человеком, владеющим языком во всех отношениях: морфологическом, орфоэпическом, синтаксическом и стилистическом, то в условиях коммуникации он успешно решает вопросы: «Как писать?», «О чем писать?», «Как деловой текст должен выглядеть вообще?». Это как раз те ситуации, которые надо решать в связи с различными формами официально-делового общения.

Понятие релевантности, широко используемое в информационном поиске, вполне подходит для описания модели коммуникативного процесса с помощью естественного языка. В описательной форме эта модель будет выглядеть следующим образом: от идеи сообщения – к выбору языковых средств, происходящему под воздействием внешних формальных условий коммуникации, далее – к порождению совокупности речевых вариантов, в которых (комбинаторно) средствами языка отражена мысль, затем – выбор оптимального речевого варианта, соответствующего формальным и содержательным требованиям, и наконец – текстовая реализация избранного текстового варианта, являющегося релевантным.

Поскольку исходная идея формируется с использованием средств языка, испытывая при этом воздействие условий коммуникации, то решающим становится критерий выбора языковых средств. Очевидно, что в результате могут появиться различные речевые варианты, которые в разной степени соответствуют всем требованиям данного коммуникативного процесса. Определение наиболее соответствующего ситуации и коммуникативной задаче (т.е. релевантного) процесса – дело опыта и знаний. Разная степень произвольности в выборе языковых единиц связана как с формальными требованиями, так и с языковыми ресурсами, используемыми для коммуникативных целей, а также с речевой традицией общества.

Довольно показательными в этом отношении являются тексты официально-делового стиля, для которых влияние дополнительных факторов становится столь значимым, что они не просто произвольно сочиняются, а составляются, то есть используются некоторые заготовки, относящиеся к разным языковым уровням.

Упомянутый выше термин **текст** как организованная, определенным образом зафиксированная речь очень важен, так как он является и отражением языкового материала, и представителем определенной формы коммуникации, отражающий и материалом, и оформлением определенный вид коммуникативной ситуации.

Таким образом, реализация языка в определенных коммуникативных условиях способствует формированию текста, в определенной степени отражающего условия коммуникации. Потенциальная изменяемость, вариантность зафиксированной речи (текста), соответствующей ситуации, отражает как возможности языка, так и влияние экстралингвистических факторов, т.е. реальных условий, в которых язык функционирует.

Вопросы для самоконтроля

1. Что такое деловой текст? Каковы его особенности?
2. Перечислите существующие языковые уровни.
3. Приведите примеры морфологической полисемии.
4. Что такое омонимия, парономия и полисемия на лексико-фразеологическом уровне? Проиллюстрируйте примерами.
5. Приведите примеры синтаксической и семантической омонимии.
6. Что такое язык, речь, текст? Объясните различия.

3. Дифференциация языка и функциональные стили

Общественно-языковая практика в разных сферах деятельности человека сопровождается специализацией используемого в них языка. Типичные формы языка, используемые в тех или иных типичных обстоятельствах, представляют **стили**. К сожалению, однозначного понимания стиля языка не существует, что свидетельствует о сложности и противоречивости этого понятия (см. разд. 1). Типичность, устойчивость ситуаций, многократно повторяемых в разное время и в разных пространственных координатах, создают в языке **полиформность**, при которой за типичной ситуацией закрепляется некоторая совокупность языковых средств.

Начиная разговор о стилях, принципиально важно рассмотреть такое явление, как дифференциация языка. Основной этап дифференциации – это

разделение современного русского языка на **литературный** язык и его **нелитературную** форму. Нормированность литературного языка – одно из важных его качеств. При этом нормированность распространяется на все языковые уровни. Зарождение и формирование литературного языка – это сложный и длительный процесс, который здесь мы не будем рассматривать.

Формы языка, не соответсвующие языковой норме и принадлежащие разным социальным группам со сниженной языковой компетенцией, называют **социолектами**, в отличие от региональных разновидностей одного языка, которые называют **диалектами**. Говоря о сферах функционирования языка, мы имеем в виду его использование в различных типовых ситуациях. В настоящее время количество стилей русского языка, которое выделяется современными исследователями, не является постоянной величиной.

В прикладной и математической лингвистике большое внимание уделяется даже не стилям, а подъязыкам. Подъязык – это набор языковых элементов и их отношений, характерных для определенной предметной области знаний, например, подъязыки экономики, техники, информатики, медицины и пр.

Языковой стиль при этом определяется как подсистема языковых элементов и их отношений, который отличает тексты с однотипной целевой установкой от подсистем с иными функциональными задачами. Таким образом, подъязык связан с тем, о ЧЕМ говорят, а стиль – КАК говорят.

3.1. Функциональные стили

Как упоминалось выше, количество функциональных стилей русского языка по-разному определяется разными лингвистическими школами. Остановимся на тех стилях, которые выделяются большинством исследователей-лингвистов.

Стиль художественной литературы (художественный стиль). В силу крайней индивидуальности речи каждого участника творческого художественного процесса, общие признаки этого стиля сталкиваются с противоречиями, обусловленными индивидуальными особенностями творчества и языкового использования каждого отдельного автора.

Разговорный стиль (иначе бытовой). Реализуется в устной форме. Характеризуется высокой степенью отступлений от требований нормы, зависимостью от условий коммуникации, многообразием установок и т.д. Этот стиль – один из самых неоднородных с точки зрения диапазона используемых языковых средств.

Публицистический, научный, официально-деловой стили. Являются группой функциональных стилей книжно-письменного языка и тоже считаются неоднородными стилями.

Относительная нестрогость публицистического стиля обусловлена эмоциональной составляющей и «социальной оценочностью». Развитие жанрового разнообразия публицистики расширяет спектр стилевых средств и размывает границы между публицистикой, официально-деловыми, техническими и научными текстами.

Множество подъязыков, создаваемых на основе правил общего языка и отличающихся от составляющих последнего лексико-фразеологическим составом, грамматикой, формой представления сообщения, лексикой конкретной предметной области может называться «языком научной прозы». При этом термин «научная проза» используется при наименовании того, что относится к средствам и текстам официально-делового стиля.

Термин научная (научно-техническая) проза относится к текстам, представляющим научный стиль. Интерес к текстам научного стиля обусловлен двумя обстоятельствами: 1) поисками оптимальных языковых средств для человеко-машинного общения и 2) необходимостью оптимизации научного обмена в условиях количественного роста и качественного усложнения накапливаемых знаний.

Очевидная связь между научно-техническим и официально-деловым стилями часто в практике обработки текстов ведет к их неразличению, объединению в один «метастиль». Близость этих стилей определяется такими общими свойствами, как:

– наличие согласованных ограничений на использование языковых средств, регламентация стандартизирующими и унифицирующими документами;

– отражение документами обоих стилей того, что называют «производственными отношениями»;

– связь текстов с официальным статусом лиц, участвующих в их создании, хранении, обработке и использовании;

– модельность ситуации, которая определяет ту формализованность, что относится ко всем языковым уровням таких текстов;

– большие возможности, относящиеся к автоматизации технологических процессов по созданию, обработке и использованию текстов;

– способность текстов обоих стилей поддаваться автоматическому переводу и реферированию, информационному поиску и другим аналитико-синтетическим операциям.

При формировании текстов обоих стилей используют высокую степень нормирования, которая осуществляется с помощью дополнительных инструментов: словарей, стандартов, списков, клише (штампов) и т.д. (Нормирование – это ряд последовательных процедур ограничительного характера). Разнообразие документов одного стиля, их функциональная и лингвистическая неоднородность свидетельствуют о том, что разбиение на функциональные стили является слишком общим и условным. Степень внутренней неоднородности стиля определяет количество его подстилей как разновидностей основного варианта. На практике существует взаимопроникновение стилей, что как раз и характерно для научно-технического и официально-делового стилей. В конечном счете, проблема выявления и определения функциональных стилей – это задача классификационнаяя и при этом совсем непростая, с учетом сложности лингвистических объектов.

Официально-деловой стиль. Трудность определения официально-делового стиля связана с большой неоднородностью типов текстов, представляющих этот стиль. Эта неоднородность имеет многоуровневый характер: количество подстилей, о которых упоминалось выше, в официально-деловом стиле трудно определить и провести условные границы между ними. При этом наблюдается парадокс официально-делового стиля: при высоком уровне формализованности материалов, однозначная внутренняя стратификация стилей и их составляющих удовлетворительных результатов не дает. Поэтому найти достаточно точное и строгое определение официально-делового стиля довольно сложно. Исторические корни формирования этого стиля восходят к приказному делопроизводству. Определить окончательным образом официально-деловой стиль еще не пришло время, так как невозможно даже перечислить исчерпывающим образом типы документов из-за отсутствия их конечного списка. Можно лишь определить ряд системных признаков этого стиля. К этим признакам относятся:

– установление посредством документа экономических или правовых отношений между субъектами этих отношений;

– непосредственная или опосредованная связь с различными государственными структурами;

– отбор некоторых формальных признаков, относящихся к оформлению вспомогательных элементов, внетекстовых параметров;

– нейтральность, отсутствие эмоциональности, полнота и точность формулировок, повторяемость, высокая частота архаичных или штамповых конструкций.

В отношении официально-делового стиля наиболее последовательно реализуются приемы стандартизации и унификации. При этом стандарти-

зуются прежде всего лексико-фразеологические и синтаксические составляющие. В настоящее время официально-деловой стиль русского языка – один из наиболее динамично развивающихся. Центральным понятием официально-делового стиля является ДОКУМЕНТ. Определить это понятие достаточно сложно. Самое лаконичное определение этого термина дано в Оксфордском словаре: «Документ – это текст или изображение, имеющее информационное значение». Универсальное определение документа как «любого материального носителя с закрепленной любым способом информацией на любом языке» также не отражает всей полноты использования этого термина.

Остается не до конца разрешенной и проблема жанрового разнообразия документов. Существующие классификации жанров оставляют много вопросов. В настоящее время предлагается следующее определение. Жанр документа – это функциональная разновидность речи, имеющая стандартизованную форму отражения, опирающаяся на средства официально-делового стиля и отвечающая требованиям композиционно-реквизитного состава, предусмотренным нормой и устоявшейся практикой. Заметим, что классификация документов не имеет единого основания. При этом отмечается, что исходные признаки классификации имеют в основном внеязыковой характер.

Вопросы для самоконтроля

1. Что такое дифференциация языка?
2. Поясните разницу между социолектом и диалектом.
3. Что такое подъязык?
4. Перечислите основные функциональные стили.
5. Какие общие свойства у научно-технического и официально-делового стилей?
6. Каковы системные признаки официально-делового стиля?

4. Композиционные особенности документов

В жизни каждому из нас в большей или меньшей степени приходится сталкиваться с написанием документов как в производственной, так и в бытовой сфере деятельности. К документам первого типа относятся разного рода акты, обоснования, рекламации, деловые письма и пр., а к документам второго – заявления, доверенности, объяснительные записки и т.д.

Работая с деловыми текстами или документами, надо знать, как правильно их составлять. Чаще всего под композицией документа понимаются

особенности пространственной соотнесенности частей документа, построение документа с точки зрения взаиморасположения тематически завершенных микротекстов в пределах документа, их физическое и смысловое позиционирование. Довольно интересным является подход, в котором композиция связывается с понятием информационной емкости документа, т.е. композиционные составляющие влияют на информативность документа, иными словами, на реальное количество информации, вложенной в текст. В отличие от текстов научно-технического стиля (монографий, статей, тезисов, отчетов и различных инструкций), имеющих довольно много общего по причине одинаковой целевой направленности, тексты документов отражают слишком разные цели. Универсальная структура научно-технического текста выглядит довольно устойчиво: постановка проблемы – гипотеза – объем исследований и терминология – общее рассмотрение проблемы – исторический обзор (анализ проблемы) – методика изучения и анализ полученных результатов – неблагоприятствующие факторы – аналогии – анализ причин, выделение важнейших противоречий и факторов – окончательное рассмотрение проблемы – выводы. Не все приведенные компоненты научно-технического стиля должны присутствовать в каждой работе, различные причины будут определять выбор компонента из перечня. Различным будет и объем разделов, соотнесенных с компонентами структуры.

Для деловых же документов установить единую структуру композиционных составляющих вряд ли возможно. Говорить о «правильном» построении документа можно лишь как о проявлении логики и вербализации той части документа, которая предполагает не составление, а сочинение документа (например, заявление, где заданы позиции и их порядок: адресат, адресант, название документа), а суть просьбы или обоснования излагаются посредством «свободного», нерегламентированного текста.

Композиция – это в известной степени пространственная характеристика текста. Заданность композиции документа решает важную задачу: способствует поддержке таких параметров, как полнота и точность сведений, необходимых для данного типа документов. Регламентация порядка изложения – это одна из основных составляющих композиционной характеристики, которая в значительной мере зависит от функционально-прагматического типа документа. Относительно невелика регламентация в деловых письмах, где ориентиром служит предметный интерес и сложившаяся традиция письма.

При составлении документа мы должны решать, по крайней мере, две задачи: ЧТО нужно и можно сказать в этом документе (определение смысла

документа) и КАК это можно сказать, чтоб документ имел соответствующий статус. В число единиц, формирующих документ, войдет большое количество устойчивых словосочетаний как опорных конструкций. Они определяют композицию текста, его модальность и тональность.

В документе (как и в научно-технических текстах разного рода) обязательно присутствуют рубрикации – выделение в тексте документа составных частей. Кроме ТОЧКИ – знака окончания предложения, для внешнего выражения композиционной структуры текста используется АБЗАЦ, под которым понимается либо отступ вправо на несколько знаков в начале строки, либо пропуск строки между сверхфразовыми единствами. Абзацный пропуск – это композиционный прием, используемый для показа перехода от одной мысли к другой. Наличие абзаца подчеркивает структуру и внутреннюю логику текста. Помимо абзаца, фрагменты в деловом тексте могут иметь также разные функции и размеры. Это могут быть главы, параграфы, разделы, подразделы, части, блоки. На практике текст часто строится так, что параграф и глава соотносятся как часть и целое. Очень часто эти фрагменты имеют свои собственные заголовки. В настоящее время сохраняется довольно большое разнообразие в использовании средств рубрикации.

Интересной и эффективной формой рубрикации является введение внутренних заголовков и подзаголовков. Эти средства не только вполне определенным образом отражают границы между фрагментами, но и логически упорядочивают текст, несут значительную коммуникационную нагрузку, выражая в сжатой форме содержание соответствующего фрагмента. Необходимо также обратить внимание на различия между внутренними заголовками и подзаголовком, следующим за основным заглавием. В этом случае подзаголовок применяется для более полного или аспектного раскрытия содержания. Заметим, что в отличие от текстов иных функциональных стилей, тексты официально-делового стиля чаще всего имеют небольшие размеры, а тексты научно-технической тематики могут быть как небольшими по размеру (инструкции), так и значительными (монографии).

Вопросы для самоконтроля

1. Что понимается под композицией документа?
2. Какие задачи решаются при составлении документа?
3. Что такое рубрикации? Перечислите их виды.
4. Для чего нужны рубрикации?

5. Состав лексико-фразеологических единиц официально-деловых текстов

Заметим, что здесь и далее, говоря об официально-деловом стиле, мы имеем в виду и стиль научно-технических текстов. В качестве общих характерных особенностей этих стилей выделяют следующие качества: точность, простота, ясность, объективность, абстрактность, обобщенность, информационная насыщенность, лаконичность, эмоциональная нейтральность, однозначность, безличность, логическая связность, использование терминологии, символов и графики, стандартизованность. Эти признаки в каждом конкретном тексте указанных выше стилей проявляются в различной степени. Абсолютное большинство перечисленных выше свойств находит свое проявление на лексико-фразеологическом уровне.

Предполагается, что такое качество, как **точность**, определяется группой факторов: на лексико-фразеологическом уровне – использованием моносемантической (однозначной по смыслу) лексики, терминологических единиц. При этом отвергается полисемичная (многозначная) лексика, а также лексика с размытыми, неустоявшимися значениями.

Простота как основное проявление рациональности реализуется через отказ от слов с эмоционально-оценочной окраской при одновременном достаточно частом использовани штампов-фразеологизмов. Кроме того, простота исключает словесную вычурность.

Ясность обеспечивается использованием лексики с устоявшимися значениями, исключением перенасыщенности текста терминологией.

Объективность не допускает использования разговорно-бытовой, оценочной лексики, вносящей субъективизм.

Понятию **абстрактности** также противоречит использование разговорно-бытовой лексики. Абстрактность связана с использованием абстрактной лексики, слов с отвлеченными значениями, строгой официально-деловой лексики.

Информационной насыщенности текстов официально-деловых документов способствует включение в них аббревиатур, различной терминологии.

Отказ от поэтизмов, разговорной, просторечной лексики способствует **эмоциональной** нейтральности.

Однозначность является принципиальным свойством. Моносемичная (однозначная) лексика, терминологизация, применение приемов, снимающих многозначность (в том числе использование синонимов и повторов),

способствуют однозначности документа и невозможности его двоякого толкования.

Стандартизованность обеспечивается не только использованием терминологических единиц, но и закреплением тех лексико-фразеологических сочетаний, которые носят характер клише (штампов), даже при непостоянности, вариативности их состава.

Если определить общие характеристики лексико-фразеологических единиц в официально-деловых текстах, то можно выделить следующие.

1. Способ номинации: стремление к реализации прямых лексических значений; семантическая мотивированность; отсутствие ограничений в использовании слов с мотивированными и немотивированными значениями; реализация общеязыковых закономерностей.

2. Возможности лексической сочетаемости: в этом отношении тексты официально- делового стиля (в частности, документы) обладают достаточно определенной спецификой, увеличение доли слов с несвободными значениями является одним из проявлений действия унифицирующих законов; унификация, кроме того, связывает используемую лексику.

3. Характер выполняемых функций: количество единиц со строгой номинацией не просто превышает количество экспрессивно-образных, но первые создают лексическую основу официально-деловых текстов.

4. Характер связи между лексическими значениями: различен для разных типов лексики, слова с автономными значениями (т.е. относительно независимые в языковой системе, обозначающие в основном конкретные предметы) в официально-деловых текстах заметно варьируют в зависимости от жанровой принадлежности и степени конкретности – абстрактности данного документа.

Надо отметить также, что использование однозначной лексики в текстах официально-делового стиля диктуется более общим требованием – необходимостью однозначности самого документа. Совершенно понятно в этом случае желание свести к минимуму встречаемость лексических омонимов, затрудняющих понимание текста.

Вопрос о степени использования синонимов в официально-деловых текстах не является простым. На первый взгляд, синонимия нежелательна, так как нарушает требования однотипности наименования предметов рассмотрения. Из синонимического ряда выбирается единица, если не стандартизованная, то унифицированная в рамках определенного вида документа. При этом предпочтения определяются традицией, распространенностью, понятийной определенностью самой единицы. С другой стороны, ограничения, ведущие к использованию лишь одной единицы синонимического ряда,

противоречат стилевым, доказательно-аргументирующим свойствам текста. Отсюда и ситуации, в которых синонимия может быть признана допустимой и даже желательной.

Большого внимания требуют к себе и паронимические (см. выше) явления, довольно часто представленные в деловых текстах разных жанров. Достаточно обратиться к малой части той высокочастотной лексики, которая встречается в текстах: «уплатить – оплатить», «безответный – безответственный», «командированный – командировочный», «экономичный – экономический», «информативный – информационный», «программный – программистский» и пр. Внимательное отношение к использованию паронимов препятствует искажению смысла делового текста.

Что касается использования фразеологизмов (устойчивых оборотов) в деловых текстах, то их состав довольно узок. В основном используются разнообразные клишированные единицы (штампы).

Анализ официально-деловых текстов различных типов позволяет говорить о ТРЕХ видах устойчивых многокомпонентных единиц:

1) единицы, входящие в состав терминологии соответствующей предметной области. Например, в конкретных документах: «гарантийные условия», «выполнение обязательств», «форс-мажорные обстоятельства», «спецификация компонентов программы», «взаимная ответственность сторон» и пр.;

2) довольно высокочастотные фразеологизмы собственно деловой сферы: «служебный документ», «административно-производственная ситуация», «образец оформления документа», «инструкция пользователя» и др.;

3) стандартные обороты речи, коммуникативные клише, которые с минимальными вариациями воспроизводятся в деловых текстах, придавая им определенную стилистическую окраску. Речь идет о таких единицах, как «учитывая вышеприведенные аргументы», «в связи со сложившейся ситуацией», «довести до Вашего сведения», «в порядке исключения» и т.д. Таким образом, типичность ситуаций порождает формирование класса единиц, называемых «речевыми формулами».

Вопросы для самоконтроля

1. Какие общие характерные свойства у официально-делового и научно-технического стилей?
2. На каком языковом уровне проявляются эти свойства?
3. Каковы общие характеристики лексико-фразеологических единиц в официально-деловых текстах?
4. Чем диктуется использование однозначной лексики?

5. Каковы противоречия в использовании единиц синонимического ряда?
6. Перечислите виды устойчивых многокомпонентных единиц.

6. Синтаксис официально-делового текста

Рассмотрим теперь синтаксические особенности деловых текстов. Принципиальным считается соображение о точности и однозначности официально-делового стиля. Следствия этого соображения, проявляющиеся на синтаксическом уровне, взаимодействуют с рядом других синтаксических особенностей, которые обусловлены лингвистическими, функциональными, историческими и прагматическими причинами.

Первой особенностью предложений в деловых текстах является их насыщенность синтаксическими клише (штампами). Это своеобразные «строительные блоки» предложений. Такая особенность характерна как для документов, так и для научно-технического стиля. В некоторых пособиях даже даны упорядоченные списки клишированных фрагментов предложений. Изучение таких списков может помочь в работе над текстами делового и научного стиля, улучшить эти тексты в стилевом отношении и ускорить работу по их написанию.

Вторая особенность предложений в деловых текстах – насыщенность разного рода второстепенными членами предложений. Связано это с необходимостью максимально снять неопределенности, однозначно определить объекты и отношения, рассматриваемые в таких текстах.

Третья особенность формулируется как гипотеза об особом характере распределительных характеристик предложений различного типа и разной длины. Существует даже утверждение, что сложность описываемых ситуаций и стремление к точности приводят к преобладанию сложных предложений над простыми. Но в этой ситуации пока скорее порождаются новые вопросы, чем решаются старые.

Четвертой особенностью синтаксиса делового текста является тенденция к обезличиванию предложений. На синтаксическом уровне это проявляется в потере подлежащего со значением лица.

Синтаксические особенности включают и способы изложения, в наибольшей степени присущие данному стилю. Из довольно большого набора способов изложения в текстах делового стиля чаще всего реализуются констатирующий, описательный, объяснительный, монологический, повествовательный способы.

К характерным особенностям текстов делового стиля можно отнести также и строгость порядка слов в предложении. Эта строгость является

следствием обязательного выполнения условий – однозначности и точности. Иногда отмечается частое употребление инфинитивных конструкций в рассматриваемых нами текстах. По-видимому, часто это связано как с их императивностью («создать», «обеспечить», «предложить», «приложить» и пр.), или, иными словами, с повелительным наклонением, так и с безличностью как типовой характеристикой этого стиля. Наряду с высокой частотой инфинитивных конструкций, отмечается также частое употребление условных предложений.

В текстах официально-делового стиля рассматривается также возможность употребления неполных предложений, но при этом не рекомендуется регулярное и частое их использование, поскольку это ведет к размыванию принципов стиля и приобретению текстом разговорной окраски.

Встречаемость сложных предложений в рассматриваемых нами текстах достаточно высока. Это обуславливается коммуникативными требованиями определенности и однозначности, так как с помощью сложного предложения более точно выражается сложная, многоаспектная мысль. Вместе с тем, довольно часто встречаются случаи, когда синтаксическая сложность приобретает неоправданно гипертрофированные формы: возникают предложения, которые слишком длинны и трудны для осмысления. Однако, как правило, сложность содержания отражается в синтаксической сложности, но, как и везде, при этом надо знать меру сложности.

Вопросы для самоконтроля

1. Что считается принципиальным для текстов делового стиля?
2. Перечислите особенности предложений в текстах официально-делового стиля.
3. Какие Вы узнали способы изложения в рассматриваемых нами текстах?
4. Чем обусловлена высокая частота сложных предложений в текстах делового стиля?

7. Стандартизация и унификация средств в деловых текстах

Отдельным вопросом при создании деловых текстов стоит вопрос об использовании речевых штампов, или клише, который относится к разделу лексики и фразеологии.

7.1. Роль речевых клише (штампов) в подготовке и восприятии делового текста

Хотя термин **клише**, о котором мы уже упоминали выше, довольно распространен в современной практической стилистике, но наряду с ним широко используется термин **речевой штамп**. Содержание этих терминов-синонимов толкуется как «стилистически окрашенное средство речи, отложившееся в коллективном сознании носителей данного языка как устойчивый и наиболее удобный знак для выражения определенного языкового содержания» [6].

Долгое время существовало весьма критическое отношение к речевым штампам при их использовании в текстах большинства функциональных стилей, однако сейчас это считается неуместным при рассмотрении роли штампов в текстах официально-делового стиля. Использование речевых штампов (клише) в деловых текстах является одним из проявлений логико-стилистичеких правил. Если для любого текста делового стиля должен выполняться комплекс условий, таких как унифицированность формы и лингвистических средств, стремление к однозначности, обеспечивающей информационную точность и компактность текста, то устойчивость и относительная малочисленность речевых штампов вполне объяснима, так как они помогают соблюсти вышеперечисленные условия. Таким образом, речевые штампы являются отражением процесса формализации в деловых текстах, а высокая частота встречаемости устойчивых оборотов деловой речи является отражением стандартизованности.

Среди элементов, образующих штампы, отмечается довольно большое их разнообразие:

– глагольные речевые штампы («ввести в действие», «просим принять участие», «утвердить отчет» и пр.);

– субстантивные речевые штампы («предварительное рассмотрение», «создание необходимых условий», «организация конференций» и др.);

– наречные и союзные речевые штампы («согласно полученным данным», «по мере необходимости/возможности», «в силу указанных причин», «в соответствии с требованиями» и т.д.).

Известно, какие затруднения испытывает большая часть учащихся, студентов, сотрудников разного ранга при составлении деловых бумаг различного рода, поэтому использование речевых штампов в деловых текстах помогает конкретнее, лаконичнее и легче выразить мысль, но при этом надо стараться не перегружать текст излишними штампами: все хорошо в меру. По-

строение модели делового текста средствами клишированных конструкций может быть интересной учебной и исследовательской задачей.

Все рассмотренное в этом разделе относится к **написанию** деловых текстов с использованием клише. Но и при **восприятии** делового текста речевые штампы играют положительную роль, так как для опытного специалиста компонентный состав речевых штампов в определенной мере соотносится с содержательно- функциональными особенностями текста.

Таким образом, структурная и смысловая устойчивость речевых штампов служит одним из факторов, способствующих созданию унифицированных и стандартизованных текстов. Формульный характер речевых клише способствует экономии времени, однородности однотипных документов и упрощает написание, обработку и восприятие деловых текстов.

Вопросы для самоконтроля

1. Как толкуется термин клише?
2. Каковы функции и роль клише в текстах?
3. Перечислите типы клише.
4. Чему способствуют клише?

Заключение

В этом разделе нашего пособия мы познакомились с тем, что такое информационная стилистика, с другими функциональными стилями, в частности, официально-деловым стилем, стилем научно-технических текстов, стилем документов и узнали, чем они отличаются от стиля художественной литературы и разговорно-бытового стиля. Рассмотрели язык как основное средство коммуникации и его основные уровни, познакомились с понятием дифференциации языка, узнали, что такое жанр документа. Коснулись также композиционных особенностей делового текста, его состава и лексико-фразеологических единиц. Узнали, что синтаксис официально-делового текста имеет свои особенности и выяснили, каким образом эти особенности помогают построению и восприятию текстов официально-делового стиля. А в заключение рассмотрели, какую роль играют стандартизация и унификация средств делового текста на примере понятия клише.

Все это были в основном теоретические лингвистические аспекты, без знания которых невозможно правильное композиционное составление и грамотное написание статей, отчетов, инструкций, монографий и прочих деловых документов, а также их восприятие.

Список литературы

1. Энциклопедический лингвистический словарь. – М: Сов. энциклопедия, 1990.
2. Несговорова Г.П. Программно-методические средства подготовки русскоязычных текстов при разработке программных систем // Методы теоретического и системного программирования. – Новосибирск, 1991. – С. 40-48.
3. Ершов А.П. К методологии построения диалоговых систем: феномен деловой прозы // Вопросы кибернетики: Общение с ЭВМ на естественном языке. – М., 1982.
4. Андреев Н.Д. Стилистико-комбинаторные методы в теоретическом и прикладном языкознании. – Л., 1984.
5. Винников Ю.В. Типы научных и технических текстов и их лингвистические особенности. – М., 1984.
6. Кушнерук С.П. Документная лингвистика (русский деловой текст). - Волгоградский гос. университет, 1997.
7. Винокур Т.Г. Лингвистический энциклопедический словарь. – М., 1990.
8. Городняя Л.В., Несговорова Г.П. Принципы конструирования электронного учебника по информационной стилистике русского языка. – Новосибирск, 1994. – 19 с. – (Препр. / РАН. Сиб. отд-ние. ИСИ; № 19).
9. Жинкин Н.И. Речь как проводник информации. – М., 1982.
10. Кожина М.Н. Стилистика русского языка. – М., 1983.

II. ЯЗЫКОВЫЕ И СТИЛИСТИЧЕСКИЕ НОРМЫ РУССКОГО ЯЗЫКА

Введение

В представленном выше разделе («Стилистика деловых текстов») мы рассматривали в основном теоретические аспекты. В этом же разделе разберем некоторые вопросы, связанные с практическим использованием русского языка в той или иной деятельности человека, в частности, при написании деловых текстов.

Русский язык 10 лет учат в средней школе, но постигать все тонкости «великого и могучего» русского языка можно всю жизнь. Подлинной грамотностью следует считать не только умение читать и писать без орфографических и пунктуационных ошибок, но и умение правильно выражать свои мысли в устной и письменной форме, т.е. овладевать культурой родной речи. Поэтому ниже мы постараемся рассмотреть те тонкости русского языка, до изучения которых не дошло дело в курсе средней школы и которые так необходимы в жизненной практике каждому человеку.

1. Культура речи и языковая норма

Что же понимать под культурой речи? Культурной можно считать такую речь, которая отличается смысловой точностью, богатством и разносторонностью словаря, грамматической правильностью, логической стройностью и выразительностью. Культурная речь – это, прежде всего, речь нормированная. В своей устной форме она должна отвечать существующим в настоящее время нормам произношения (орфоэпии), а в письменной – нормам орфографии и пунктуации.

Языковые нормы бывают двоякого рода. Одни из них строго обязательны и не допускают никакого нарушения, как, например, нормы грамматические или орфографические. Такие нормы применяются, когда решается вопрос: как надо сказать или написать? Другие нормы представляют собой наиболее распространенные, предпочтительные речевые варианты, закрепившиеся в практике использования, наилучшим образом выполняющие свою функцию. Именно с вариантами норм мы сталкиваемся в стилистике, когда решаем вопрос, как лучше сказать?

Однако ответить на этот вопрос однозначно достаточно сложно. Несмотря на свою устойчивость, норма языка изменяется в ходе развития самого языка. Это приводит к тому, что в данный период для одного и того же языкового явления существует несколько способов его выражения. Например, допустимы обе формы: «чашка чаю – чашка чая» и «много народу – много народа».

Другой путь появления вариантов нормы связан с тем, что в языке, в зависимости от выполняемой им функции, от конкретных условий его использования, возникают особые разновидности – стили, о чем мы говорили выше. Каждый стиль характеризуется своими признаками: преимущественным использованием определенных лексических и фразеологических средств, синтаксических конструкций и т.д. Среди стилей в общем случае выделяются две большие группы: 1) книжные стили (чаще в письменной форме) и 2) разговорный стиль (чаще в устной форме), которые, в свою очередь, делятся на подгруппы (см. Часть I).

Каждая группа стилей характеризуется своими вариантами норм. Сравните формы: «договор**ы**- договор**а**» и «слесар**и** – слесар**я**», из которых одни (с окончанием -ы/-и) являются книжными, а другие (с окончанием -а/-я) – разговорными. Таким образом, наряду со старыми и новыми вариантами нормы сосуществуют варианты, обязанные своим появлением выделению в языке различных стилей, – книжные и разговорные. Как правило, старые варианты совпадают с книжными, а новые – с разговорными.

Наличие вариантов нормы, т.е. двояких форм для выражения одного и того же языкового явления, обогащает язык, позволяет более точно передать мысль. Но при этом выбор варианта в каждом конкретном случае должен быть обоснован. Сформулируем десять основных признаков хорошей, точной речи (это относится как к письменной, так и к устной формам речи).

1. Правильность речи – это ее соответствие принятым в определенную эпоху литературно-языковым нормам.

2. Точность речи – это ее соответствие мыслям говорящего или пишущего.

3. Ясность речи – это ее доступность пониманию слушающего или читающего.

4. Логичность речи – это ее соответствие законам логики. Небрежность языка обуславливается нечеткостью мышления.

5. Простота речи – это ее естественность, отсутствие красивостей, вычурности слога.

6. Богатство речи – это разнообразие используемых языковых средств.

7. Сжатость речи – это отсутствие лишних слов, повторов.

8. Чистота речи – это отсутствие в ней диалектных, жаргонных, просторечных, вульгарных слов, а также иностранных, если в их использовании нет крайней необходимости.

9. Живость речи – это ее выразительность, образность, эмоциональность.

10. Благозвучие речи – это ее соответствие требованиям приятного для слуха звучания, т.е. подбор слов с учетом их звуковой стороны, выполнение орфоэпических правил (правильной постановки ударений).

Если **язык** – это совокупность лексико-фразеологических и грамматических средств, используемых его носителями для целей общения, воздействия, то **стиль** – это приемы, способы, манера их использования, т.е. другими словами, языковые стили – это разновидности языка, связанные с различными сферами деятельности людей, различными условиями человеческого общения. Строя высказывание, важно учитывать не только значение отдельно взятого слова, но и его связь с другими словами, так называемую лексическую сочетаемость, т.е. способность одного слова образовывать по смыслу грамотное сочетание с другим словом.

Рассмотрим пример: «Наш народ сумел завоевать техническую революцию и начать покорять космос». Синтаксически предложение построено верно, но «революцию» не «завоевывают», а «совершают», т.е. правильная лексическая сочетаемость с точки зрения русского языка здесь нарушена.

Вопросы для самоконтроля

1. Найдите ошибки в лексической сочетаемости в следующих предложениях и исправьте их.

«Однажды Павел предупредил мать, что к нему придут запрещённые люди».

«Молодогвардейцы верили в неминуемую победу советского народа в Великой Отечественной войне».

«Добролюбов под Катериной видел луч света, а под Кабанихой – тёмное царство.»

«Ученик подробно рассказал автобиографию писателя А.А. Фадеева».

«Самым бедным из этой группы действующих лиц является язык Варвары».

2. Стилистический синтаксис

Можно наверняка предположить, что очень немногие из вас станут литераторами. Но с полной уверенностью можно сказать, что в любой области вашей будущей деятельности умение выразительно, живо, убедительно, аргументировано и доходчиво выражать свои мысли в устной и письменной форме сослужит вам хорошую службу. «Великий, могучий, правдивый и свободный русский язык» содержит в себе неисчерпаемые стилистические богатства, овладеть которыми в большей или меньшей степени по силам каждому.

Взятые сами по себе, слова не выражают мысли, они лишь являются строительным материалом для предложения. И только включенные в предложение, приобретя нужную форму и выстроившись в надлежащем порядке, они могут выполнять свое назначение – выражать мысли, чувства, волю.

Поговорим о стилистическом синтаксисе. Сопоставим три варианта примерно одного и того же текста:

Кто такие современные исследователи космоса?

Кто они, современные исследователи космоса?

Современные исследователи космоса. Кто они?

Наиболее выразительным считается последний вариант. Если в первых двух содержится как бы нейтральный вопрос, то третий вариант стилистически окрашен благодаря тому, что текст разделен на две части: в первой части привлекается внимание к предмету мысли и только потом задается вопрос, в результате чего и создается большая выразительность. Подобные «раздвоенные» конструкции напоминают назывные предложения: и в том и

в другом случае в центре внимания оказывается форма именительного падежа. Она создает представление о предмете или называет тему дальнейшего высказывания (их так и называют: именительный представления). Сравните: именительный представления: «Земля. На ней никто не тронет. Лишь крепче прижимайся к ней.» (К.Симонов) и назывное предложение: «Шепот, робкое дыханье, трели соловья...» (А.Фет). «Разрыв» предложения на части, подачу его отдельными «порциями» находим и в так называемых присоединительных конструкциях: «В комнате грянул марш. Походный марш. Такой бодрый, веселый.» (К.Федин).

Построением предложения можно выразить многое. Богатые стилистические возможности дает использование в речи разных стилей сложных предложений. Сопоставим три типа сложных предложений: сложносочиненное, сложноподчиненное и бессоюзное.

Приближалась ночь, и пришлось возвращаться домой.

Приближалась ночь, поэтому пришлось возвращаться домой.

Приближалась ночь – пришлось возвращаться домой.

Пришлось возвращаться домой, так как приближалась ночь.

Пришлось возвращаться домой, приближалась ночь.

Здесь имеет место синонимичность этих предложений, т.е. близость выражаемого ими содержания и наличие причинно-следственных отношений между частями предложения. Но при этом каждому из приведенных сложных предложений присущи свои смысловые и грамматические особенности, обусловленные наличием или отсутствием союза, значением этого союза, порядком следования частей предложения, интонацией, которая на письме отражается пунктуацией.

Выбрать подходящий вариант предложения можно только в условиях конкретной ситуации, с учетом контекста, стиля, жанра произведения. Обобщенно можно лишь сказать, что более простые конструкции – с союзом – являются межстилевыми, т.е. используются как в книжном стиле, так и в разговорном, тогда как бессоюзные предложения больше характерны для книжных стилей. Таким образом, подтверждается положение, что стилистика – учение о наиболее удачном конкретном выборе языковых средств и начинается она там, где имеется возможность выбора.

Вопросы для самоконтроля

1. Приведите примеры именительного представления.
2. Почему такие конструкции создают большую выразительность?
3. Приведите примеры присоединительных конструкций.
4. В чем заключается синонимичность предложений?

5. Приведите примеры синонимичных предложений.
6. Что присуще каждому из синонимичных предложений?

3. Порядок слов в русском языке

В русском языке порядок слов (точнее, порядок членов предложения) считается свободным. Это значит, что в предложении нет строго закрепленного места за тем или иным его членом. Например, предложение, состоящее из пяти знаменательных слов: «Редактор вчера внимательно прочитал рукопись» допускает 120 вариантов в зависимости от перестановки членов предложения. (Потренируйтесь на досуге в составлении этих вариантов).

Различаются прямой порядок слов, определяемый типом и структурой предложения, способом синтаксического выражения данного члена предложения, его местом среди других слов, которые непосредственно с ним связаны, а также стилем речи и контекстом, и обратный порядок, являющийся отступлением от обычного порядка и выполняющий чаще всего функцию инверсии, т.е. стилистического приема выделения отдельных членов предложения путем их перестановки. Прямой порядок характерен для научной и деловой речи, обратный – широко используется в публицистических и литературно-художественных текстах. Особую роль обратный порядок играет в разговорной речи, имеющей свои типы построения предложений.

Определяющим фактором расположения слов в предложении является целенаправленность высказывания, его коммуникативное задание. С ним связано так называемое актуальное членение высказывания, которое предполагает движение мысли от известного, знакомого – к неизвестному, новому: первое (основа высказывания, тема) обычно заключено в начальной части предложения, второе (ядро высказывания, рема) – в его конечной части. Сравним два предложения:

1) «12 апреля 1961 года состоялся полет Ю.А. Гагарина в космос, первый в истории человечества» (исходной точкой, основой высказывания является указание на дату, т.е. сочетание «12 апреля 1961 года» – это тема, а ядром высказывания – остальная часть предложения, которая логически подчеркивается, – это рема);

2) «Полет Ю.А. Гагарина в космос, первый в истории человечества, состоялся 12 апреля 1961 года» (здесь основой высказывания является сообщение об историческом полете Ю.А. Гагарина – это тема, а ядром высказывания – указание на дату, которое логически подчеркивается, – это рема).

Вопросы для самоконтроля

1. Какой порядок слов существует в русском языке?
2. Какой порядок слов характерен для научной и деловой речи, а какой – для публицистической и литературно-художественной?
3. Что такое актуальное членение высказывания?
4. Дайте определение темы и ремы. Приведите примеры.

4. Типы синтаксической связи в предложении

В русском языке в предложении различаются два основных типа синтаксической связи – сочинение и подчинение.

При сочинении в связь вступают синтаксически равноправные, независимые друг от друга члены предложения, например: «книга и тетрадь (лежат на столе)», «(читаю) книги, газеты, журналы». При подчинении в связь вступают синтаксически неравноправные элементы: один зависит от другого, например: «читать книгу», «совет друга». Подчинение имеет три разновидности синтаксических связей: 1) согласование, 2) примыкание, 3) управление.

Согласование – это такой вид подчинительной связи, при котором зависимое слово уподобляется в своей форме главному слову: «ранняя весна», «третье издание», «мои увлечения».

Примыкание – это такой вид подчинительной связи, при котором зависимость подчиненного слова выражается лексически, порядком слов и интонацией. Примыкают неизменяемые знаменательные слова (наречие, инфинитив, деепричастие): «тихо шептать», «предложил войти», «говорил улыбаясь».

Управление – это такой вид подчинительной связи, при котором зависимое слово ставится в определенной падежной форме (без предлога или с предлогом), обусловленной лексико-грамматическим значением главного слова, например: «читать письмо», «интересоваться искусством», «любовь к родине».

Выделяются различные виды управления в зависимости от морфологической природы главного слова, наличия или отсутствия предлога перед зависимым словом, характера связи между обоими словами. Управление бывает присубстантивное (чтение книги), приадъективное (способный к музыке), приглагольное (писать пером), принаречное (ответил лучше меня). Различается управление непосредственное, или беспредложное (видеть картину), и посредственное, или предложное (смотреть на картину). Управ-

ление называется сильным, если между главным и зависимым словами существует необходимая связь, выражающаяся в том, что главное слово нуждается в распространении определенной падежной формой, например: «испытывать станок», «нарушать тишину», «полон бодрости». При слабом управлении связь между обоими словами необязательная, она не обусловлена лексико-грамматическими особенностями главного слова как связь необходимая: «солнце закатилось (за лесом)», «часто гуляю (по вечерам)».

Согласование и примыкание являются довольно простыми видами синтаксической связи. Наиболее сложным из них является управление. Использование этого вида синтаксической связи связано с определенными трудностями, поэтому ниже мы уделим ему особое внимание.

Вопросы для самоконтроля

1. Назовите основные типы синтаксических связей в русском языке.
2. Какие синтаксические связи имеет подчинение?
3. Дайте определение синтаксических связей при подчинении.
4. Перечислите виды управлений. Приведите соответствующие примеры.

5. Модель управления, критерии выбора грамматической формы

В этом разделе на конкретных примерах рассмотрим различные, наиболее типичные, виды управления в русском языке.

5.1. Беспредложное и предложное управление

При выборе вариантных беспредложных и предложных конструкций типа «вытянуться линией – вытянуться в линию», «собираться кучками – собираться в кучку» учитывается различие в смысловых оттенках: предложные сочетания «в линию, в кучку» указывают на большую степень концентрации действия. Обычно предложные конструкции, в которых отношения между словами выражаются не только падежным окончанием, но и предлогом, имеют более конкретный характер.

Сопоставляя сочетания «лицо у девушки – лицо девушки», отмечаем в первом из них более четкое выражение принадлежности и связь со сказуемым в предложении (ср.: «лицо у девушки побледнело»). Более конкретное значение предложных конструкций находим в следующих примерах: «интересный всем – интересный для всех», «полезный детям – полезный для детей», «видеть первый раз – видеть в первый раз».

В сочетаниях типа «вернуться поездом – вернуться на поезде», «ехать трамваем – ехать на трамвае» предложные конструкции характеризуются большей степенью сочетаемости с названиями средств транспорта.

В других случаях выявляется дополнительное смысловое различие. Сочетания «идти по полю», «идти по лесу» обозначают движения в отдельных местах названного пространства, а синонимические сочетания «идти полем», «идти лесом» указывают на непрерывность линейного движения.

Сочетания типа «работать вечерами – работать по вечерам» различаются тем, что беспредложные конструкции указывают только на совершение действия в одни и те же отрезки времени, а предложные конструкции обозначают к тому же регулярное повторение действия.

Предложная конструкция позволяет устранить двузначность в сочетаниях типа «письмо матери» (чье письмо? или письмо к кому?). В значении «письмо, адресованное матери» следует употреблять предлог: «письмо к матери», что снимает семантическую омонимию (см. об этом выше). В сочетаниях «оперировать фактами или проверенными данными» правильной считается беспредложная конструкция, а в сочетаниях «равноправны друг другу – равноправны друг с другом» правильной считается только предложная конструкция. В других случаях различие между предложными и беспредложными конструкциями имеет стилистический характер: в парах «считать бездельником – считать за бездельника» предложный вариант является устаревшим.

Беспредложные конструкции, появившиеся сначала в профессионально-технической речи, получили в некоторых случаях широкое распространение: «глубиной пять метров», «ценой три рубля», «массой два килограмма» и т.д.

5.2. Выбор предлога

При выборе предлога в синонимических конструкциях учитывается различие в смысловых и стилистических оттенках между ними. Рассмотрим это на примерах:

«за подписью и печатью – с подписью и печатью» – первый вариант присущ официально-деловому стилю;

«в подтверждение – для подтверждения» – первое сочетание свойственно книжно-деловому стилю, второе – нейтральному;

«стрелять в противника – стрелять по противнику» – в первом сочетании указывается направление действия на объект, во втором – распределение действия на ряд лиц;

«пройти около километра – пройти с километр» – второй вариант имеет разговорный характер;

«лекции на объявленные темы – лекции по всем объявленным темам» – во втором варианте значение обобщения;

«с помощью техники – при помощи техники» – второй вариант конкретизирует действующее лицо;

«с целью осуществить – в целях осуществления» – первая конструкция связана с инфинитивом, вторая – с отглагольным существительным, преимущественно в деловой речи ;

«стол о трех ножках – стол на трех ножках» – первый вариант устаревший.

Синонимический ряд образуют предлоги с изъяснительным значением, например: «разговоры о поездке – про поездку – насчет поездки – относительно поедки – касательно поездки». В этих сочетаниях можно отметить убывающую конкретизацию предмета речи и стилистическое различие: разговорный характер предлогов «про, насчет», книжный (присущий старой и деловой речи) – предлогов «касательно, относительно» и нейтральный характер предлога «о» при глаголах речи или мысли.

Синонимичны также многие предлоги, выражающие пространственные отношения, например: «у дома – при доме – около дома – возле дома – подле дома – вблизи дома». Значение наибольшей степени близости выражается сочетаниями с предлогами «при, у», средней близости предлогами «около, подле, возле», а значение наименьшей степени близости – предлогом «вблизи». Синонимичны в ряде конструкций предлоги «в – на» и их антонимы «из – с». Например: «ехать в поезде – ехать на поезде», «работать в огороде – работать на огороде». Здесь между вариантными конструкциями имеются смысловые или стилистические различия.

5.3. Выбор падежной формы

В некоторых конструкциях допускаются двоякие падежные формы в зависимости от той или иной стилистической окраски. В паре «вершить судьбами – вершить судьбы» второй вариант характерен для официального стиля, с оттенком устарелости. Выбор падежа может зависеть от лексического состава конструкции. Ср.: «в отсутствие»... (форма винительного падежа) – «в присутствии»... (форма родительного падежа); «в прошлые годы – в двадцатых годах» – здесь при указании десятилетий посредством порядковых числительных обычно употребляется не винительный, а предложный падеж.

При переходных глаголах с отрицанием в одних случаях явно преобладает употребление родительного падежа дополнения, в других – винитель-

ного, в-третьих – наблюдается их факультативное использование. Рассмотрим некоторые примеры.

Родительный падеж (при глаголе с отрицанием) употребляется в следующих случаях.

1. При наличии в предложении частицы «ни» или местоимения/наречия с этой частицей: «Никогда еще он не ощущал так горестно своей беззащитности». «До вас еще никто этого браслета не надевал».

2. При глаголах «иметь, получать, доставать» и пр., а также глаголах мысли, желания, ожидания: «Ты не чувствуешь его пафоса», «не хватает времени, не получил приказа, не достал билета».

3. В устойчивых сочетаниях: «не испытывает желания, не питает надежды, не обращает внимания».

4. В безличных предложениях: «Не нагнать тебе бешеной тройки». Эти примеры, конечно же, не исчерпывают всего списка случаев употребления родительного падежа при глаголе с отрицанием.

Винительный падеж, ослабляющий значение отрицания, употребляется в следующих случаях.

1. При указании на конкретный объект: «не отрецензировал рукопись, которую ему прислали».

2. При выражении дополнения одушевленным существительным: «не пожалеет ни папу, ни маму». То же при географических названиях: «не сдавай Порт-Артур».

3. При двойном отрицании: «не могу не сказать несколько слов».

4. В побудительных предложениях: «гляди под ноги, не смеши народ» и во многих других случаях.

Факультативное употребление родительного и винительного падежей при переходном глаголе с отрицанием связано со стилистическим различием: конструкции с родительным падежом характерны для книжной речи, а с винительным – для разговорной.

5.4. Модель управления

Перечислить, а тем более запомнить все указанные типы конструкций очень сложно, а приведенные выше примеры свидетельствуют о трудности выбора правильного варианта, и при этом не существует конкретных правил, какой из вариантов предпочесть. Но для этого существует модель управления, которая призвана облегчить правильный выбор.

Модель – это образец, служащий стандартом (эталоном) для массового воспроизведения; то же, что «тип», «схема», «парадигма», «структура». Например, модель спряжения или склонения, словообразовательная модель,

модель предложения и в этом же ряду – модель управления. Конструирование модели не только одно из средств отображения языковых явлений и процессов, но и объективный практический критерий проверки истинности наших знаний о языке. Чтобы выбрать правильную конструкцию русского языка для данного контекста, достаточно обратиться к словарю-справочнику «Управление в русском языке» [4].

Некоторые глаголы имеют при себе управляемое слово в различной предложно-падежной форме, что связано с разными смысловыми или стилистическими оттенками. Рассмотрим на конкретных примерах модели управления некоторых глаголов русского языка:

бросить «что» (значение объекта: «бросить камень в воду») – бросить «чем» (значение орудия действия: «бросить палкой в собаку»);

вершить «что» (решать, давать какое-либо решение: «вершить правое дело») – вершить «чем» (распоряжаться: «вершить судьбами») ;

воздать «что» (отдать, выразить, оказать: «воздать должное») – воздать «чем» (отплатить: «воздать добром за зло»);

гнушаться «кого» («гнушаться нечестных людей») – гнушаться «чем» («гнушаться подачками»);

«завязать узел» (на пакете) – «завязать узлом» (галстук);

«лежать на постели» (отдыхать) – «лежать в постели» (быть больным);

напомнить «что» (в полном объеме: «напомнить сказанное») – напомнить «о чем» (в общих чертах: «напомнить о случившемся»);

поражаться «чем» (восхищаться: «поражаться величием и красотой сооружения») – поражаться «чему» (удивляться: «поражаться отваге альпинистов»);

принадлежать «кому» (составлять собственность: «приусадебный участок принадлежит брату») – принадлежать «к кому» (входить в состав: «все они принадлежат к одной семье»);

сообщить «что» (по существу: «сообщить новости») – сообщить «о чем» (в общем виде: «сообщить о результатах»);

удостоить «чего» (признав достойным, наградить чем-либо: «удостоить награды») – удостоить «чем» (сделать что-либо в знак внимания: «удостоить ответом»);

упрекать «в чем» (объект упрека: «упрекать в небрежности») – упрекать «за что» (причина упрека: «упрекать за плохое поведение»).

Управляемое слово в различной падежной форме может находиться не только при глаголе, но и при имени существительном, например: необходимость «чего» (в значении «обязательность»: «доказывать необходимость

подкреплений») – необходимость «в чем» (в значении «потребность»: «необходимость в его присутствии»).

Здесь приведено лишь небольшое количество примеров, связанных с управлением, и напоминаем, что в случае сомнения в выборе правильной конструкции, следует обращаться к словарю-справочнику Э.Д. Розенталя [4].

Вопросы для самоконтроля

1. Какие конструкции более характерны для профессионально-технической речи (беспредложные или предложные)? Приведите примеры.
2. Что учитывается при выборе предлога в синонимических конструкциях?
3. В каких случаях употребляется родительный падеж при глаголе с отрицанием, а в каких – винительный? Перечислите, приведите примеры.
4. Что понимается под моделью в русском языке?
5. Чем является конструирование разных моделей в языке?
6. Приведите примеры моделей управления глаголов и существительных.

Заключение

Итак, наше путешествие в мир языка и стилистики закончилось. Мы затронули совсем небольшую часть того огромного мира, который зовется «русский язык». Но перед каждым из вас при желании открыты широкие возможности продолжать обучение самостоятельно, так как нет пределов изучению русского языка.

В заключение уместно напомнить высказывание знаменитого французского философа и писателя-просветителя Вольтера: «Выучить несколько языков – дело одного или двух лет; а чтобы научиться говорить на своём родном языке как следует, надо полжизни».

Список литературы

1. Розенталь Д.Э. Говорите и пишите по-русски правильно. – М., 2007.
2. Розенталь Д.Э. Справочник по правописанию и литературной правке. – М., 2007.
3. Лингвистический энциклопедический словарь. – М., 1987.
4. Розенталь Д.Э. Управление в русском языке. Словарь-справочник. – М., 1986.

ПРИЛОЖЕНИЕ

Чтобы закрепить все то, о чем было рассказано выше, предлагаются упражнения, к которым в конце даются справки и пояснения.

Грамматико-стилистические упражнения со справками и пояснениями

I. Грамматико-стилистические упражнения

1. Найдите ошибки в употреблении форм рода и исправьте их.
   1) Вся школа участвовала в уборке ранней картофели.
   2) Руководительницей кружка «Умелые руки» у нас была Валя Зайцева.
   3) Скоро наступит долгожданное день рождение.
   4) Билет с плацкартом на субботний день достать трудно.
   5) Она получила диплом учительницы русского языка и литературы.

2. Объясните, почему существительные употреблены в единственном числе.

Материальная и духовная культура; внешняя и внутренняя политика; постоянный и действенный контроль; тяжелая и легкая атлетика; головной и спинной мозг; в верхнем и нижнем течении реки; существительные мужского, женского и среднего рода; программы начальной и средней школы; не только большой, но и красивый дом; строительство первой и второй очереди метрополитена.

3. Допишите окончания, объясните, почему возможен только один вариант.
   1) Музыкальные инструменты изготовляются из дуб... и берез... .
   2) Герб – это особый знак, символ государств... или город... .
   3) Семь воинов награждены орден... «Знак почета».
   4) Люди взрывали кам..., рубили лес... и стояли по пояс... в ледяной воде.
   5) Девушки закрыли лиц... передник... .

4. Установите сходство и различие слов I и II группы.
   I. Свидетель, житель, любитель, избиратель, читатель, мыслитель.
   II. Выключатель, громкоговоритель, путеводитель, предохранитель.

5. Распределите слова по группам в зависимости от стилистической окраски: общеупотребительные (1), книжные (2), разговорные (3).

Угнетатель, обманщик, фрезеровщик, нытик, второгодник, озорник, умник, весельчак, юнец, упрямец, республиканец, болельщик, глашатай, коллекционер, журналист.

6. Найдите синонимы.

Дорогой зовут автостраду
И тропку, бегущую рядом,
И шлях, что лежит на равнине,
И путь каравана в пустыне.

7. Укажите стилистическую окраску выделенных слов.

1) Я занялся рассмотрением картинок, украшавших его скромную ОБИТЕЛЬ.

2) Белка находит себе теплое ЖИЛЬЕ в дупле дерева.

3) Темная и грязная КАМОРКА – жалкое ЖИЛИЩЕ старика.

8. Вставьте подходящее слово из паронимов.

1) В ... Черноморского побережья Кавказа отдыхали пионеры.

2) Гость провозгласил ... в честь юбиляра.

3) Мы остановились в ... . Куда идти?

4) Произошло небольшое ... : мы перепутали адрес.

5) Мать была в ... : до сих пор нет письма от сына.

6) Он был человек упрямый, неустрашимый до ... .

9. Вместо точек вставьте подходящие по смыслу прилагательные. Сравните, насколько Вы близки к оригиналу.

... шагами прошел я ... площадь кустов, взобрался на холм и, вместо ожиданий ... равнины с ... леском направо и ... церковью в отдалении, увидал совершенно другие ... места. У ног моих тянулась ... долина; прямо напротив ... стеной возвышался ... осинник. Я остановился в недоумении. Меня тотчас обхватила ..., ... сырость, точно я вошел в погреб: ..., ... трава на дне долины, ..., белела ... скатертью. (И. Тургенев)

10. Охарактеризуйте данные предметы с разных сторон, подобрав к ним относительные и притяжательные прилагательные.

Игрушка, стол, газета, галстук, песня, линейка, часы, лодка, гнездо, сад, дом, кукла, ваза, платье, кресло.

11. Покажите многозначность прилагательного, подобрав к нему существительные.

Живой, зеленый, сахарный, ледяной, железный, мраморный, минорный, горячий, свинцовый, стеклянный, дорогой, круглый, добрый, бронзовый.

12. Расположите данные синонимы по степени нарастания основного признака.

Колоссальный, большой, гигантский, громадный, огромный.

Дремучий, частый, густой.

Крохотный, небольшой, маленький, микроскопический.

Молниеносный, стремительный, быстрый.

Гениальный, способный, талантливый, даровитый.

Пытливый, любопытный, любознательный.

13. Подберите к данным прилагательным слова с противоположным значением (антонимы). Употребите их в сочетании с существительными.

Смешной, роскошный, тучный, внимательный, заурядный, бодрый, обрывистый, твердый, деликатный, мрачный, тусклый, пестрый, шершавый, ясный, разговорчивый, пушистый, скучный, энергичный, горячий, простодушный.

14. Вместо точек поставьте числительные «оба – обе» в нужном падеже.

1) Недавно я получил два письма, ... от брата.

2) По ... берегам Невы раскинулся город.

3) ... лыжи оказались сломанными.

4) Спускаясь вниз, я схватился ... руками за канат.

5) Вдоль ... сторон аллеи росли розы.

15. В бытовой и художественной речи числительные могут употребляться в переносном значении; объясните значение слова «один» в примерах.

1) В одном невзрачном домишке на окраине городка родился мальчик.

2) Мы вместе с ней в одной учились школе.

3) На двоих у нас с братом были одни лыжи.

4) За одни сутки он стал совсем седым.

5) Мне осталась одна забава: пальцы в рот да веселый свист.

16. Определите, в каких предложениях притяжательные местоимения употреблены неправильно. Внесите поправки.

1) В комедии югославского классика Нушича показана жизнь своей Родины.

2) Книги молодых писателей идут в одном строю с произведениями своих отцов.

3) Упорно и прилежно работал он над его стихами.

4) Во время болезни Володя получал письма от его товарищей.

17. Замените повторяющиеся существительные подходящими по смыслу местоимениями.

1) Вожатый поручил пионеру вырастить рассаду у пионера дома.

2) Отец сказал, чтобы Владимир отнес прибор в кабинет отца.

3) Командир приказал бойцу раздобыть оружие для бойца.

18. Вставьте вместо точек указательные местоимения в нужной форме.

1) Ребята неправильно обрабатывали грядки. Бригадир указал ... .

2) Туристы надеялись ... , что погода их не подведет.

3) Благодаря ... , что товарищи помогли, Борис не отстал в учебе.

4) На выставке нам показали ... , о чем мы раньше только читали.

19. Замените данные выражения подходящими по смыслу устойчивыми сочетаниями, в состав которых входят местоимения.

Потерять самообладание; нечто неопределенное; смотреть внимательно; искусный мастер; спешить; скучать от безделья; чувствовать себя неловко; оглядываться беспрестанно.

20. С данными глаголами составьте словосочетания, определите, что обозначают переходные глаголы.

1) Лечить, построить, учить, пилить, читать, остановить.

2) Похвалить, сообщить, выяснить, успокоить, выполнить.

3) Забыть, любить, слышать, узнавать, чувствовать.

21. Глаголы «брызгать, метать, двигать, копать» употребляются в двух вариантах. Образуйте 3-е лицо единственного числа (оба варианта) и составьте с тем и другим предложения. Укажите разницу в значении.

22. Найдите и исправьте ошибки в употреблении глаголов.

1) Пальто перевернули на другую сторону, и оно стало выглядеть совсем новым.

2) На дополнительных занятиях учитель интерпретировал трудную задачу.

3) Я оплатил за проезд и начал наблюдать в окно.

4) Стакан упал, и молоко полилось на стол.

5) Я ездию в деревню каждое лето.

23. От следующих глаголов образуйте возможные действительные причастия настоящего и прошедшего времени. Определите, имеется ли между этими формами причастий различие в смысловых оттенках.

1) Дымить – дымиться, брызгать – брызгаться, катить – катиться, кружить – кружиться, собирать – собираться, свернуть – свернуться, тянуть – тянуться, светить – светиться.

2) Прощать – прощаться, пробить – пробиться, сражать – сражаться, раздать – раздаться, вернуть – вернуться, плести – плестись.

24. Поставьте ударения. Определите вид глагола. Подберите парный глагол другого вида. Образуйте возможные действительные причастия, укажите вид и время.

Осыпаться, рассыпать, сбегать, выкупать, срезать, отрезать.

25. Составьте предложения так, чтобы в одном случае данные слова выступали в значении причастий, а в другом – в значении прилагательных.

Образованный, оживленный, преданный, измученный, испытанный, подтянутый.

26. Употребите нужный падеж и предлог.

1) Он раскаялся (... свой поступок).
2) Все трудящиеся протестуют (... война) во Вьетнаме.
3) Нельзя не преклоняться (... геройство) борцов за свободуу.
4) Артист всегда готовится (... свое выступление).
5) Докладчик привлек внимание (... интересный факт.
6) Докладчик обратил вниманиие (... интересный факт).
7) Бойцы с негодованием отнеслись (... предложение сдаться).
8) Бойцы пришли в негодование (... предложение сдаться).

27. Составьте предложения со следующими словами и словосочетаниями.

Имеется спрос, имеется нужда, верить, быть уверенным, вспомнить, помнить, обратить внимание, сосредоточить внимание.

28. Найдите ошибки в построении предложений и исправьте их.

1) На курортах и санаториях люди укрепляют свое здоровье.
2) В Сибири и Урале начались заморозки.
3) Производственная практика проходит на заводах и колхозах.

29. Раскройте скобки и вставьте нужный союз (а, но).
1) Он (не) трус, ... храбрец.
2) Задача (не) легкая, ... трудная.
3) Было (не) весело, ... скучно.
4) Речка (не) широкая, ... быстрая.
5) Живу (не) далеко, ... высоко.
6) Спал (не) спокойно, ... долго.

30. Замените словосочетания одним словом, используя словообразовательные частицы.
1) Крепость, которой нельзя овладеть.
2) Темнота такая, что ничего нельзя разглядеть.
3) Враги, которых нельзя примирить.
4) Число, которое не делится на два.
5) Желание, которое нельзя преодолеть.
6) Друзья, которые всегда вместе.

II. Справки и пояснения к упражнениям

1. 1) картофель; 2) руководителем; 3)долгожданный день рождения; 4)плацкартой; 5) учителя.

2. При однородных определениях в единственном числе ставится существительное: а) если оно не употребляется во множественном числе; б) если во множественном числе имеет другое значение; в) если определяемые предметы внутренне связаны, терминологически близки; г) если между определениями стоит разделительный, противительный или сопоставительный союз; д) если определения выражены порядковыми числительными или притяжательными местоимениями.

3. 1) -а, -ы; 2) -а; -а. 3) -ами; 4) -ень; -; -. 5) -а; -ами.

4. Все слова с суффиксом -тель. Но в I группе одушевленные существительные со значением лица, во II группе – со значением не лица (обозначают приборы, механизмы, их части, растворы, составы, виды документов); в отглагольных существительных с суффиксом -тель значение уясняется из основы.

5. 2 1 2 3 3 1 1 3 3 1 2 1 2 2 2

6. Дорога, автострада, тропка, шлях, путь.

7. Обитель – слово книжное, жилье – разговорное, каморка – устаревшее, жилище – нейтральное.

8. Здравница, здравица, недоумение, недоразумение, отчаяние, отчаянность.

9. Быстрыми, длинную, знакомой, дубовым, низенькой белой, мне неизвестные, узкая, крутой, частый, неприятная, неподвижная, густая, высокая, вся мокрая, ровной.

10. Образец. Ситцевый бабушкин платок; соломенная Машина шляпка.

11. Образец. Туманный – туманное утоpо, туманные воспоминания, туманное сочинение.

12. Образец. Миловидный, красивый, прекрасный.

13. Образец. Храбрый – трусливый человек; добрый – злой взгляд.

14. «Оба» употребляется с существительными мужского рода, «обе» – женского.

15. Одни мыши шуршали за обоями. Здесь «одни» в значении «только».

16. Его, их, своими, своих.

17. У себя дома, в его кабинет, для себя оружие.

18. Им на это, на то, тому, то.

19. Не в своей тарелке; во все глаза; выйти из себя; нестись на всех парусах; от нечего делать; ни то ни се; на каждом шагу; на все руки мастер.

20. Переходные глаголы первой группы обозначают конкретное действие, второй – различные воздействия, третьей – восприятия.

21. Брызгает (опрыскивает) и брызжет (разбрасывает капли).

Метает (бросает, обшивает) и мечет (направляет, разбрасывает).

Двигает (переставляет, перетаскивает) и движет (побуждает, руководит).

Капает (падает каплями) и каплет (протекает).

22. Перелицевали, объяснил, оплатил проезд, смотреть, пролилось, езжу.

23. Синонимичные пары первого ряда могут совпадать в некоторых значениях, а во втором ряду совершенно разные по значению слова, но сходные по звучанию – это паронимы.

24. Среди данных глаголов имеются омографы, например, выкуп**а**ть, в**ы**купать, поэтому парные глаголы другого вида будут в**ы**купить, купа**т**ь и т.д.

25. Для подтверждения качественного значения омонимов-прилагательных к ним можно прибавить слова, указывающие на степень: самый блестящий ученик, самые испытанные друзья, очень смущенный вид и т.д.

26. Примеры на применение модели управления глагола.

Раскаяться (в чем?), протестовать (против чего?), преклоняться (перед чем?), готовиться (к чему?), привлечь внимание (к чему?), обратить внимание (на что?), отнестись с негодованием (к чему?), прийти в негодование (от чего?).

27. Примеры на применение модели управления глагола.

Иметь спрос (на что?), иметь нужду (в чем?), верить (во что?), быть уверенным (в чем?), вспомнить (про что?), напомнить (о чем?), обратить внимание (на что?), сосредоточить внимание (на чем?).

28. Обратите внимание на предлоги.

29. 1–3 союз а, «не» раздельно; 4–6 союз но, «не» слитно.

30. Неприступная, непроглядная, непримиримые, нечетное, непреодолимое, неразлучные.

А. А. Перфильев

ПОИСКОВАЯ СИСТЕМА С ЭЛЕМЕНТАМИ ЛИНГВИСТИЧЕСКОГО АНАЛИЗА

ВВЕДЕНИЕ

К настоящему времени четко выделились задачи, которые компьютеры пока не умеют решать хорошо. Одной из таких проблемных задач в области информационных технологий и искусственного интеллекта является задача по извлечению информации из текста или, в более широком смысле, задача понимания текста; к ним также можно отнести задачу распознавания образцов текста по смыслу или, более конкретно, задачу эффективного поиска.

Задача эффективного поиска или Интернет-поиска требует вовлечения контекста для ее решения. Если реализация семантического поиска — дело трудоемкое и требующее больших усилий по описанию каждой предметной области и каждого понятия и для описания присущих только этим понятиям характеристик, то использовать поиск с вовлечением контекста можно уже на уровне синтаксиса с грамматикой. Этому и посвящена данная работа.

1. ОБЩИЕ ПРИНЦИПЫ СИСТЕМ, РАБОТАЮЩИХ С ЕСТЕСТВЕННЫМ ЯЗЫКОМ

С одной стороны, довлеет требование эффективности, поэтому, как правило, решение строится индивидуально под каждую конкретную задачу, где требуется извлечение смысла из текста; подходы, инновации и ноу-хау глубоко зашиты в программный код и не переносимы на другие схожие задачи.

Несмотря на глубину анализа, общими, ключевыми моментами остаются понятия образца и структуры. Но и сама структура над текстом строится с использованием образцов.

Любая система, имеющая дело с текстом на естественном языке и производящая анализ этого текста, так или иначе сталкивается с задачей построения структуры над этим текстом. Под структурой может пониматься очень широкий класс сущностей: от тривиальной стилевой разметки документа (с которой имеют дело текстовые процессоры) до сложных семанти-

ческих сетей, отражающих смысл текста (необходимых в задачах искусственного интеллекта).

Итак, пониманием текста можно считать построение над текстом некоторой структуры, представляющей смысл этого текста с точки зрения конкретной задачи. После того, как над текстом построена некая первичная структура, она либо является искомой, либо над ней строится другая, более глубинная структура, и так далее до тех пор, пока не будет достигнута требуемая в данной задаче глубина «понимания» текста. Таким образом, структура текста — это иерархическая система объектов, причем терминальными объектами в этой системе являются некие отрезки текста, например, слова. После того, как эта структура построена, она может анализироваться специальными алгоритмами, преобразовываться, модифицироваться и т.п. Задача построения структуры текста не считается самоцелью, она лишь является составляющей более сложных задач по обработке текста, более того, в ряде случаев она считается вспомогательной, и в описании различных систем ей не уделяется должного внимания. Однако благодаря своей базисности (лежит в основе обработки текста) и универсальности (каждая система обработки текста так или иначе реализует данную стадию) она является важной и актуальной в области информационных технологий и искусственного интеллекта. Построение структуры текста является частью любой системы, так или иначе связанной с обработкой текста на ЕЯ [2].

2. ПОСТАНОВКА ЗАДАЧИ

Итак, задача состоит в том чтобы построить алгоритмы, которые, проникая в структуру текста, смогут вывести адекватную оценку релевантности текста. Важно чтобы данная оценка выводилась основываясь на контексте поискового запроса а не ограничивалась только ключевыми словами.

3. МЕТАПОИСКОВАЯ СИСТЕМА INETFINDER

В рамках дипломной работы была создан поисковый бот (iNetFineder an Internet crawler) который автоматизирует работу по поиску информации в сети. Работа сводится к тому чтобы отдать запрос программе и дождаться когда она закончит поиск и сбор информации из сети. При завершении она предложит просмотреть результаты поиска.

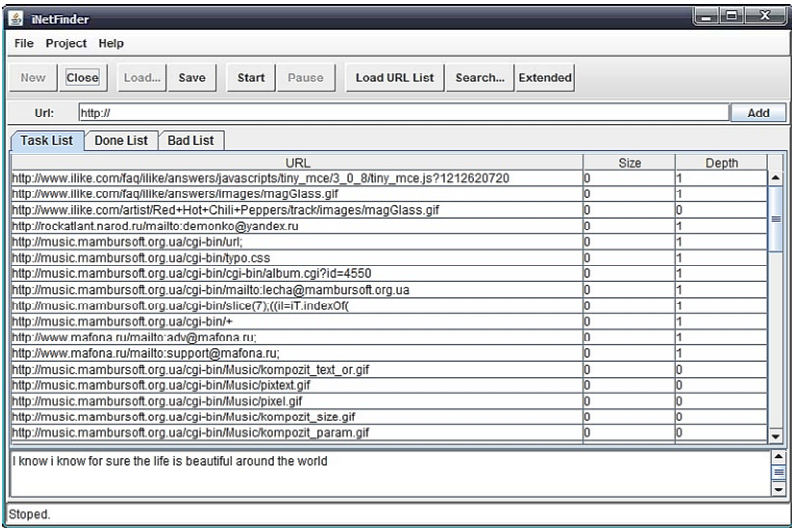


Рис. 1. Скриншот рабочего окна поисковой системы iNetFinder

Особенности ИПС

1. Система находится на стороне пользователя, требует подключения к сети Интернет.
2. Использует результаты запросов к существующим поисковым системам, таким как Yandex, Google. Корректирует и уточняет их.
3. Метаинформации недостаточно. Важна семантическая связность текста. Система просматривает текстовое содержимое Интернет-страничек как базу поиска. Если источник не содержит текста соответствующего запросу — он отбрасывается.

ИПС может работать в нескольких режимах:

- целенаправленная загрузка списка введенных Интернет-адресов, поиск информации соответствующей запросу;
- отправка запроса поисковой системе, получение списка Интернет-адресов, просмотр этого списка, поиск информации соответствующей, запросу;
- рекурсивный просмотр каталога, просмотр файлов, поиск информации соответствующей, запросу;

- целенаправленная загрузка файлов указанных типов с сайта.

Процесс закачки Интернет-страничек предполагает ряд действий.

1. Отправка запроса поисковой системе, разбор полученного гипертекста, пополнение списка ссылок.
2. Загрузка содержимого Интернет-страничек из списка.
3. Просмотр гипертекста поиск и сбор ссылок.
4. Сбор информации, удовлетворяющей запросу пользователя.

4. ПОИСКОВОЕ ЯДРО СИСТЕМЫ INETFINDER

От пользователя в систему поступает текстовый запрос, где из него выделяются ключевые слова и термины. Пополнение запроса с помощью *синонимов* и *гипонимов* (слов с более узким значением) дает более широкий круг поиска слов. При недостаточном количестве найденных документов, поисковый запрос повторяется с привлечением *гиперонимов* (слов с более общим смыслом, например *собака — зверь*). Что значительно расширяет область поиска.

Базой поиска в системе iNetFinder служит текстовое содержимое Интеренет-страничек, которые приходят от встроенного менеджера загрузок. Далее текстовые образцы поступают в систему первичных фильтров, где проходит первичная оценка релевантности текста. Наличие соответствующих ключевых слов говорит о возможной релевантности рассматриваемых образцов. Первичные фильтры ограничивают работу высокопроизводительных алгоритмов синтаксического анализатора, что значительно способствует ускорению работы.

Получаемые на входе предложения транслируются в синтаксические диаграммы. Транслятор проводит лемматизацию слов, приписывание метаинформации словам. Добавление синтаксических связей между словами, типизация этих связей. Приписывание зависимостей между придаточными предложениями. Совокупность этих особенностей дают достаточную информацию о предложении.

Синтаксический анализатор генерирует диаграммы синтаксического разбора, которые используются в системе. Они отображают синтаксическую взаимосвязь между словами.

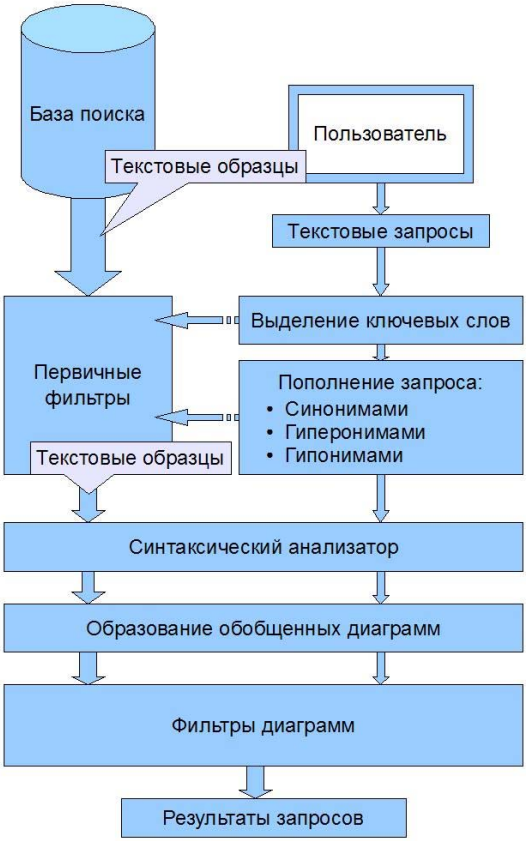


Рис. 2. Схема поискового ядра системы iNetFinder

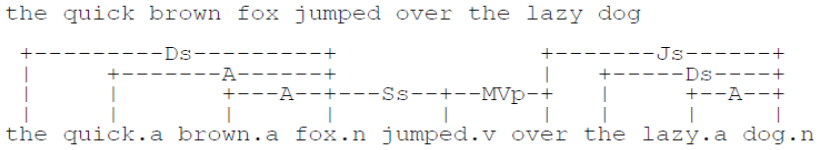
Несколько слов о синтаксическом анализаторе, который был использован в системе. Link Grammar Parser (далее Link) – это синтаксический *парсер* английского языка, основанный на грамматических связях из теории синтаксиса Английского языка. Получив предложение, система приписывает к нему синтаксическую структуру, которая состоит из множества помеченных связей, соединяющих пары слов. Синтаксический анализатор (далее

разборщик или парсер) также создает представление предложения. Link реализован на языке Си и для Unix и Windows, имеет открытый код, распространяется по лицензии совместимой с GNU GPL.

Парсер имеет словарь, включающий около 60000 словарных форм. Он охватывает огромную долю синтаксических конструкций, включая многочисленные редкие выражения и идиомы. Парсер довольно устойчив; он может пропустить часть предложения, которую не может понять и определить некоторую структуру оставшейся части предложения. Он способен обработать неизвестный лексикон, и делать разумные предположения из контекста и написания о синтаксической категории неизвестных слов. У него есть данные насчет различных названий, числовых выражений, и разнообразных знаков препинания.[3]

Внутри парсер использует методы динамического программирования для сопоставления связей между словами. В настоящее время данное программное средство является наиболее перспективным инструментом по работе с текстом.

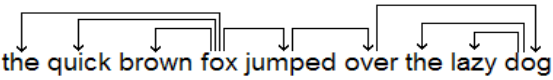
Пример разбора предложения:



- Текстовыми средствами изображаются связи между словами. Каждая связь соединяет только два слова. Скрещивающихся связей нет.
- Каждая связь имеет свою подпись. Например: A ("attributive") – определение
- К некоторым словам добавлены пометки (.n – noun, .a – adverb, .v – verb,) Они указывают на трактовку слов парсером (могут быть различные варианты). Разборщик показывает только те связи, которые найдет в предложении.
- Возможны случаи, когда разборщик выдает несколько вариантов диаграмм, не выдает вообще. Иногда парсер пропускает некоторые слова (для них он не находит грамматических связей).

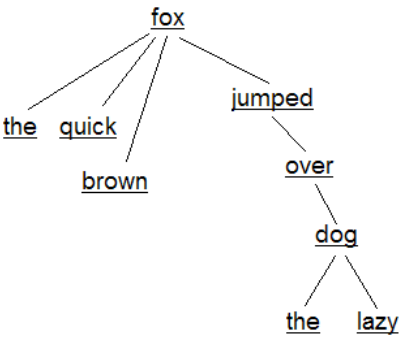
Получаемые диаграммы по сути являются аналогом деревьев подчинения в предложении. В деревьях подчинения от главного слова в предложении

можно задать вопрос к второстепенному. Таким образом слова выстраиваются в древовидную структуру.



Сущность такой взаимосвязи может нести чисто синтаксический характер, но также несет и смысловую компоненту. Если рассматривать языки программирования то они несут сценарную компоненту: это набор объектов, их поля и методы, а также то что нужно с ним и сделать. Разговорные языки несут в себе другие компоненты: смысловую, описательную. В общем случае, чтобы понимать естественный язык требуется установить отображение между объектами речи (в данном случае словами, словосочетаниями, идиомами) и объектами физического и умопостигаемого мира. Важно отметить, что для успешного поиска текста в предложении не обязательно полностью понимать предложение. Вполне достаточно знать как можно перефразировать словосочетания или сами предложения.

Итак, у нас есть дерево разбора. Дальше происходит обобщение таких деревьев. На этом этапе происходит нормализация словоформ. Обратный порядок слов заменяется на прямой. Пассивные формы переделываются (если возможно) в активные. Сложные формы глаголов обрезаются до простой формы. Глаголы переводятся в одну нормализованную форму в настоящем времени в простом виде. Сложные комбинации предлогов объединяются. В результате получается остов дерева, в котором удалены речеобразовательные конструкции. Такие деревья проходят процесс сравнения с диаграммой запроса пользователя.



Фильтрация диаграмм. Перед сличением слова проходят простой фильтр на словоформу – было бы глупо считать глагол и существительное одинаковым словом. Само сличение или наложение слов происходит просто: проверяются гипотезы на соответствие двух слов по набору правил, если все правила проверены, и соответствия не выявлено, то слова считаются далекими по смыслу. Набор правил представляет собой условия, при

которых мы можем считать слова близкими. Туда входят такие правила как прямое равенство слов, совпадение с точностью до окончания, синонимическая близость слов, наличие отношения гипоним-гипероним, слова с трансмутациями и прочие возможные близости между словами.

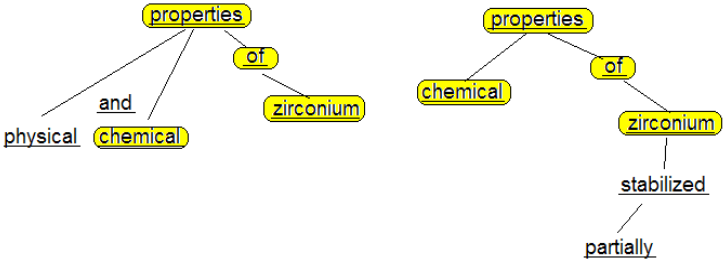


Рис. 3. Пример наложения двух деревьев

Алгоритм сличения выражений, напоминает работу конечного автомата работающего на узлах дерева. По словосочетанию строится конечный автомат. Если в тексте находятся словосочетания удовлетворяющие условиям, это зависит от набора используемых приемов: то автомат переводит свои головки в очередные состояния там где условия удовлетворены. Если хоть одна фраза привела автомат в конечное состояние, значит это выражение из набора и оно релевантно. Количество головок автомата зависит от набора проверяемых схем. Использование автоматов существенно ускоряет обработку выражений. Особенно в сочетании с быстрым синтаксическим анализатором – это делает поиск текста очень быстрым. Тем более, что большинство предложений фильтруются на начальном этапе [1].

Таким образом, приведенная степень оценок позволяет ввести определенную меру близости на предложениях. Она учитывает связь между словами, поиск по словосочетаниям, а также связность слов.

Предложения прошедшие последний фильтр считаются релевантными и выдаются пользователю. Для завершения своей работы система iNetFinder формировала аннотацию из найденного релевантного текста.

5. ДОПОЛНИТЕЛЬНЫЕ ВОЗМОЖНОСТИ

Помимо этого, возможно формировать конкретные запросы к закачанному содержимому. Например, можно выяснить всевозможные определения, появляющиеся с каким то химическим элементом, и таким образом выяснить его химические и физические свойства.

Типичные приемы:

1) Отбор словосочетаний. Имеется словосочетание: существительное и несколько прилагательных. Ищем словосочетания, в которых присутствует то же существительное и частичный набор прилагательных.

2) Подлежащее-сказуемое. Имеется подлежащее и сказуемое или просто глагол и существительное. Строим всевозможные времена глагола, форму в пассивном залоге, также герундий. Если в просматриваемом тексте находятся похожие конструкции, то подбираем их.

3) Преобразование аббревиатур, идиом, поиск синонимов. Искомое словосочетание пополняем всевозможными сокращениями, аббревиатурами, синонимами. Если в просматриваемом тексте находятся похожие конструкции, то подбираем их.

Нечеткий поиск слов и исправление ошибок в словах:

Режим *нечеткого* поиска позволяет найти документы, которые содержат слова, похожие по написанию на слова запроса — например, слова с опечатками: вкрапления, пропуски букв, перестановка рядомстоящих букв, замена символа на неправильный, перепутанная раскладка клавиатуры, просторечные выражения, сокращения, транслитерация и пр. Режим также включает в себя исправление слов написанных похожими символами из других языков и специальными символами, такие приемы обычно используются хакерами для маскировки слов.

Типы ошибок в словах:

1. приставка/суффикс;
2. вкрапление/вакансия;
3. перестановка пары рядом стоящих букв;
4. замена символа на другой символ;
5. “мутации” замены символами из другого набора;
6. “мутации” при рисовании специальными символами;
7. перепутанная раскладка клавиатуры;
8. просторечные выражения и сокращения.

6. РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ СИСТЕМЫ INETFINDER

Для демонстрации эффективности работы системы были произведены испытательные загрузки с помощью данной системы. Были сформированы десять простых запросов из области неорганической химии. По каждому запросу произведены загрузки и поиск релевантного запросу текста. Для сравнения с поисковой системой nigma.ru в таблице можно увидеть результаты запросов. Первая колонка — количество ссылок, выданных системой nigma.ru; из них iNetFinder выбрала некоторое число (вторая колонка) действительно релевантных ссылок. Третья и четвертая колонки — это ошибки системы iNetFinder; уровень ошибок около 5%.

Запрос	Всего ссылок получено от поисковой системы	Количество релевантных ссылок, одобренных системой	Количество релевантных ссылок, пропущенных системой	Количество не релевантных ссылок, одобренных системой
the burning rate of rocket fuels	99	15	8	1
using of liquid nitrogen	85	29	2	0
physical and chemical properties of zirconium	96	8	2	9
raw material for produce of medicine	121	26	7	9
use of zirconium in medicine	97	9	1	1
harmful influence of strontium on a man	102	6	0	0
molecular structure of products of diseintegration of alcohol	85	20	1	12

ways of getting glycerin	89	3	2	0
physical properties of oxides	95	17	4	8
classifying of separation techniques	107	10	0	1

ЗАКЛЮЧЕНИЕ

Когда реализация сложных семантических поисковых запросов очень сложна сама по себе ввиду необходимости проникать вглубь в каждое понятие, обладать необходимым набором знаний о нем, и даже в этом случае система все еще остается “туповатой”. Это является следствием огромной выразительной силы естественного языка. В то же время данная система является не единственной возможностью по улучшению поиска, что в свою очередь, свидетельствует о необходимости развития данного направления. Также, несмотря на примитивность алгоритмов применяемых в прототипе iNetFinder, система показала довольно хорошие результаты, что свидетельствует о перспективности развития данной системы.

СПИСОК ЛИТЕРАТУРЫ

1. **Батура Т.В., Корда О.В., Мурзин Ф.А.** Исследовательская система для анализа текстов на естественном языке // Методы и инструменты конструирования и оптимизации программ. – Новосибирск, 2005. – С. 7–21.
2. **Шевченко М.И.** Технология построения многовариантных объектно-ориентированных структур текстов // Диссертационная работа на соискание ученой степени кандидата физико-математических наук. – МГУ, Москва, 2005. – С. 5–21.
3. **Daniel D.K.Sleator, David Temperly** Parsing English with a Link Grammar // School of Computer Studies. – Carnegie-Melon University, Pittsburg, PA, 1991.

С. А. Полетаев

ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛЕНИЯ НА ГРАФИЧЕСКИХ ПРОЦЕССОРАХ

ВВЕДЕНИЕ

Видеочипы в параллельных математических расчётах пытались использовать довольно давно. Самые первые попытки такого применения были крайне примитивными и ограничивались использованием некоторых аппаратных функций, таких, как растеризация и Z-буферизация. Но в нынешнем веке, с появлением шейдеров, появилась необходимость ускорять вычисления матриц. В 2003 году на SIGGRAPH отдельная секция была выделена под вычисления на GPU, и она получила название GPGPU (General-Purpose Computation on GPU) – универсальные вычисления на GPU).

Наиболее известен BrookGPU – компилятор потокового языка программирования Brook, созданный для выполнения неграфических вычислений на GPU. До его появления разработчики, использующие возможности видеочипов для вычислений, выбирали один из двух распространённых API: Direct3D или OpenGL. Это серьёзно ограничивало применение GPU, ведь в 3D графике используются шейдеры и текстуры, о которых специалисты по параллельному программированию знать не обязаны, они используют потоки и ядра. Brook: смог помочь в облегчении их задачи. Эти потоковые расширения к языку C, разработанные в Стэндфордском университете, скрывали от программистов трёхмерный API, и представляли видеочип в виде параллельного сопроцессора. Компилятор обрабатывал файл .br с кодом C++ и расширениями, производя код, привязанный к библиотеке с поддержкой DirectX, OpenGL или x86.

Естественно, у Brook было множество недостатков, о которых мы подробнее поговорим далее. Но даже просто его появление вызвало значительный прилив внимания тех же NVIDIA и ATI к инициативе вычислений на GPU, так как развитие этих возможностей серьёзно изменило рынок в дальнейшем, открыв целый новый его сектор – параллельные вычислители на основе видеочипов.

В дальнейшем некоторые исследователи из проекта Brook влились в команду разработчиков NVIDIA, чтобы представить программно-аппаратную стратегию параллельных вычислений, открыв новую долю

рынка. И главным преимуществом этой инициативы NVIDIA стало то, что разработчики детально знают все возможности своих GPU, и в использовании графического API нет необходимости, а работать с аппаратным обеспечением можно напрямую при помощи драйвера. Результатом усилий этой команды стала NVIDIA CUDA (Compute Unified Device Architecture) – новая программно-аппаратная архитектура для параллельных вычислений на NVIDIA GPU, которой посвящена эта статья.

1. РАЗНИЦА МЕЖДУ CPU И GPU В ПАРАЛЛЕЛЬНЫХ РАСЧЁТАХ

Рост частот универсальных процессоров упёрся в физические ограничения и высокое энергопотребление, и увеличение их производительности всё чаще происходит за счёт размещения нескольких ядер в одном чипе. Продаваемые сейчас процессоры содержат лишь до четырёх ядер (дальнейший рост не будет быстрым) и они предназначены для обычных приложений, используют MIMD – множественный поток команд и данных. Каждое ядро работает отдельно от остальных, исполняя разные инструкции для разных процессов.

Специализированные векторные возможности (SSE2 и SSE3) для четырехкомпонентных (одинарная точность вычислений с плавающей точкой) и двухкомпонентных (двойная точность) векторов появились в универсальных процессорах, из-за возросших требований графических приложений, в первую очередь. Именно поэтому для определённых задач применение GPU выгоднее, ведь они изначально сделаны для них.

Например, в видеочипах NVIDIA основной блок – это мультипроцессор с восемью-десятью ядрами и сотнями ALU в целом, несколькими тысячами регистров и небольшим количеством разделяемой общей памяти. Кроме того, видеокарта содержит быструю глобальную память с доступом к ней всех мультипроцессоров, локальную память в каждом мультипроцессоре, а также специальную память для констант.

Самое главное – эти несколько ядер мультипроцессора в GPU являются SIMD (одиночный поток команд, множество потоков данных) ядрами. И эти ядра исполняют одни и те же инструкции одновременно. Такой стиль программирования является обычным для графических алгоритмов и многих научных задач, но требует специфического программирования. Зато такой подход позволяет увеличить количество исполнительных блоков за счёт их упрощения.

Итак, перечислим основные различия между архитектурами CPU и GPU. Ядра CPU созданы для исполнения одного потока последовательных инструкций с максимальной производительностью, а GPU проектируются для быстрого исполнения большого числа параллельно выполняемых потоков инструкций. Универсальные процессоры оптимизированы для достижения высокой производительности единственного потока команд, обрабатывающего и целые числа и числа с плавающей точкой. При этом доступ к памяти случайный.

Разработчики CPU стараются добиться выполнения как можно большего числа инструкций параллельно, для увеличения производительности. Для этого, начиная с процессоров Intel Pentium, появилось суперскалярное выполнение, обеспечивающее выполнение двух инструкций за такт, а Pentium Pro отличился внеочередным выполнением инструкций. Но у параллельного выполнения последовательного потока инструкций есть определённые базовые ограничения, и увеличением количества исполнительных блоков кратного увеличения скорости не добиться.

У видеочипов работа простая и распараллеленная изначально. Видеочип принимает на входе группу полигонов, проводит все необходимые операции, и на выходе выдаёт пиксели. Обработка полигонов и пикселей независима, их можно обрабатывать параллельно, отдельно друг от друга. Поэтому из-за изначально параллельной организации работы в GPU используется большое количество исполнительных блоков, которые легко загрузить, в отличие от последовательного потока инструкций для CPU. Кроме того, современные GPU также могут исполнять больше одной инструкции за такт (dual issue). Так, архитектура Tesla в некоторых условиях запускает на исполнение операции MAD+MUL или MAD+SFU одновременно.

GPU отличается от CPU и по принципам доступа к памяти. В GPU он связанный и легко предсказуемый – если из памяти читается тексель текстуры, то через некоторое время придёт время и для соседних текселей. При записи происходит то же самое – пиксель записывается во фреймбуфер, и через несколько тактов будет записываться пиксель расположенный рядом с ним. Поэтому организация памяти отличается от той, что используется в CPU. И видеочипу, в отличие от универсальных процессоров, просто не нужна кэш-память большого размера, а для текстур требуются лишь несколько (до 128–256 в нынешних GPU) килобайт.

Да и сама по себе работа с памятью у GPU и CPU несколько отличается. Так, не все центральные процессоры имеют встроенные контроллеры памяти, а у всех GPU обычно есть по несколько контроллеров, вплоть до восьми 64-битных каналов в чипе NVIDIA GT200. Кроме того, на видеокартах при-

меняется более быстрая память, и в результате видеочипам доступна в разы большая пропускная способность памяти, что также весьма важно для параллельных расчётов, оперирующих с огромными потоками данных.

В универсальных процессорах большие количества транзисторов и площадь чипа идут на буферы команд, аппаратное предсказание ветвления и огромные объёмы начиповой кэш-памяти. Все эти аппаратные блоки нужны для ускорения исполнения немногочисленных потоков команд. Видеочипы тратят транзисторы на массивы исполнительных блоков, управляющие потоками блоки, разделяемую память небольшого объёма и контроллеры памяти на несколько каналов. Вышеперечисленное не ускоряет выполнение отдельных потоков, но позволяет чипу обрабатывать несколько тысяч потоков, одновременно исполняющихся чипом и требующих высокой пропускной способности памяти.

Рассмотрим отличия в кэшировании. Универсальные центральные процессоры используют кэш-память для увеличения производительности за счёт снижения задержек доступа к памяти, а GPU используют кэш или общую память для увеличения полосы пропускания. CPU снижают задержки доступа к памяти при помощи кэш-памяти большого размера, а также предсказания ветвлений кода. Эти аппаратные части занимают большую часть площади чипа и потребляют много энергии. Видеочипы обходят проблему задержек доступа к памяти при помощи одновременного исполнения тысяч потоков – в то время, когда один из потоков ожидает данных из памяти, видеочип может выполнять вычисления другого потока без ожидания и задержек.

Есть множество различий и в поддержке многопоточности. CPU исполняет 1-2 потока вычислений на одно процессорное ядро, а видеочипы могут поддерживать до 1024 потоков на каждый мультипроцессор, которых в чипе несколько штук. И если переключение с одного потока на другой для CPU стоит сотни тактов, то GPU переключает несколько потоков за один такт.

Кроме того, центральные процессоры используют SIMD (одна инструкция выполняется над многочисленными данными) блоки для векторных вычислений, а видеочипы применяют SIMT (одна инструкция и несколько потоков) для скалярной обработки потоков. SIMT не требует, чтобы разработчик преобразовывал данные в векторы, и допускает произвольные ветвления в потоках.

Вкратце можно сказать, что в отличие от современных универсальных CPU, видеочипы предназначены для параллельных вычислений с большим количеством арифметических операций. И значительно большее число

транзисторов GPU (Рис. 1) работает по прямому назначению – обработке массивов данных, а не управляет исполнением (flow control) немногочисленных последовательных вычислительных потоков. Это схема того, сколько места в CPU и GPU занимает разнообразная логика:

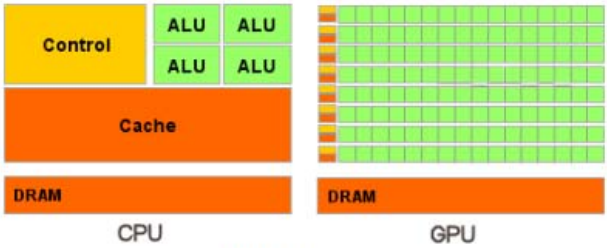


Рисунок 1

В итоге основой для эффективного использования мощи GPU в научных и иных неграфических расчётах является распараллеливание алгоритмов на сотни исполнительных блоков, имеющихся в видеочипах. Например – множество приложений по молекулярному моделированию отлично приспособлено для расчётов на видеочипах, расчеты требуют больших вычислительных мощностей и поэтому удобны для параллельных вычислений. Использование нескольких GPU даёт ещё больше вычислительных мощностей для решения подобных задач.

Выполнение расчётов на GPU показывает отличные результаты в алгоритмах, использующих параллельную обработку данных. То есть, когда одну и ту же последовательность математических операций применяют к большому объёму данных. При этом лучшие результаты достигаются, если отношение числа арифметических инструкций к числу обращений к памяти достаточно велико. Это предъявляет меньшие требования к управлению исполнением (flow control), а высокая плотность математики и большой объём данных отменяет необходимость в больших кэшах, как на CPU.

В результате всех описанных выше отличий, теоретическая производительность видеочипов значительно превосходит производительность CPU. На рис. 2 показан опубликованный компанией NVIDIA приводит такой график роста производительности CPU и GPU за последние несколько лет.

Естественно, эти данные не без доли лукавства. Ведь на CPU гораздо проще на практике достичь теоретических цифр, да и цифры приведены для одинарной точности в случае GPU, и для двойной – в случае CPU. В любом случае, для части параллельных задач одинарной точности хватает, а раз-

ница в скорости между универсальными и графическими процессорами весьма велика.

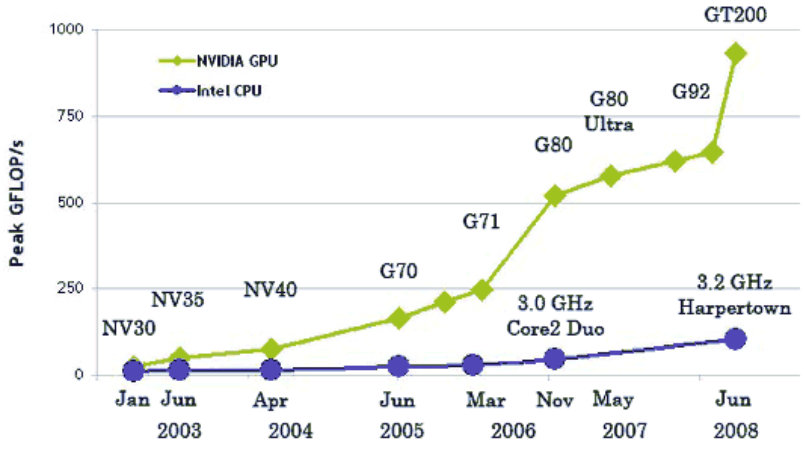


Рисунок 2

2. API CUDA

Успеха Brook оказалось достаточно, чтобы привлечь внимание ATI и NVIDIA, у них зародился интерес к подобной инициативе, поскольку она могла бы расширить рынок, открыв для компаний новый немаловажный сектор.

Исследователи, изначально вовлечённые в проект Brook, быстро присоединились к командам разработчиков в Санта-Кларе, чтобы представить глобальную стратегию для развития нового рынка. Идея заключалась в создании комбинации аппаратного и программного обеспечения, подходящего для задач GPGPU. Поскольку разработчики NVIDIA знают все секреты своих GPU, то на графическое API можно было и не опираться, а связываться с графическим процессором через драйвер. Хотя, конечно, при этом возникают свои проблемы. Итак, команда разработчиков CUDA (Compute Unified Device Architecture) создала набор программных уровней для работы с GPU(рис. 3).

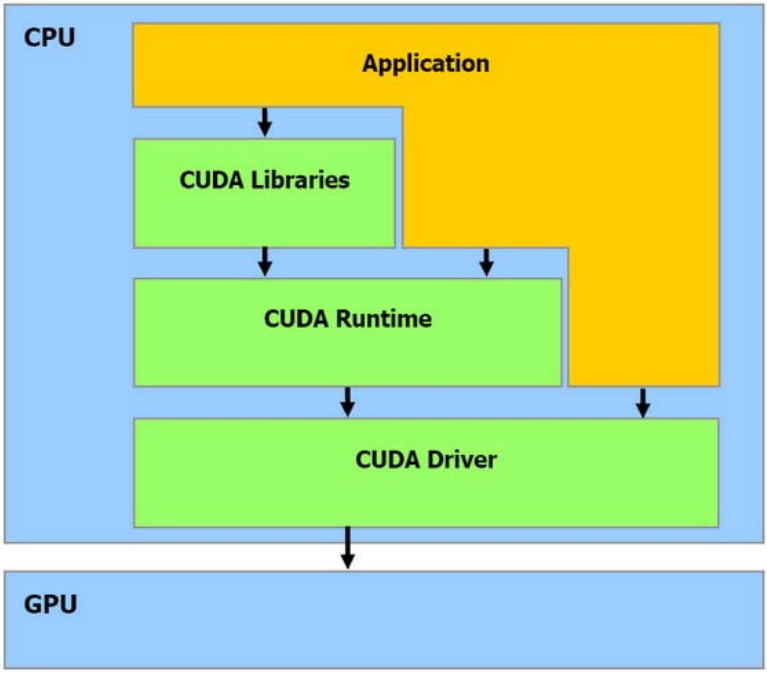


Рисунок 3

Как можно видеть на диаграмме, CUDA обеспечивает два API:

- высокоуровневый API: CUDA Runtime API;
- низкоуровневый API: CUDA Driver API.

Поскольку высокоуровневый API реализован над низкоуровневым, каждый вызов функции уровня Runtime разбивается на более простые инструкции, которые обрабатывает Driver API. Обратите внимание, что два API взаимно исключают друг друга: программист может использовать один или другой API, но смешивать вызовы функций двух API не получится. Вообще, термин «высокоуровневый API» относителен. Даже Runtime API таков, что многие сочтут его низкоуровневым; впрочем, он всё же предоставляет функции, весьма удобные для инициализации или управления контекстом. Но не ожидайте особо высокого уровня абстракции вам всё равно нужно

обладать хорошим набором знаний о NVIDIA GPU и о том, как они работают.

С Driver API работать ещё сложнее; для запуска обработки на GPU вам потребуется больше усилий. С другой стороны, низкоуровневый API более гибок, при необходимости предоставляя программисту дополнительный контроль. Два API способны работать с ресурсами OpenGL или Direct3D (только девятая версия). Польза от такой возможности очевидна CUDA может использоваться для создания ресурсов (геометрия, процедурные текстуры и т.д.), которые можно передать на графическое API или, наоборот, можно сделать так, что 3D API будет отсылать результаты рендеринга программе CUDA, которая, в свою очередь, будет выполнять пост-обработку. Есть много примеров таких взаимодействий, и преимущество заключается в том, что ресурсы продолжают храниться в памяти GPU, их не требуется передавать через шину PCI Express, которая по-прежнему остаётся «узким местом».

Впрочем, следует отметить, что совместное использование ресурсов в видеопамяти не всегда проходит идеально и может привести к некоторым «головным болям». Например, при смене разрешения или глубины цвета, графические данные приоритетны. Поэтому, если требуется увеличить ресурсы в кадровом буфере, то драйвер без проблем сделает это за счёт ресурсов приложений CUDA, которые попросту «вылетят» с ошибкой. Конечно, не очень элегантно, но такая ситуация не должна случаться очень уж часто. Если вы хотите использовать несколько GPU для приложений CUDA, то вам нужно сначала отключить режим SLI, иначе приложения CUDA смогут «видеть» только один GPU.

Наконец, третий программный уровень отдан библиотекам.

- **CUBLAS** – CUDA вариант BLAS (Basic Linear Algebra Subprograms), предназначенный для вычислений задач линейной алгебры и использующий прямой доступ к ресурсам GPU;
- **CUFFT** – CUDA вариант библиотеки Fast Fourier Transform для расчёта быстрого преобразования Фурье, широко используемого при обработке сигналов. Поддерживаются следующие типы преобразований: complex-complex (C2C), real-complex (R2C) и complex-real (C2R).

CUBLAS – это переведённые на язык CUDA стандартные алгоритмы линейной алгебры; на данный момент поддерживается только определённый набор основных функций CUBLAS. Библиотеку очень легко использовать: нужно создать матрицу и векторные объекты в памяти видеокарты, заполнить их данными, вызвать требуемые функции CUBLAS и загрузить

результаты из видеопамяти обратно в системную. CUBLAS содержит специальные функции для создания и уничтожения объектов в памяти GPU, а также для чтения и записи данных в эту память. Поддерживаемые функции BLAS: уровни 1, 2 и 3 для действительных чисел, уровень 1 CGEMM для комплексных. Уровень 1 – это векторно-векторные операции, уровень 2 – векторно-матричные операции, уровень 3 – матрично-матричные операции.

CUFFT – CUDA вариант функции быстрого преобразования Фурье – широко используемой и очень важной при анализе сигналов, фильтрации и т.п. CUFFT предоставляет простой интерфейс для эффективного вычисления FFT на видеочипах производства NVIDIA без необходимости в разработке собственного варианта FFT для GPU. CUDA вариант FFT поддерживает 1D, 2D, и 3D преобразования комплексных и действительных данных, пакетное исполнение для нескольких 1D трансформаций в параллели, размеры 2D и 3D трансформаций могут быть в пределах [2, 16384], для 1D поддерживается размер до 8 миллионов элементов.

3. ФУНДАМЕНТАЛЬНЫЕ ПОНЯТИЯ

Перед тем, как мы погрузимся в CUDA, позвольте определить ряд терминов, разбросанных по документации NVIDIA. Компания выбрала весьма специфическую терминологию, к которой трудно привыкнуть. Прежде всего, отметим, что **поток (thread)** в CUDA имеет далеко не такое же значение, как поток CPU. Потоком GPU в данном случае является базовый набор данных, которые требуется обработать. В отличие от потоков CPU, потоки CUDA очень «лёгкие», то есть переключение контекста между двумя потоками – отнюдь не ресурсоёмкая операция.

Второй термин, часто встречающийся в документации CUDA **варп (warp)**. Термин взят из текстильной промышленности, где через основную пряжу (warp yarn), которая растянута на станке, протягивается уточная пряжа (weft yarn) (рис. 4).

Варп в CUDA представляет собой группу из 32 потоков и является минимальным объёмом данных, обрабатываемых SIMD-способом в мультипроцессорах CUDA.

Рисунок 4

Но подобная «зернистость» не всегда удобна для программиста. Поэтому в CUDA вместо работы с варпами напрямую можно работать с **блоками (block)**, содержащими от 64 до 512 потоков (Рис. 5).

Наконец, эти блоки собираются вместе в **сетки (grid)**. Преимущество подобной группировки заключается в том, что число блоков, одновременно обрабатываемых GPU, тесно связано с аппаратными ресурсами, как мы увидим ниже. Группировка блоков в сетки позволяет полностью абстрагироваться от этого ограничения и применить ядро (kernel) к большему числу потоков за один вызов, не думая о фиксированных ресурсах. За всё это отвечают библиотеки CUDA. Кроме того, подобная модель хорошо масштабируется. Если GPU имеет мало ресурсов, то он будет выполнять блоки последовательно. Если число вычислительных процессоров велико, то блоки могут выполняться параллельно. То есть, один и тот же код может работать на GPU как начального уровня, так и на топовых и даже будущих моделях. Есть ещё пара терминов в CUDA API, которые обозначают CPU (**хост/host**) и GPU (**устройство/device**).

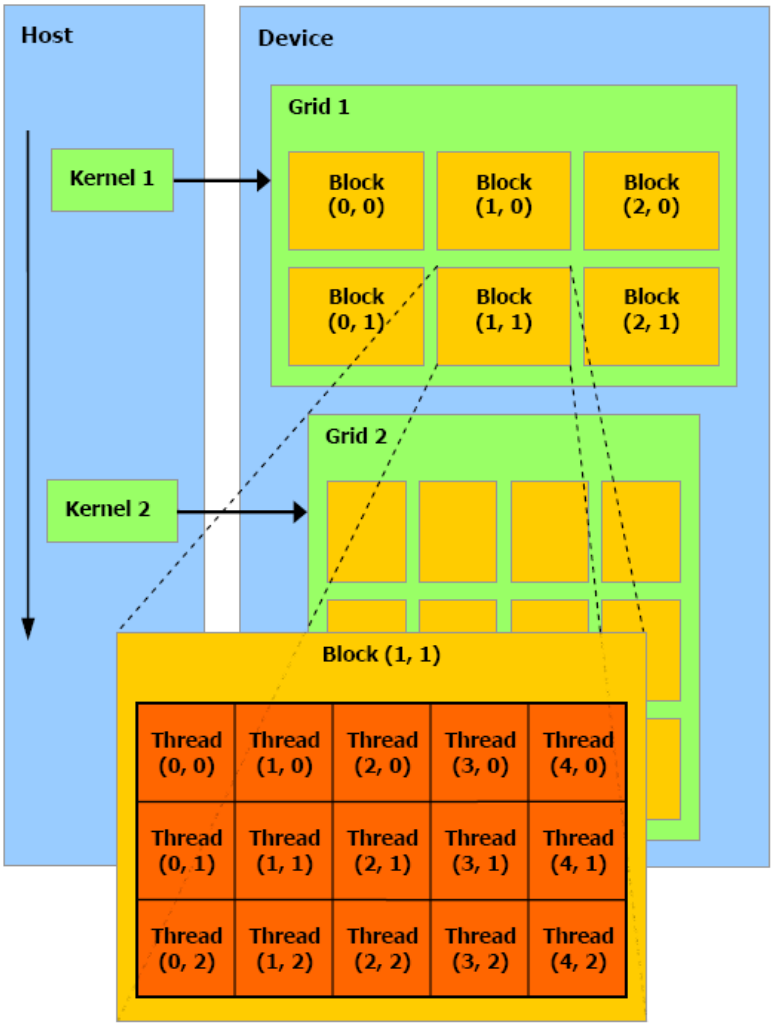

Host
Device
Grid 1
Kernel 1
Block (0, 0)
Block (1, 0)
Block (2, 0)
Block (0, 1)
Block (1, 1)
Block (2, 1)
Grid 2
Kernel 2
Block (1, 1)
Thread (0, 0)
Thread (1, 0)
Thread (2, 0)
Thread (3, 0)
Thread (4, 0)
Thread (0, 1)
Thread (1, 1)
Thread (2, 1)
Thread (3, 1)
Thread (4, 1)
Thread (0, 2)
Thread (1, 2)
Thread (2, 2)
Thread (3, 2)
Thread (4, 2)

Рисунок 5

4. CUDA С АППАРАТНОЙ ТОЧКИ ЗРЕНИЯ

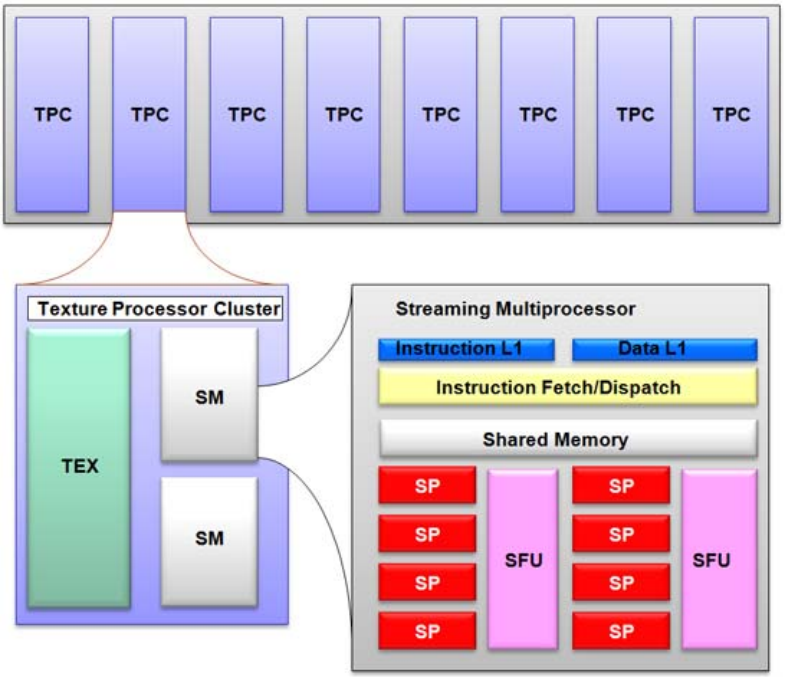


Рисунок 6

Как можно понять по иллюстрации выше на рис. 6, ядро шейдеров NVIDIA состоит из нескольких кластеров текстурных процессоров **(Texture Processor Cluster, TPC)**. Видеокарта 8800 GTX, например, использовала восемь кластеров, 8800 GTS шесть и т.д. Каждый кластер, по сути, состоит из текстурного блока и двух **потоковых мультипроцессоров (streaming multiprocessor)**. Последние включают начало конвейера (front end), выполняющее чтение и декодирование инструкций, а также отсылку их на выполнение, и конец конвейера (back end), состоящий из восьми вычислительных устройств и двух суперфункциональных устройств **SFU (Super Function Unit)**, где инструкции выполняются по принципу SIMD, то есть одна инструкция применяется ко всем потокам в варпе. NVIDIA называет такой способ выполнения **SIMT** (single instruction multiple threads, одна инструкция, много потоков). Важно отметить, что конец конвейера работает на частоте в два раза превосходящей его начало. На практике это означа-

ет, что данная часть выглядит в два раза «шире», чем она есть на самом деле (то есть как 16-канальный блок SIMD вместо восьмиканального). Потоковые мультипроцессоры работают следующим образом: каждый такт начало конвейера выбирает варп, готовый к выполнению, и запускает выполнение инструкции. Чтобы инструкция применилась ко всем 32 потокам в варпе, концу конвейера потребуется четыре такта, но поскольку он работает на удвоенной частоте по сравнению с началом, потребуется только два такта (с точки зрения начала конвейера). Поэтому, чтобы начало конвейера не простаивало такт, а аппаратное обеспечение было максимально загружено, в идеальном случае можно чередовать инструкции каждый такт, классическая инструкция в один такт и инструкция для SFU в другой.

Каждый мультипроцессор обладает определённым набором ресурсов, в которых стоит разобраться. Есть небольшая область памяти под названием **«Общая память/Shared Memory»**, по 16 кбайт на мультипроцессор (рис. 7). Это отнюдь не кэш-память: программист может использовать её по своему усмотрению. То есть перед нами что-то близкое к Local Store у SPU на процессорах Cell. Данная деталь весьма любопытная поскольку она подчёркивает, что CUDA это комбинация программных и аппаратных технологий. Данная область памяти не используется для пиксельных шейдеров.

Данная область памяти открывает возможность обмена информацией между потоками *в одном блоке*. Важно подчеркнуть это ограничение: все потоки в блоке гарантированно выполняются одним мультипроцессором. Напротив, привязка блоков к разным мультипроцессорам вообще не оговаривается, и два потока из разных блоков не могут обмениваться информацией между собой во время выполнения. То есть пользоваться общей памятью не так и просто. Впрочем, общая память всё же оправданна за исключением случаев, когда несколько потоков попытаются обратиться к одному банку памяти, вызывая конфликт. В остальных ситуациях доступ к общей памяти такой же быстрый, как и к регистрам.

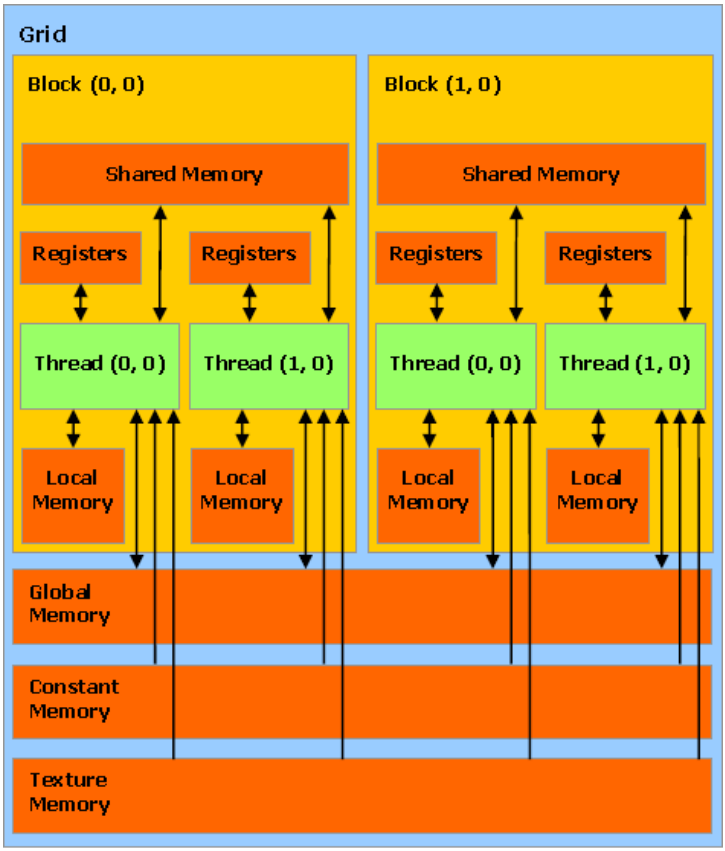

Grid
Block (0, 0)
Block (1, 0)
Shared Memory
Shared Memory
Registers
Registers
Registers
Registers
Thread (0, 0)
Thread (1, 0)
Thread (0, 0)
Thread (1, 0)
Local Memory
Local Memory
Local Memory
Local Memory
Global Memory
Constant Memory
Texture Memory

Рисунок 7

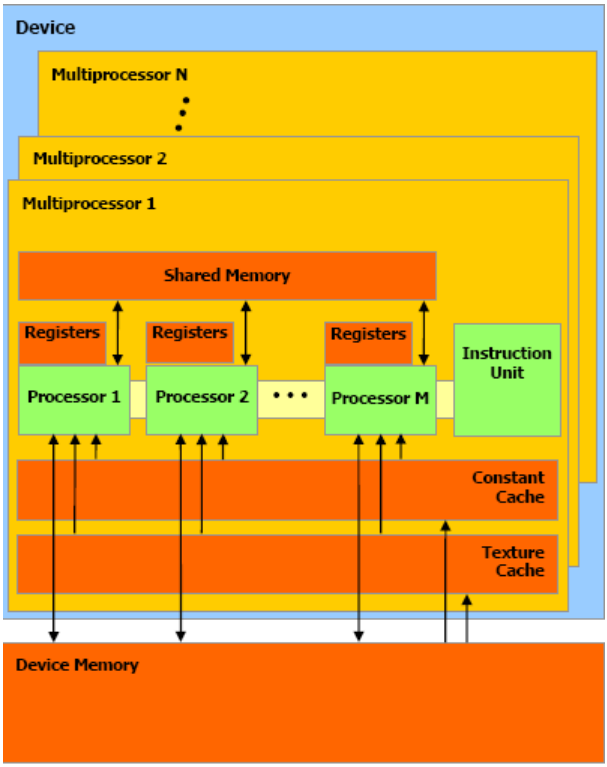


Рисунок 8

Общая память не единственная, к которой могут обращаться мультипроцессоры. Они могут использовать видеопамять, но с меньшей пропускной способностью и большими задержками. Поэтому, чтобы снизить частоту обращения к этой памяти, NVIDIA оснастила мультипроцессоры кэшем (примерно 8 кбайт на мультипроцессор), хранящим константы и текстуры.

Мультипроцессор имеет 8192 регистра, которые общие для всех потоков всех блоков, активных на мультипроцессоре. Число активных блоков на мультипроцессор не может превышать восьми, а число активных варпов ограничено 24 (768 потоков). Поэтому 8800 GTX может обрабатывать до 12 288 потоков в один момент времени. Все эти ограничения стоило упомянуть, поскольку они позволяют оптимизировать алгоритм в зависимости от доступных ресурсов (Рис. 8).

Оптимизация программы CUDA, таким образом, состоит в получении оптимального баланса между количеством блоков и их размером. Больше потоков на блок будут полезны для снижения задержек работы с памятью, но и число регистров, доступных на поток, уменьшается. Более того, блок из 512 потоков будет неэффективен, поскольку на мультипроцессоре может быть активным только один блок, что приведёт к потере 256 потоков. Поэтому NVIDIA рекомендует использовать блоки по 128 или 256 потоков, что даёт оптимальный компромисс между снижением задержек и числом регистров для большинства ядер/kernel.

5. CUDA С ПРОГРАММНОЙ ТОЧКИ ЗРЕНИЯ

С программной точки зрения CUDA состоит из набора расширений к языку C, что напоминает BrookGPU, а также нескольких специфических вызовов API. Среди расширений присутствуют спецификаторы типа, относящиеся к функциям и переменным. Важно запомнить ключевое слово **__global__**, которое, будучи приведённым перед функцией, показывает, что последняя относится к ядру/kernel эту функцию будет вызывать CPU, а выполняться она будет на GPU. Префикс **__device__** указывает, что функция будет выполняться на GPU (который CUDA и называет "устройство/device") но она может быть вызвана только с GPU (иными словами, с другой функции __device__ или с функции __global__). Наконец, префикс **__host__** обозначает функцию, которая вызывается CPU и выполняется CPU - другими словами, обычную функцию.

Есть ряд ограничений, связанных с функциями __device__ и __global__: они не могут быть рекурсивными (то есть вызывать самих себя) и не могут иметь переменное число аргументов. Наконец, поскольку функции __device__ располагаются в пространстве памяти GPU, вполне логично, что получить их адрес не удастся.

API CUDA содержит функции для работы с памятью в VRAM: cudaMalloc для выделения памяти, cudaFree для освобождения и cudaMemcpy для копирования памяти между RAM и VRAM и наоборот.

Компиляция выполняется в несколько этапов (Рис. 9). Сначала извлекается код, относящийся к CPU, который передаётся стандартному компилятору. Код, предназначенный для GPU, сначала преобразовывается в промежуточный язык PTX. Он подобен ассемблеру и позволяет изучать код в поисках потенциальных неэффективных участков. Наконец, последняя фаза

заключается в трансляции промежуточного языка в специфические команды GPU и создании двоичного файла.

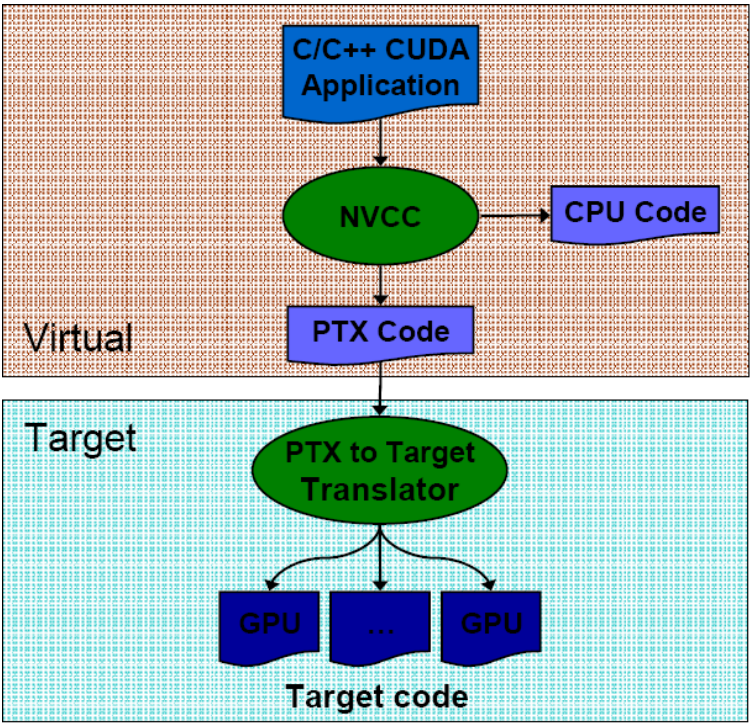


Рисунок 9

Переменные тоже имеют ряд квалификаторов, которые указывают на область памяти, где они будут храниться. Ниже перечислен список предусмотренных CUDA API префиксов:

- **__device__** описывает переменную, создающуюся в памяти GPU.
- **__constant__** опционально используется вместе с **__device__**, объявляет переменную как:
  1. Существующую в константном пространстве памяти
  2. Обладающую временем жизни приложения.
  3. Доступную из всех потоков сетки и с CPU через библиотеки.
- **__shared__** опционально используется вместе с **__device__**, объявляет переменную как:

1. Существующую в памяти блока потоков.
2. Обладающую временем жизни блока потоков.
3. Доступную всем потокам блока.

Только после выполнения **__syncthreads**() записанные данные в **__shared__** переменные будут гарантированно видны другим потокам блока.

Данные определители не могут быть применены к членам структур (**struct**) и объединений (**union**), формальным параметрам и локальным переменным функций которые выполняются на CPU.

- **__shared__** и **__constant__** хранятся статически.
- **__device__**, **__shared__** и **__constant__** не могут быть определены как глобальные переменные, т.е. имеющие идентификатор **extern**.
- **__device__** и **__constant__** действуют только в границах одного файла.
- **__shared__** не может быть объявлено при инициализации.

Переменные, объявляющиеся в функциях, исполняемых на видеоадаптере без данных идентификаторов, обычно хранятся в регистрах. Однако, в некоторых случаях, компилятор может разместить их в локальной памяти потока. Это обычно случается с большими массивами данных занимающих очень много регистровой памяти.

Для любого вызова **__global__** функции должна быть описана исполняемая конфигурация. Она определяет измерения сетки и блоков, которые будут выполнять функцию на устройстве. Задается конфигурирование выражением **<<<Dg, Db, Ns, S >>>** между именем функции и списком аргументов:

- **Dg** имеет тип **dim3** и описывает измерение и размер сетки, такие что **Dg.x * Dg.y** эквивалентны числу блоков которые будут запущены; **Dg.z** не используется.
- **Db** имеет тип **dim3** и описывает измерение и размер каждого блока, такой, что **Db.x * Db.y * Db.z** эквивалентно числу потоков на блок.

- **Ns** имеет типа **size_t** и описывает число байт в общей памяти, которое динамически выделено на блок; эта динамически выделенная память используемая любыми переменными объявленными как внешний массив; **Ns** опциональный аргумент который по умолчанию равен 0.
- **S** имеет **cudaStream_t** тип и описывает поток; **S** опциональный аргумент который по умолчанию равен 0.

Функция, объявленная следующим образом:

__global__ void Func(float* parameter);

Должна вызываться как:

Func<<< Dg, Db[, Ns, S]>>>(parameter);

Аргументы конфигурации вычисляются перед аргументами функции. Вызов функции может завершиться неудачно, если **Db** или **Dg** больше, чем максимальный размер доступный для установленного оборудования или если **Ns** больше чем максимальное количество общей памяти доступной на устройстве минус количество памяти, требуемое для размещения статических объектов, аргументов функций, и конфигураций.

5.1. Предопределенные переменные

- **gridDim** – тип переменной **dim3**, содержит размерность сетки.
- **blockIdx** – тип переменной **uint3**, содержит индекс блока в сетке.
- **blockDim** – тип переменной **dim3**, содержит размерность блока.
- **threadIdx** - тип переменной **uint3**, содержит индекс потока в блоке.
- **warpSize** - тип переменной **int**, содержит размер оболочки в потоках.

Данные переменные не позволяют взятие их адреса и построение указателей и предоставлены только для чтения.

5.2. Предопределенные типы данных

char1, **uchar1**, **char2**, **uchar2**, **char3**, **uchar3**, **char4**, **uchar4**, **short1**, **ushort1**, **short2**, **ushort2**, **short3**, **ushort3**, **short4**, **ushort4**, **int1**, **uint1**, **int2**, **uint2**, **int3**, **uint3**, **int4**, **uint4**, **long1**, **ulong1**, **long2**, **ulong2**, **long3**, **ulong3**, **long4**, **ulong4**, **float1**, **float2**, **float3**, **float4**, **double2 –** все это встроенные типа данных наследованные от фундаментальных аналогичных типов языка С. Они

структурированы по 1, 2, 3 – 4 компонентам по доступности полей **x, y, z, w**.

Их можно конструировать с помощью функции **make<type name>**, например:

int2 make_int2(int x, int y);

данная функция создает вектор типа **int2** со значениями **(x, y).**

5.3. Инициализация CUDA

Перед вызовом любой первой **__global__** функции необходимо инициализировать CUDA устройство. Для этого существуют специальные функции в CUDA API. Такие как: **cudaGetDeviceCount** и **cudaGetDeviceProperties**.

- **cudaGetDeviceCount(int *deviceCount)** – возвращает количество устройств поддерживающих технологию CUDA.
- **cudaGetDeviceProperties(cudaDeviceProp devProp, int devIndex)** – позволяет получить свойства устройства с индексом devIndex, в виде заполненной структуры cudaDeviceProp.
- **cudaSetDevice(int device) –** используется для выбора устройства связанного с данным потоком CPU.

Таким образом, простейшая процедура инициализации CUDA будет выглядеть следующим образом:

```
bool InitCUDA(void)
{
  int count = 0;
  int i = 0;
  cudaGetDeviceCount(&count);
  if(count == 0) {
        fprintf(stderr, " CUDA  устройств нет.\n");
        return false;
  }
  for(i = 0; i < count; i++) {
        cudaDeviceProp prop;
        if(cudaGetDeviceProperties(&prop, i) == cudaSuccess) {
```

```
                if(prop.major >= 1) {
                    break;
                }
            }
        }
        if(i == count) {
            fprintf(stderr, " CUDA  устройств нет.\n");
            return false;
        }
        cudaSetDevice(i);
        printf("CUDA initialized.\n");
        return true;
    }
```

В данном листинге мы видим, что с помощью функции **cudaGetDeviceCount** мы получаем количество доступных поддерживающих технологию CUDA устройств и проверяем свойства каждого устройства на пример **major** версии. Как только мы нашли устройство с версией больше либо равной единице устанавливаем его как активное, для данного потока, с помощью функции **cudaSetDevice**, иначе, если таких устройств нет, выдаем оповещение, что нет устройств, поддерживающих CUDA, и возвращаем из функции значение **false**.

Мажорная и минорная версии обозначают вычислительную мощность устройства. Устройства с мажорными версиями имеют одинаковую архитектуру ядра. Минорная версия соответствует возрастанию улучшений ядра, добавлению новых особенностей и свойств.

5.4. Управление памятью

Как уже упоминалось выше, для выделения памяти на устройстве можно использовать функцию **cudaMalloc**, а для освобождения - **cudaFree**. В данной главе мы рассмотрим еще некоторые очень полезные функции для управления памятью. Первая из них это **cudaMallocPitch(void **devPtr, int *pitch, size_t width, size_t height)** – функция рекомендуется для выделения памяти под двумерный массив для того, чтобы память была соответствующим образом выровнена. Следовательно, будет уверенность в улучшении производительности при доступе к памяти и при копировании между двумерными массивами или другими регионами памяти устройства. Следую-

щий код пример демонстрирует выделение памяти под двумерный массив размерности **width*heigh** и показывает, как перебрать все элементы массива:

```
//Код, выполняемый на хосте(CPU)
float* devPtr;
int pitch;
cudaMallocPitch((void**)&devPtr, &pitch, width * sizeof(float), height);

myKernel<<<100, 512>>>(devPtr, pitch);
```

```
//Код, выполняемый на GPU
__global__ void myKernel(float* devPtr, int pitch)
{
    for (int r = 0; r < height; ++r)
    {
        float* row = (float*)((char*)devPtr + r * pitch);
        for (int c = 0; c < width; ++c)
            float element = row[c];
    }
}
```

Под массивы память можно выделить функцией **cudaMallocArray** и освободить функцией **cudaFreeArray**. **cudaMallocArray** требует описание формата созданного массива созданное функцией **cudaCreateChannelDesc.**

Следующий код, выделяет память для массива размером **width×height** - содержащего 32 битные числа с плавающей точкой:

```
cudaChannelFormatDesc channelDesc = cudaCreateChannelDesc<float>();
cudaArray* cuArray;
cudaMallocArray(&cuArray, &channelDesc, width, height);
```

Функция **cudaGetSymbolAddress** используется, чтобы получить адрес переменной расположенной в глобальной памяти. Размер выделенной памяти доступен через функцию **cudaGetSymbolAddress.**

Существуют различные функции для копирования регионов памяти, например, для копирования двумерного массива, для массива показанного в предыдущем примере используется следующая функция:

```
cudaMemcpy2DToArray(cuArray, 0, 0, devPtr, pitch, width * sizeof(float), height, cudaMemcpyDeviceToDevice);
```

Чтобы скопировать в константную память из памяти, хоста нужно сделать следующее:

```
__constant__ float constData[256];
float data[256];
cudaMemcpyToSymbol(constData, data, sizeof(data));
```

Каждый поток копирует свою порцию данных входного массива **hostPtr** в массив **inputDevPtr** в памяти устройства, **inputDevPtr** обрабатывается на устройстве посредством вызова **myKernel**, и копирует результат **outputDevPtr** назад в **hostPtr**.

6. УПРАВЛЕНИЕ ПРОИЗВОДИТЕЛЬНОСТЬЮ

Для того, чтобы обработать инструкцию для варпа потоков, мультипроцессор должен:

- прочитать операнды инструкций для каждого потока варпа;
- выполнить инструкцию;
- записать результат для каждого потока варпа.

Следовательно, эффективная производительность инструкций зависит от номинальной производительности, так же как задержка памяти и пропускная способность. Производительность можно максимизировать посредством:

- минимизации использования инструкций с низкой производительностью,

- максимизации использования доступной полосы пропускания памяти для каждой категории памяти.
- позволения планировщику памяти перекрывать транзакции памяти с математическими вычислениями настолько, насколько это возможно, требуют чтобы:
  - программа, выполняющаяся посредством потоков с высокой арифметической интенсивностью, имела высокое число арифметических операций на одну операцию с памятью;
  - существовало много активных потоков на один мультипроцессор.

1. Производительность инструкций.

   Чтобы выдать одну инструкцию, мультипроцессор занимает:
   - 4 такта для:
     - операций добавления с одинарной точностью и плавающей точкой, умножения, и многократного добавления,
     - целочисленного добавления,
     - битовых операций, сравнения, инструкций преобразования типов.
   - 16 тактов для обратных операций, квадратного корня, **__logf(x).**

32 – битное целочисленное умножение занимает 16 тактов, но **__mul24** и **__umul24** обеспечивает знаковое и беззнаковое 24-битное умножение в 4 такта. На будущих архитектурах, однако, **__[u]mul24** будет медленнее, чем 32- битное целочисленное умножение, рекомендуется обеспечить два ядра («**kernels**»), одно использует **__[u]mul24**, а другое общее 32-битное умножение и вызывать их соответствующим образом.

Целочисленное деление и операция взятия модуля являются очень дорогостоящими и должны быть обойдены, если это возможно, либо заменены битовыми операциями: если n –степень двойки, тогда **i/n** эквивалентно **i >> log2(n)** и **i % n** эквивалентно **(i & (n - 1))**. Другие функции занимают больше тактов, так как они являются комбинацией нескольких инструкций.

__sinf(x), **__cosf(x)**, **__expf(x)** занимают 32 такта, **sinf(x)**, **cosf(x)**, **tanf(x), sincosf(x)** являются более дорогостоящими операциями, если абсолютное значение **x** больше чем 48039. Более того, в этом случае, для уменьшения аргумента код использует локальную память, которая может влиять на производительность больше в силу медленного доступа и малой пропускной способности.

Иногда компилятор может вставлять преобразовывающие инструкции, которые уменьшают количество дополнительных циклов. Это случается, когда:

- Функция оперирует на **char** или **short,** чьи операнды, как правило, должны быть преобразованы к **int**.
- Константы с плавающей точкой имеющие двойную точность(определенные без каких либо типовых суффиксов) используются как ввод для вычислений с плавающей точкой имеющих одинарную точность.

  Последнего пункта можно избежать использованием:
- Констант с плавающей точкой одинарной точности, определенных с суффиксом **f,** таких как **3.141592653589793f, 1.0f, 0.5f.**
- Версий функций с одинарной точностью, имеющих суффикс **f – sinf(), logf(), expf()**.

Для операций с одинарной точностью строго рекомендуется использовать тип **float** и версии функций с одинарной точностью.

2. Контроль потока инструкций

Любой поток управляющих инструкций (**if, switch, do, for, while**) может значительно ударить по эффективности производительности инструкций посредством расщепления потоков в одном варпе, что приводит к различным путям выполнения. Если это случилось, потоки будут упорядочены, что вызовет выполнение дополнительных инструкций и затраты времени. Чтобы получить лучшую производительность в случае, когда контроль потока зависит от **id** потока, управляющие условия должны быть написаны так, чтобы минимизировать число различных разветвлений.

Иногда компилятор может развернуть циклы или оптимизировать использование **if** или switch операторов посредством ветвления предикатов. Это можно контролировать, используя **#pragma unroll** директиву.

3. Глобальная память

Глобальная память не кэшируется, а это очень важно для получения максимальной производительности. Ниже описаны два способа, помогающих ускорить процесс работы с глобальной памятью:

Первое устройство способно читать 32, 64, 128 битные слова из глобальной памяти в регистры в одну инструкцию. Чтобы иметь такое определение:

```
__device__ type device[32];
type data = device [tid];
```

компилируемое в выполняемое в одну инструкцию. **type** должен быть такой, чтобы **sizeof(type)** был эквивалентен 4, 8 или 16 байтам, и переменные типа type должны быть выравнены на **sizeof(type)** байт.

Для структур требования выравнивания и размера могут быть выполненными компилятором посредством использования директивы **__align__(8)** или **__align__(16)**, например:

```
struct __align__(16) {
float a;
float b;};
```

Для структур больше 16 байт компилятор генерирует различные инструкции загрузки. Для того чтобы убедиться что будет генерироваться минимальное количество инструкций мы можем определить структуру с директивой **__align__(16)**. Тогда компилятор будет использовать выравнивание по максимальному размеру.

Любой адрес переменной, живущей в глобальной памяти или возвращающийся одной операцией выделения памяти из драйвера или во время выполнения, всегда выровнен по наименьшим 256 байтам.

Второе, пропускная способность глобальной памяти используется более эффективно когда одновременный доступ к памяти потоков в половине варпа может быть объединен в одну транзакцию, размер которой может быть также 32, 64 или 128 байт.

7. ОБЪЕДИНЕНИЕ НА УСТРОЙСТВАХ С COMPUTE CAPABILITY 1.0 И 1.1

Доступ к глобальной памяти всеми потоками половины варпа объединяется в одну или две транзакции, если это удовлетворяет следующим трем условиям:

- Потоки должны получить доступ:
  - Любым 32-битным словам, выливающийся в 64-байтовую транзакцию.

  - Или 64-битным словам, выливающийся в 128-байтовую транзакцию.
  - Или 128-битным словам, выливающийся в 128-байтовую транзакцию
- Все 16 слов должны лежать в одном сегменте имеющего размер эквивалентный размеру транзакции.
- Потоки должны получить доступ к словам в последовательности: **k**-й поток обращается к **k**-му слову.

Если эти условия не будут выполняться, тогда транзакции будут осуществляться для каждого потока, что значительно повлияет на производительность системы. Рис. 10 показывает некоторые примеры объединенного доступа к памяти, в то время, как рис. 11 и рис. 12 показывают примеры не объединенного доступа.

8. ОБЪЕДИНЕНИЕ НА УСТРОЙСТВАХ С COMPUTE CAPABILITY 1.2 И ВЫШЕ

Доступ к глобальной памяти всеми потоками половины варпа объединяется в одну или две транзакции как только слова доступные всем потокам лежат в одном сегменте размером:

- 32 байта, если доступ к 8 – битным словам.
- 64 байта, если доступ к 16 битным словам.
- 128 байт, если доступ к 32 или 64 битным словам.

Объединение будет возможно для любых запрошенных образцов адресов, включая образцы, когда много потоков обращаются к одному адресу. Если потоки адресуют слова в n различных сегментах, тогда будет осуществлено n транзакций памяти, по транзакции на каждый сегмент, тогда как устройства с меньшей версией вычислительной способности выпускали бы 16 транзакций, как только n было бы больше 1. В частности, если поток обращается к 128 – битному слову, это вызвало бы менее чем 2 транзакции.

Неиспользуемые слова в транзакции так же читаются, хотя они не нужны и уменьшают производительность. Необходимо добиться как можно меньшего наличия таких слов с целью увеличения полосы пропускания памяти.

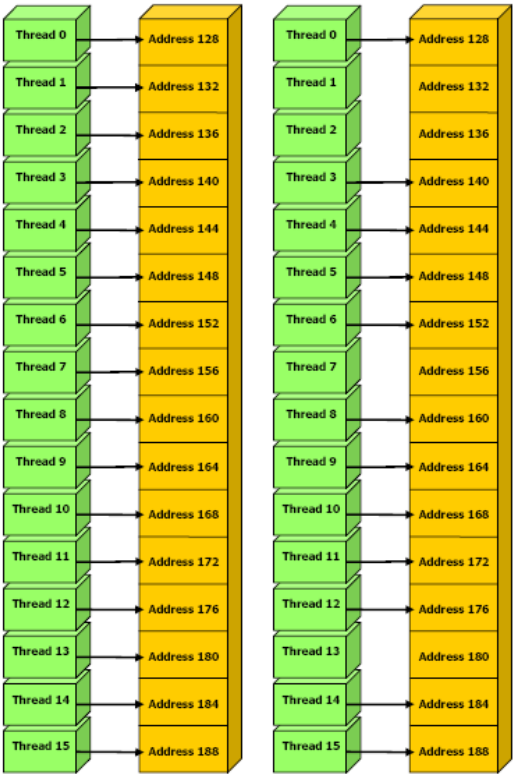


Рисунок 10
Левая колонка: объединенный float доступ к памяти, результат 1 транзакция.
Правая колонка: объединенный float доступ к памяти(расходящиеся потоки), результат 1 транзакция.

Ниже показан алгоритм, который используется для осуществления транзакции памяти:

- Найти сегмент памяти, который содержит в себе адрес, запрошенный наименьшим числом активных потоков. Размер сегмента в 32-байта для 8 битных данных, 64 – байта для 16 – битных данных, 128 байт для 32-, 64- и 128 битных данных.
- Найти все другие активные потоки, чьи адреса лежат в одном сегменте.
- Уменьшить размер транзакции, если возможно.
- Если размер транзакции 128 байт и только нижняя или верхняя половина использовалась, уменьшить размер до 64 байт.

Рисунок 11
Левая колонка: непоследовательный доступ к памяти, результат 16 транзакций.
Правая колонка: доступ с невыровненным стартовым адресом, результат 16 транзакций.

- Если размер транзакции 64 байт и только нижняя или верхняя половина использовалась, уменьшить размер до 32 байт.
- Выполнить транзакцию и пометить обслуживаемые потоки как неактивные.
- Повторить пока все потоки не будут обслужены.

Рис. 13 показывает некоторые примеры доступа к глобальной памяти для устройств с Compute Capability 1.2.

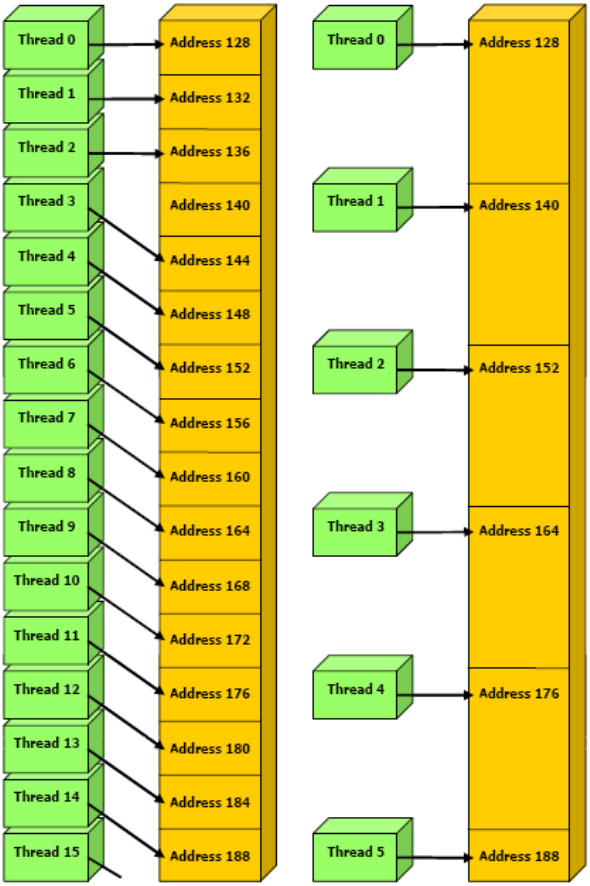


Рисунок 12
Левая колонка: невыровненный доступ, результат 16 транзакций.
Правая колонка: необъединенный доступ, результат 16 транзакций.

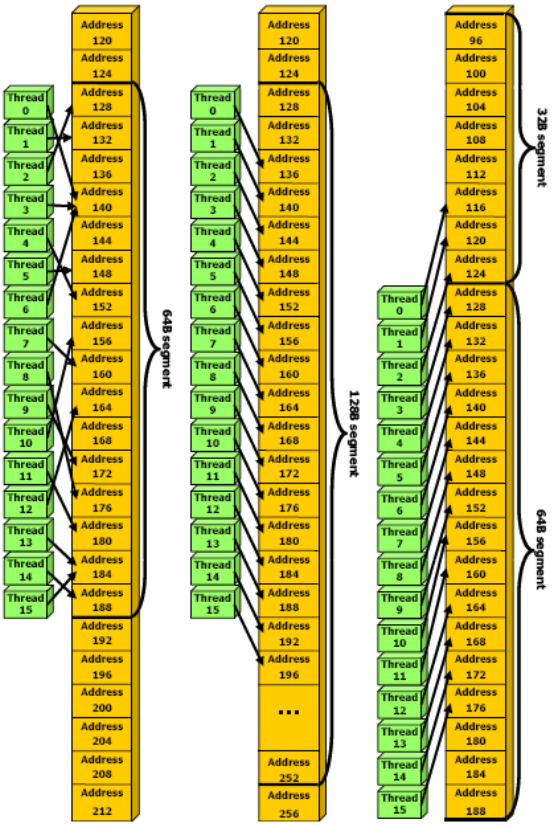


Рисунок 13
Слева: произвольный доступ в пределах 64 байтового сегмента, результат 1 транзакция.
Центр: невыровненный доступ, результат 1 транзакция.
Справа: невыровненный доступ, результат 2 транзакции.

СПИСОК ЛИТЕРАТУРЫ

1. Берилло А. NVIDIA CUDA – неграфические вычисления на графических процессорах // Информационный ресурс сети интернет IXBT.com, 23.09.2008 г. – http://www.ixbt.com/video3/cuda-1.shtml.
2. Чеканов Д. NVIDIA CUDA: вычисления на видеокарте или смерть CPU? // Tom’s Hardware Guide, 22.07.2008 г. – http://www.thg.ru/graphic/nvidia_cuda/index.html.
3. CUDA 2.0 Programming Guide // NVIDIA Corporation, 2007–2008.

А. П. Стасенко

ГЕНЕРАЦИЯ ИСПОЛНЯЕМЫХ ТЕСТОВ ДЛЯ КОМПИЛЯТОРА

ВВЕДЕНИЕ

Данная статья посвящена описанию генератора тестов, пригодного для автоматического тестирования компилятора языка программирования и выявления в нём ошибок, связанных с синтаксисом и семантикой языка программирования [1]. Автоматическая генерация тестов является важной вспомогательной частью тестирования компилятора, так как тесты, разработанные вручную, не могут покрыть все возможные комбинации использования конструкций языка. Для автоматизации тестирования компиляторов были разработаны многочисленные подходы [2].

Генерация синтаксически правильных тестов не представляет затруднений и обычно осуществляется с помощью легко создаваемой контекстно-свободной грамматики. Однако генерация компилируемой (статически семантически правильной) программы намного сложнее, так как в ней требуется обеспечить выполнение семантических условий, таких как использование только определённых переменных и т.п. Такого рода условия возможно учитывать с помощью контекстно-зависимых грамматик, создание которых весьма нетривиально. Ещё более сложной представляется задача генерации конечных и детерминированных программ (динамически семантически правильных). Свойство детерминированности программы важно, так как позволяет утверждать наличие ошибки компилятора, если при его использовании наблюдается изменение поведения программы.

Ввиду сложностей создания контекстно-зависимых грамматик, генераторы конечных и детерминированных программ, как правило, представляют собой некоторую монолитную программу на языке высокого уровня (например, языке Perl). Такой генератор обычно очень сложно модифицировать, и поэтому он рассматривается как черный ящик, пригодный для генерации определенного класса программ некоторого языка. Существует подход к генерации исполняемых программ, удовлетворяющих некоторому набору предварительно заданных шаблонов, задающих вариацию некоторых параметров, однако гибкость такого подхода весьма ограничена и практически равноценна ручной разработке тестов.

Ввиду недостатков указанных подходов, для генерации тестов был выбран подход, основанный на использовании так называемой параметрической контекстно-свободной грамматики [3]. Существуют работы по использованию самомодифицирующихся контекстно-свободных грамматик [4], однако они проигрывают в наглядности и простоте параметрической контекстно-свободной грамматике.

Статья устроена следующим образом. В разд. 1 приводится общее описание параметрической контекстно-свободной грамматики. В разд. 2 находится описание основных моментов, связанных с разработанным генератором и грамматикой. Более формальное описание синтаксиса грамматики располагается в разд. 3. Разд. 4 состоит из примера грамматики и сгенерированной по ней программы. Статья завершается разд. 5, в котором освещена практика применения генератора.

1. ОПИСАНИЕ ПАРАМЕТРИЧЕСКОЙ КОНТЕКСТНО-СВОБОДНОЙ ГРАММАТИКИ

Каждый нетерминал параметрической контекстно-свободной грамматики имеет параметр, являющийся цепочкой, задающей контекст. Левая часть правил, помимо нетерминала, содержит некоторый образец, которому должен удовлетворять контекст нетерминала при определении возможности применить данное правило. Параметрическая контекстно-свободная грамматика по наглядности сопоставима с обычной контекстно-свободной грамматикой, но позволяет генерировать более широкой класс контекстно-зависимых языков. Например, в данной грамматике описывается генерация блочных операторов языка Pascal, в которых объявляются целочисленные и вещественные переменные и осуществляется присвоение между переменными только одного типа:

```
1) <S> ::= <PROG>
2) <PROG> ::= <BLK>IDLIST
3) <BLK>IDLIST = BEGIN <DCL>IDLIST <STM>IDLIST END
4) <DCL>TYPE ID,IDLISTV ::= ID:TYPE; <DCL>IDLISTV
5) <DCL>::=
6) <STM>IDLISTV1 TYPE ID, IDLISTV2 ::=
     ID := <VAR>TYPE*IDLISTV1 TYPE ID, IDLISTV2;
     <STM>IDLISTV1 TYPE ID,IDLISTV2
7) <STM>IDLIST1 ::= <BLK>IDLIST ; <STM>IDLIST1
8) <STM>IDLISTV ::=
```

```
 9) <VAR>TYPE*IDLISTV1 TYPE ID, IDLISTV2 ::= ID
10) <TYPE> ::= INTEGER | REAL
```

Идентификатор (имя переменной) задаётся извне нетерминалом ID. Под нетерминалом IDLISTV (задаваемым специальным образом) подразумевается список уникальных типизированных идентификаторов, включая пустую цепочку[1]. В правилах 2 и 7 непустой список IDLISTV, обозначаемый как IDLIST, находится в правой части правил, не встречается в их левых частях, и поэтому задаёт новый случайный список. В правиле 9 повторение слова TYPE задаёт ту же цепочку, что и первое использование этого слова. Правило 9 отвечает за выбор переменной с типом, равным типу переменной, стоящей в левой части присваивания правила 6.

При генерации допустимые правила выбираются случайным образом, что не даёт гарантий достижимости всех правил. Как показывают последние исследования в этой области, такая ситуация характерна и для других современных подходов к тестированию статической и динамической семантики компиляторов [2]. Например, язык, описываемый приведённой выше грамматикой, содержит следующую цепочку[2]:

```
BEGIN
  T : INTEGER;
  G : REAL;
  A : REAL;
  C : REAL;
  G := A;
  BEGIN
    B : REAL;
    G : INTEGER;
    A : REAL;
  END;
  T := T;
END
```

Для получения цепочки, соответствующей читаемой программе с переносами строк и соответствующими отступами, дополнительно предполагается использовать самоочевидные команды $RIGHT, $LEFT и $LINE:

```
<DCL>TYPE ID, IDLISTV ::= ID:TYPE; $LINE <DCL>IDLISTV
```

[1] Например, цепочка «integer T,real G,real A,» принадлежит языку IDLISTV.

[2] При использовании правил 2 и 7 соответственно создаются списки IDLIST «integer T,real G,real A,real C,» и «real B,integer G,real A,».

2. ОБЩЕЕ ОПИСАНИЕ ГЕНЕРАТОРА И ГРАММАТИКИ

Для создания генератора параметрической контекстно-свободной грамматики требовалось разрешить несколько неясностей оригинальной статьи [3]. Первая неясность связана с алгоритмом определения кратчайших правил грамматики, необходимых алгоритму генерации для возможности гарантированного завершения генерации и, тем не менее, порождения цепочки, принадлежащей языку грамматики. Пояснения к алгоритму были опубликованы позже в другой работе [5].

Вторая неясность связана с распознаванием образца левой части правил. В оригинальной статье для этого предлагается использовать нетерминалы этой же грамматики. Такой подход может оказаться крайне неэффективным, так как даже если используемый для этого нетерминал задаёт контекстно-свободный язык, не определяемый через параметризированные правила, то в общем случае его распознавание возможно за кубическое время от длины распознаваемой цепочки.

В разработанном генераторе распознавание осуществляется путём использования обычных регулярных выражений языка Python. Регулярные выражения допускают распознавание за линейное время и достаточны для распознавания контекста, достаточного для генерации программ реальных языков программирования. Язык Python был выбран в качестве языка реализации генератора[3] по причине того, что в его регулярные выражения включена возможность давать имена распознанным группам символов[4]. Например, здесь распознается и используется в правой части правила имя группы vars:

```
assign[.*@(?P<vars>.+)] = lref[vars] " = " int_expr[*] ";"
```

Распознавание контекста может состоять из нескольких регулярных выражений, причём в последующих выражениях можно использовать имена групп символов, распознанных ранее. Также для выборки случайного элемента списка (полученного разбиением строки некоторым разделителем) существует специальный синтаксис[5]. Например, следующее правило сначала обозначает весь непустой контекст именем vars, разделяет строку vars на

[3] Текст программы генератора занимает около 1000 строк хорошо комментированного кода.

[4] В распространённых версиях языка Perl, распознанные группы символов можно использовать только по их порядковому номеру.

[5] Выбор совпадения случайным образом (из всех возможных совпадений) нереализуем с помощью обычных регулярных выражений.

элементы символом запятой и случайным выбирает элемент списка, удовлетворяющий второму регулярному выражению в квадратных скобках:

```
lref [(?P<vars>.+)] vars=~?",":[\w+ (?P<id>\w+)] = id
```

В генераторе реализовано несколько интересных идей, возможно, использующихся в контексте генерации тестов впервые. Например, от языка Python была позаимствована идея использовать отступы в многострочных правилах грамматики для генерации отступов (и соответствующих переводов строк) в генерируемой цепочке символов. Благодаря этому, описание грамматики языка программирования и порождаемые ею программы становятся более наглядными без дополнительных пометок в грамматике. Например, следующее правило генерирует блочный оператор, ограниченный фигурными скобками, содержимое которых смещено вправо:

```
block [(?P<ftype>\w+)@(?P<old_vars>.*);(?P<new_vars>.*)] =
  "{"
    decls[new_vars]
    stmts[ftype "@" set_join[new_vars ";" old_vars]]
  "}"
```

Генератор поддерживает несколько правил специального вида для облегчения генерации программ языков программирования. Во-первых, существует правило для генерации цепочки символов, состоящей из уникальных частей, разделенных некоторым разделителем. Данное правило полезно для порождения уникальных последовательностей идентификаторов. Например, следующее правило порождает от нуля до трёх уникальных идентификаторов, разделённых символом запятой:

```
ids = { sep=",", min=0, max=3 } id
```

Во-вторых, существуют правила, которые для некоторого нетерминала запрещают генерацию цепочек, удовлетворяющих некоторому регулярному выражению. Данное правило полезно для генерации идентификаторов, не принадлежащих множеству ключевых слов языка программирования. Например, следующая последовательность правил генерирует цепочку букв, которая не может быть равна слову «for»:

```
id  = letter letters
id -= "for"
```

Как в первом, так и во втором случае может происходить повторная генерация нетерминала[6]. Количество попыток ограничено некоторой (достаточно большой) константой. При исчерпании доступного количества попыток грамматика считается неправильной и генерация выходной цепочки аварийно прерывается. Если для текущего нетерминала и его контекста не существует правила, им удовлетворяющего, то также происходит аварийное прекращение генерации цепочки, выводится стек нетерминалов и их контекстов, приведших к аварийной ситуации.

В грамматике есть средства для дополнительного управления процессом генерации выходной цепочки. Во-первых, для каждого правила существует возможность указать его вес, который прямо пропорционально влияет на вероятность срабатывания данного правила среди всех применимых правил[7]. Например, данное правило будет порождать в три раза (18 / 6 = 3) больше операторов присваивания (assign), чем условных операторов (if):

```
stmt = *18 assign[*] | *6 if[*]
```

Во-вторых, для каждого правила можно вручную указать кратчайшую альтернативу[8], когда алгоритм автоматического её определения не может сделать этого ввиду потенциальной бесконечности всех альтернатив. Если среди всех применимых альтернатив кратчайшая альтернатива неизвестна, то происходит аварийное прекращение генерации цепочки и выводится стек нетерминалов и их контекстов, приведших к аварийной ситуации. Например, в данном правиле указывается («!1»), что первая альтернатива («int_expr2[*]») обычно самая короткая (и генерируется в два раза чаще, чем вторая альтернатива):

```
int_expr = !1 *2 int_expr2[*]
         | int_expr[*] " " op2 " " int_expr[*]
```

3. ФОРМАЛЬНОЕ ОПИСАНИЕ ГРАММАТИКИ

Грамматика состоит из строк, каждая из которых может заканчиваться строковым комментарием, начинающимся с символа «#» (для упрощения

[6] Повторная генерация может быть вызвана нарушением свойства уникальности цепочки или генерацией запрещенной последовательности символов.

[7] Правильная балансировка правил важна для предупреждения излишней генерации одного и того же правила.

[8] Кратчайшая альтернатива используется при необходимости корректно завершить генерацию выходной цепочки (обычно при достижении ею определенного размера).

далее описаны только комментарии, занимающие целую строку и не являющиеся частью правил):

```
<грамматика> ::= <строки>
<строки> ::= <строка> [«\n»[9] <строки>][10]
<строка> ::= <правило> |[11] <комментарий> | <пустая строка>
  <комментарий> ::= # <произвольные символы>
```

Правило может быть как генерирующим (левая часть правила отделяется от правой части знаком «=»), так и ограничивающим (левая часть правила отделяется от правой части знаком «-=»):

```
<правило> ::= <левая часть правила> = <правая часть правила> |
  <левая часть правила> -= "<регулярное выражение Python>"[12]
```

Имя параметра шаблона контекста в левой части правила может быть любым именем распознанной группы символов регулярного выражения языка Python, стоящей в предыдущем шаблоне контекста левой части правила. Если имя параметра шаблона оканчивается символами «!~», то шаблон становится негативным (в случае наличия нетерминала разделителя ни один из элементов полученного списка не должен удовлетворять шаблону)[13].

```
<левая часть правила> ::= <имя нетерминала> [<шаблоны контекстов>]
<имя нетерминала> ::= <идентификатор>
<шаблоны контекстов> ::= <шаблон контекста> [<шаблоны контекстов>]
<шаблон контекста> ::= [<параметр шаблона>] [<разделитель>] <шаблон>
<параметр шаблона> ::= <имя параметра> =~ | <имя параметра> !~
<имя параметра> ::= <идентификатор>
<разделитель> ::= ?"<строка>"
<шаблон> ::= «[» <регулярное выражение Python> «]»
```

Правая часть правила может задаваться либо альтернативами, разделёнными вертикальной чертой, которые могут располагаться на нескольких строках, либо многострочным правилом, которое начинается с перевода строки (сразу после знака «=») и заканчивается пустой строкой. Отступы и переводы строк в многострочном правиле используются при его генерации.

---

[9] Здесь и далее в угловых двойных кавычках находятся строки нетерминалов, которые могут быть неоднозначно истолкованы. Под нетерминалом «\n» подразумевается перевод строки.

[10] В квадратных скобках находятся необязательные части грамматики.

[11] Прямая линия обозначает несколько возможных альтернатив правила.

[12] Прямая двойная кавычка обозначает нетерминал двойной кавычки.

[13] В негативном шаблоне нет смысла именовать распознанные группы.

```
<правая часть правила> ::= <альтернативы> | <многострочное правило>
<альтернативы> ::= <альтернатива> [ «|» [«\n»] <альтернативы>]
<альтернатива> ::= <модификаторы> <элементы> | <модификаторы> 0[14]
<многострочное правило> ::= «\n» <модификаторы>
                            <строки правила> <пустая строка>
<строки правила> ::= <строка правила> [«\n» <строка правила>]
<строка правила> ::= <элементы>
```

Все модификаторы необязательны. По умолчанию приоритет определяется автоматически, вес полагается равным единице, а в генераторе множества разделительная строка полагается равной пробелу, минимальное количество – единице, а максимальное количество – максимуму из значений min+19 и min*2.

```
<модификаторы> ::= [<приоритет>] [<вес>] [<генератор мн-ва>]
<приоритет> ::= ! <число>
<вес> ::= * <число>
<генератор мн-ва> ::= «{» [<параметры мн-ва>] «}»
<параметры мн-ва> ::= sep="<строка>"[, min=<число>][, max=<число>] |
                      min=<число>[, max=<число>] | max=<число>
```

Звездочка в качестве элемента правой части правила обозначает весь контекст нетерминала в левой части правила.

```
<элементы> ::= <элемент> [« » <элементы>]
<элемент> ::= "<строка терминалов>" | «*» | <имя параметра>[15] |
              <имя нетерминала> [«[» <элементы> «]»]
```

4. ПРИМЕР ГРАММАТИКИ И СГЕНЕРИРОВАННОЙ ПРОГРАММЫ

Далее приводится пример полной грамматики, отдельные правила которой уже рассматривались выше. Грамматика генерирует только правильные (компилируемые) программы на языке Си, имеющие детерминированное исполнение и завершающиеся за конечное время. Программы содержат скаляры и массивы целого типа, произвольные выражения целого типа от скалярных переменных и элементов массивов, операции присваивания, условные и блочные операторы.

[14] Для повышения наглядности и уменьшения ошибок пустое множество явным образом обозначается символом нуля.

[15] Имена распознанных групп (имена параметров) перекрывают имена нетерминалов.

```
#$ GEN_LENGTH = 512[16]

S = program[vars]
program = decls[*]
          "int main()"
          block["int" "@" * ";" vars]

### объявления и определения

decls[] = 0
decls[(?P<type>\w+) (?P<id>\w+)(?:,(?P<tail>.*))?] =
        type " " id " = " num ";"
        decls[tail]

decls[(?P<type>\w+) (?P<id>\w+)\[\],?(?P<tail>.*)] =
        type " " id "[4] = {" num ", " num ", " num ", " num "};"
        decls[tail]

### утверждения

stmts  = *3 stmts2[*] | stmts2[*] return[*]
stmt   = *18 assign[*] | *6 if[*]
stmts2 = 0
stmts2 = *3 stmt[*]
         stmts2[*]

assign[.*@(?P<vars>.+)] = lref[vars] " = " int_expr[*] ";"
assign[.*@] = ";"

if = if_part[*] | if_part[*] else_part[*]
if_part =
    "if ( " rel_expr[*] " )"
    block[* ";" vars]

else_part =
    " else"
    block[* ";" vars]

return [(?P<ftype>\w+)@.*] = "return " int_expr ";"

block [(?P<ftype>\w+)@(?P<old_vars>.*);(?P<new_vars>.*)] =
  "{"
    decls[new_vars]
    stmts[ftype "@" set_join[new_vars ";" old_vars]]
  "}"

### выражения

int_expr  = !1 *2 int_expr2[*] | int_expr[*] " " op2 " " int_expr[*]
```

[16] Генератор поддерживает некоторые прагмы. Например, прагма GEN_LENGTH задаёт количество символов в генерируемой последовательности, после которого генератор переходит в режим выбора кратчайших правил.

```
int_expr2 = !1 *10 term[*] | *2 "(" int_expr[*] ")" | "-" term[*]
rel_expr  = !1 *2 rel_expr2[*] |
              rel_expr[*] " " log_op2 " " rel_expr[*]
rel_expr2 = !1 *24 term[*] |
              *8  int_expr[*] " " rel_op2 " " int_expr[*] |
              *3  "!" term[*] | *1 "(" rel_expr[*] ")"
term                = num
term[.*@(?P<vars>.+)] = *2 lref[vars]
lref [(?P<vars>.+)] vars=~?",":[\w+ (?P<id>\w+)] = id
lref [(?P<vars>.+)] vars=~?",":[\w+ (?P<id>\w+)\[\]]
  = id "[" digit "]" | id "[(" int_expr[*] ")%4]"
op2     = "+" | "-"
rel_op2 = "<" | ">" | "=="
log_op2 = "||" | "&&"

### операции над множествами

ids        = { sep=",", min=0, max=3 } id
vars       = vars_gen[ids]
vars_gen[] = 0
vars_gen[(?P<id>\w+)(?P<sep>,?)(?P<tail>.*)]
  = type " " id sep vars_gen[tail] |
    type " " id "[]" sep vars_gen[tail]

# set_join бёрёт на вход два мн-ва vars A и B,
# разделённых «;» и возвращает мн-во vars = A U (B\A),
# причём B\A задаётся с помощью set_sub

set_join [(?P<s1>[^;]+);[^;]*] = s1 "," set_sub[*]
set_join [;(?P<s2>[^;]*)] = s2
set_sub [(?P<s1>.+);] = 0
set_sub [(?P<s1>.+);(?P<t>\w+)
(?P<id>\w+)(?P<arr>\[\])?(?P<sep>,?)(?P<s2>.*)]
        s1!~[(?:.+,)?\w+ (?P=id)(?:\[\])?(?:,.+)?] =
  t " " id arr sep set_sub[s1 ";" s2]

set_sub [(?P<s1>.+);\w+ (?P<id>\w+)(?:\[\])?,?(?P<s2>.*)]
        s1=~[(?:.+,)?\w+ (?P=id)(?:\[\])?(?:,.+)?] =
  set_sub[s1 ";" s2]

### идентификаторы, числа и типы (очень упрощенный вариант)

id      = letter letters
id     -= "for"
letter  = "f" | "o" | "r"
letters = letter letters | *10 0
num    = digit digits
digit  = "0" | "1" | "2" | "3"
digits = digit digits | *10 0
type = "int"
```

Приведённая выше грамматика порождает, например, следующую цепочку символов, являющуюся правильной программой на языке Си:

```
int f[4] = {3, 1, 0, 2};
int o = 0;

int main()
{

  if ( o < 2 + (-3) )
  {
    int o[4] = {2, 2, 0, 0};
    int of = 0;

    of = o[(01 - 33 - (3))%4];
    if ( 2 )
    {
      int rf = 0;

      f[(1)%4] = o[3];
      return 3;
    }
    of = (of);

  } else
  {
    int f = 30;


  }
  if ( f[(3)%4] )
  {

    return -2 + 1 + 0 - 0;
  } else
  {
    int o = 3;

    f[0] = 2 + o;
    return 2;
  }
  o = 1;
  f[3] = o;
  return 0 - (1) + (3);
}
```

5. ПРАКТИКА ПРИМЕНЕНИЯ ГЕНЕРАТОРА

В качестве доказательства полезности предлагаемого подхода к генерации тестов была написана простейшая грамматика для порождения детер-

минированных, исполняемых программ на языке Си-99. Грамматика нацелена на выявление ошибок во внутренних преобразованиях компилятора, и поэтому не содержит замысловатых синтаксических конструкций.

Грамматика генерирует программы с одной функцией (main), переменными и одномерными массивами разнообразных целочисленных типов. Функция состоит из случайной последовательности утверждений. Утверждением может быть присваивание переменной произвольному выражению, условный оператор (if-else) с произвольными логическими операциями, цикл for или while. Условные операторы и циклы содержат блочный оператор, также состоящий из случайной последовательности утверждений. Утверждением также может быть оператор вывода значения производной переменной или элемента массива и вызов специальной функции из внешнего модуля (недоступного компилятору), которая изменяет значение, переданное по его указателю (для проверки возможности этого изменения компилятором).

Индексные выражения для выбора элементов массива никогда не выходят за границы массива (для обеспечения конечности и детерминированности генерируемых программ) благодаря статическому или динамическому контролю. Статический контроль обеспечивается индексами, являющимися циклическими переменными с некоторым возможным фиксированным смещением. В свою очередь, количество итераций циклов контролируется статически или динамически и является инвариантом цикла. Грамматика также запрещает генерацию программ с произвольной модификацией циклических переменных (также во избежание появления бесконечных циклов).

Указанная грамматика занимает всего около 200 строк и описывает генерацию указанного класса программ вплоть до генерации отдельных символов. Генератор и указанная грамматика были опробована на бета версии 11.1 компилятора фирмы Intel® для операционной системы Linux. Программа компилировалась с оптимизациями и без них и, если вывод программы отличался, то производилась автоматическая минимизация программы (на текущий момент простейший алгоритм удаления строк) до программы, всё ещё дающей отличие. Минимизированная программа использовалась для дальнейшего анализа проблемы. За месяц работы генератора среди около полумиллиона тестов было найдено около двух десятков новых (разных) проблем в компиляторе, которые были признаны разработчиками компилятора и будут исправлены ими. Большинство найденных проблем вызвано ошибками в оптимизирующих преобразованиях, тогда как остальные проблемы вызваны аварийной остановкой компиляции.

ЗАКЛЮЧЕНИЕ

Разработан генератор, основанный на формализме параметрической контекстно-свободной грамматики и допускающий генерацию контекстно-зависимых языков, достаточных для автоматического построения семантически правильных (компилируемых) программ, имеющих детерминированное исполнение. Путём анализа различий вывода полученных детерминированных программ, скомпилированных с оптимизациями и без них, были обнаружены ошибки в бета версии компилятора промышленного уровня.

В дальнейшем планируется расширение грамматики для покрытия более широкого класса программ, включая возможности языка Си++, такие как обработка исключений. Есть идея усовершенствовать синтаксис грамматики путём добавления списков контекстов и операций для работы с ними вместо одной монолитной строки контекста. Логичным выглядит добавление возможности минимизации программы, диагностирующей ошибку компилятора, на основании дерева генерации[17].

СПИСОК ЛИТЕРАТУРЫ

1. Aho A.V., Sethi R., Ullman J.D. Compilers: principles, techniques, and tools. – Boston: Addison-Wesley Longman Publishing Co., Inc., 1986. – 796 p.
2. Kossatchev A. S., Posypkin M. A. Survey of compiler testing methods // Programming and Computing Software. – NY: Plenum Press, 2005. – Vol. 31, No. 1. – P. 10–19.
3. Bazzichi F, Spadafora I. An automatic generator for compiler testing // IEEE transactions on Software Engineering. – NY: IEEE, 1982. – Vol. SE-8. – P. 343–353.
4. Hanford K.V. Automatic generation of test cases // IBM Systems Journal – NY: IBM, Dec. 1970. – Vol. 9. – P. 242–257, Dec. 1970.
5. Malloy B.A., Power J. F. An Interpretation of Purdom's Algorithm for Automatic Generation of Test Cases // Proceedings of the First Annual International Conference on Computer and Information Science. – Orlando, Florida, USA, October 3-5, 2001. – P. 310–317.

[17] В корне дерева генерации находится начальный символ грамматики с пустым контекстом. В вершинах дерева генерации находятся нетерминалы и их контексты. Минимизация проводится путём попыток повторной генерации деревьев с корнем в некотором нетерминале в соответствии с его кратчайшим правилом.

Ы. Турсунбай кызы

ДЕРЕВЬЯ КЛИК ХОРДАЛЬНОГО ГРАФА И ДЕРЕВЬЯ ПОДГРАФОВ В ТЕОРИИ ГРАФОВ*

1. ВВЕДЕНИЕ

Некоторые классы графов могут рассматриваться с помощью общего вида структуры дерева, определенного относительно отобранных индуцированных подграфов.

Классический подход деревьев клик к хордальным графам, а также к строго хордальным графам может быть обобщен, для того чтобы показать общую структуру для других классов графов, таких как строго хордальные графы, полностью сбалансированные гиперграфы, хордально двудольные графы и др. Обобщение деревьев клик посредством выбора других видов индуцированных подграфов, таких как окрестности вершин, позволяет определенные понятия и результаты хордальных графов переносить на другие классы графов. Таким образом, можно сказать, что представление графа деревом индуцированных подграфов различного рода имеет ценное практическое применение в общей теории графов.

Деревья клик имеют много применений, например, они могут использоваться для изучения взаимодействия белка в биологии [3]. Большинство клеточных процессов выполняется комплексами мультибелков – группами белков, объединенных для выполнения определенной задачи. Лучшее понимание данной организации белков в перекрывающихся комплексах – важный шаг к раскрытию функциональных и эволюционных механизмов биологических сетей. Эта ситуация может быть представлена графом, где вершины – белки, и две вершины связаны ребром, если соответствующие белки взаимодействуют. Комплекс белков может быть рассмотрен как клика данного графа. Тогда, когда граф является хордальным, дерево клики и семейство поддеревьев, представляющих вершины графа, обеспечивают хорошую основу для последующей деятельности белка в различных комплексах.

*Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (грант № 09-01-90901)

2. ДЕРЕВЬЯ КЛИК ХОРДАЛЬНОГО ГРАФА

Пусть $G = (V, E)$ – некоторый граф и $S \subseteq V$. Подграф графа G, *порожденный* множеством S, – это $G[S] = (S, E[S])$, где $E[S] = \{u, v \in S \mid u, v \in S\}$. *Кликой* K графа называется подмножество графа G, в котором любые две вершины смежны, т.е. порожденный ими подграф является полным. Клика называется *максимальной*, если она не содержится в никакой другой клике. Обозначим множество максимальных клик графа G через K_G.

Предположим, что T – дерево, вершинами которого являются максимальные клики. Вершины дерева будем называть узлами, во избежание путаницы между ними и вершинами графа. Пусть для каждой вершины $v \in V(G)$ T_v означает подграф T, порожденный теми узлами, которые содержат v. Если каждый такой подграф T_v является связным, другими словами, если каждый T_v является поддеревом дерева T, тогда T называется *деревом клик* графа G.

Дерево клик T графа G обладает ***свойством индуцированных поддеревьев***: для всех $v \in V(G)$ подграф T, порожденный множеством максимальных клик графа G, содержащих вершину v, является деревом. Это эквивалентно ***свойству пересечения клик***: если любые две клики $K, K' \in K_G$, то множество $K \cap K'$ содержится в каждой клике на пути T между K и K'.

Граф называется *хордальным*, если каждый его цикл длины >3 содержит хорду, т.е. ребро, соединяющее несмежные вершины простого цикла.

Хордальные графы составляют один из наиболее изученных и применяемых классов графов. Так как хордальный граф является совершенным, его кликовое число равняется его хроматическому числу. Поэтому кликовое дерево графа непосредственно обеспечивает нахождение клик и хроматического числа графа. Более подробные определения и свойства хордальных графов приведены в [1], [2].

Авторы работ [4], [5] и [6] независимо друг от друга обнаружили одинаковые характеристики для хордальных графов.

Лемма 1. *Граф G является хордальным тогда и только тогда, когда он имеет дерево клик.*

При одном из способов доказательства того, что G имеет дерево клик T, только когда граф является хордальным, используется тот факт, что G является хордальным тогда и только тогда, когда G имеет совершен-

ное элиминирующее упорядочение, означающее упорядочение $v_1, ..., v_n$ вершин $V(G)$ так, что каждая v_i находится в уникальной максимальной клике в подграфе G, индуцированном вершинами $v_1, ..., v_n$. Совершенное элиминирующее упорядочение легко получить из дерева клик: v_1 может быть любая вершина G, которая находится в уникальном узле T, затем удалить v_1 из узла T, соединить ребром узлы, которые становятся сравнимыми, повторить. Совершенное элиминирующее упорядочение может также использоваться в рекурсивном построении деревьев клик.

Значительная работа была сделана на возможных "формах", которые принимает дерево клик. Например, авторы работы [7] определяют листву хордального графа как минимальное число листьев дерева клик. Интервальным графом является хордальный граф, имеющий представление в виде поддеревьев, в котором главное дерево является путем; это позволяет рассматривать поддеревья как интервалы на вещественной прямой. Интервальные графы – это хордальные графы с листвой, не больше чем 2.

Ниже приведенные теоремы доказывают, как могут быть распознаны и построены деревья клик. Теорема 1 доказывает, как избежать проверку связности каждого T_v для того, чтобы показать, что дерево T является деревом клик.

Теорема 1. *(Acharya and Las Vergnas [8] and McKee [9]) Предположим, что T – некоторое дерево, узлами которого являются максимальные клики графа G. Тогда T является деревом клик тогда и только тогда, когда выполняется следующее равенство:*

$$\sum_{S \in V(T)} |S| - \sum_{SS' \in E(T)} |S \cap S'| = |V(G)|. \qquad (1)$$

Пусть Ω^W является полным графом с вершинами, которые являются максимальными кликами графа G и с ребрами SS', которые имеют вес $S \cap S' \geq 0$. Теорема 2 доказывает, как легко построить дерево клик, когда он существует.

Теорема 2. *(Bernstein and Goodman [10]) Если G имеет дерево клик, тогда его дерево клик является максимальным каркасом Ω^W.*

Отсюда следует, что для графа G существует дерево клик тогда и только тогда, когда алгоритм Краскала или любой другой жадный алгоритм нахождения максимального каркаса дает нам дерево T для Ω^W,

которое удовлетворяет равенству (1). Соответственно, деревья клик не должны быть однозначно определены. Но следующая теорема доказывает, что для любого дерева клик T графа G, мультимножество $\{S \cap S' : SS' \in E(T)\}$, которое также может быть обозначен как $E(T)$, определяется уникально.

Теорема 3. *(Barrett et al. [11] and Ho and Lee [12]) Любые два дерева клик T графа G имеют одинаковые мультимножества $E(T)$.*

Раскроем значение уникальности мультимножества $E(T)$ теоремы 3 следующим образом: если T является деревом клик графа G, тогда подмножество $Q \subseteq V(G)$ содержится в $E(T)$ только в том случае, когда Q является минимальным вершинным сепаратором графа G. Это значит, что существуют $u, v \in V(G)$ такие, что каждый путь, связывающий вершины u и v, содержит вершину из подмножества Q, и никакое правильное подмножество Q не имеет данного свойства; более того, кратность Q в $E(T)$ равна на единицу меньше количества компонент в подграфе G, порожденным теми вершинами, которые смежны с каждой вершиной в Q.

3. ДЕРЕВЬЯ ПОДГРАФОВ

Пусть L является мультимножеством отобранных индуцированных подграфов графа G, таким, что каждая вершина графа G находится, по крайней мере, в одном элементе L. Пусть T является деревом с множеством узлов L и для всех $v \in V(G)$, T_v означает подграф дерева T, индуцированный узлами, содержащими вершину v. Если каждое такое T_v является связным, тогда T называется *L-деревом* графа G. *Деревом подграфа* G является L-дерево для некоторого мультимножества L подграфов G, например, дерево клик графа G является L-деревом, где L – множество всех максимальных клик G.

Теоремы 1–3 (в которых L – мультимножество всех максимальных клик) могут быть обобщены для деревьев подграфов с множеством вершин L.

Рассмотрим некоторые классы графов, для которых существует дерево подграфов натурального мультимножества L индуцированных подграфов G. В лемме 1 класс графов, имеющий L-деревья, охарактеризован как граф пересечений поддеревьев деревьев. Доказательство того, что граф имеет L-дерево тогда и только тогда, когда граф находится в

определенном классе, зависит от специфических свойств мультимножества L и класса графов.

3.1. Деревья

Граф G является *деревом*, если он связный и не имеет циклов.

Пример 1. Очевидно, что деревья имеют L-дерево, где мультимножество L – это множество всех ребер $E(G)$.

Для деревьев равенство (1) примет вид:

$$\sum_{SS' \in E(T)} |S \cap S'| = |E(G)|.$$

3.2. Графы клик

Граф клик некоторого графа G – это граф пересечений всех максимальных клик графа G. Открытая окрестность $N(v)$ вершины v состоит из всех вершин, смежных с v, а закрытая окрестность $N[v]$ – из $N[v] \cup \{v\}$.

Пример 2. Если L является мультимножеством всех закрытых окрестностей графа G, тогда граф G имеет L – дерево тогда и только тогда, когда G является графом клик хордального графа.

Отметим, что равенство (1) для графа клик уменьшится:

$$\sum_{SS' \in E(T)} |S \cap S'| = 2|E(G)|.$$

3.3. Биациклический гиперграф

Графом инцидентности гиперграфа (X, ε) называется двудольный граф с множеством вершин $(X \cup \varepsilon)$, у которого две вершины $x \in X$ и $S_i \in \varepsilon$ смежны тогда и только тогда, когда $x \in S_i$. Гиперграф (X, ε) является *деревом гиперграфа* (или *гипердеревом*) тогда и только тогда, когда найдется дерево T с множеством вершин X такое, что каждое ребро $S_i \in \varepsilon$ порождает поддерево в T и *двойственное гипердерево* тогда и только тогда, когда найдется дерево T с множеством вершин ε такое, что для всех вершин $x \in X$ множество $T_x = S_i \in \varepsilon : x \in S_i$ порождает поддерево в T. Отметим, что G имеет L-дерево тогда и только тогда, когда $(V(G), L)$ – двойственное гипердерево.

Гиперграф является *биациклическим гиперграфом*, если он и его двойственный гиперграф являются гипердеревом [13].

Пример 3. Если L является мультимножеством всех открытых окрестностей вершины графа G, тогда G имеет L-дерево тогда и только тогда, когда G является графом инцидентности биациклического гиперграфа.

Равенство (1) для графа инцидентности биациклического гиперграфа будет выглядеть следующим образом:

$$\sum_{SS' \in E(T)} |S \cap S'| = 2|E(G) - V(G)|.$$

Также, когда граф G является графом инцидентности биациклического гиперграфа, множество $S \subseteq V(G)$ содержится в $E(T)$ тогда и только тогда, когда S является минимальным вершинным сепаратором, который состоит из попарно несмежных вершин (вершинами из одноцветного класса двудольного графа G).

4. СТРОГИЕ ДЕРЕВЬЯ ПОДГРАФОВ

4.1. Строго хордальные графы

Пусть T является деревом графа G. Мультимножество

$$S \cap S' : SS' \in E(T),$$

также обозначаемое через $E(T)$, состоит из минимальных вершинных сепараторов графа G с кратностью (как описано в теореме 3). Дан граф G с деревом клик T, назовем каркас $T^{(1)}$ полного графа $\Omega^W(E(T))$ *$E(T)$-деревом*, если $T_v^{(1)}$ является связным для всех вершин v, которые являются минимальными вершинными сепараторами графа G.

Строгим L-деревом графа G является L-дерево T такое, что существует также $E(T)$ дерево $T^{(1)}$, $E(T^{(1)})$ дерево $T^{(2)}$ и так далее до тех пор, пока в $T^{(j)}$ не останется ребер.

Хорда x_i, x_j в цикле $C = (x_1, x_2, :, x_{2n})$ четной длины $2n$ является нечетной хордой, если расстояние между x_i и x_j в C является нечетным. G является *строго хордальным графом*, если G является хордальным и каждый цикл четной длины в G не меньше 6 имеет нечетную хорду [14].

Предположим, что $\Omega^W(L)$ означает полный граф с множеством вершин L и с ребром SS', имеющим вес $|S \cap S'| \geq 0$.

Пример 4. Если L является мультимножеством всех максимальных клик графа G, тогда G имеет строгое L-дерево тогда и только тогда, когда G является строго хордальным графом.

Пример 5. Если L является мультимножеством всех вершин закрытой окрестности графа G, тогда G имеет строгое L-дерево тогда и только тогда, когда G является строго хордальным графом.

Из примеров 2 и 5 можно получить следующее.

Лемма 2. *([15])G является строго хордальным графом тогда и только тогда, когда каждый индуцированный подграф графа G является графом клик хордального графа.*

4.2. Полностью сбалансированный гиперграф

Гиперграф – это пара (V, ε), где V – непустое множество объектов некоторой природы, называемых вершинами гиперграфа, а ε – семейство непустых подмножеств множества V, называемых ребрами гиперграфа.

Гиперграф называется *полностью сбалансированным* тогда и только тогда, когда каждый цикл длины больше 2 имеет ребро, содержащее, по крайней мере, три вершины цикла.

Пример 6. Граф G имеет строгое L-дерево тогда и только тогда, когда $(V(G), L)$ является тотально сбалансированным гиперграфом.

Отметим, что проверка, является ли каждая T^j $E(T^{j-1})$-деревом, может быть осуществлен проверкой следующего равенства для всех j:

$$\sum_{S \in V(T^{(j)})} |S| - \sum_{SS' \in E(T^{(j)})} |S \cap S'| = |\bigcup \{S : S \in V(T^{(j)})\}|.$$

4.3. Хордально двудольные графы

G является *хордально двудольным*, если G является двудольным и каждый цикл графа длины не меньше 6 имеет хорду [16].

Отметим, что, например, цикл C_4 является хордально двудольным – такие графы в общем случае нехордальны.

Пример 7. Если L является мультимножеством всех вершин открытой окрестности графа G, тогда G имеет строгое L-дерево тогда и только тогда, когда G является хордально двудольным графом.

Из примеров 3 и 7 можно получить следующую лемму.

Лемма 3. *Граф G является хордально двудольным тогда и только тогда, когда каждый индуцированный подграф G является графом инцидентности биациклического графа.*

СПИСОК ЛИТЕРАТУРЫ

1. **Евстигнеев В.А.** Хордальные графы и их свойства // Проблемы систем информатики и программирования. – Новосибирск, 1999 – С.33–64.
2. **Л**екции по теории графов / В.А.Емеличев, О.И.Мельников, В.И.Сарванов, Р.И.Тышкевич. – М.: Наука, 1990.
3. **Zotenko E., Guimaraes K.S., Jothi R. and Przytychka T.M.** Decomposition of overlapping protein complexes: a graph theoretical method for analyzing static and dynamic protein associations. Algorithms for Molecular Biology, 2006. – Vol.1(7).
4. **Buneman P.** A characterization of rigid circuit graphs. Discrete Math., 1974. – Vol.9. – P.205–212.
5. **Gavril F.** The intersection graphs of subtrees in trees are exactly the chordal graphs. J.Combinatorial Theory (Series B), – 1974. – Vol.16. – P.47–56.
6. **Walter J.R.** Representations of chordal graphs as subtrees of a tree. – J.Graph Theory. 1978. – Vol.2(3). – P.265–267.
7. **Lin L.-J., McKee T.A., West D.B.** Leafage of chordal graph. – Discuss.Math.Graph Theory, 1998. – Vol.18. – P.23–48.
8. **Acharya B.D., Las Vergnas M.**, Hypergraphs with cyclomatic number zero, triangulated graphs, and an inequality. – Journal of Combinatorial Theory (Series B), 1982. – Vol.33. – P.52–56.
9. **McKee T.A.** How chordal graphs work. – Bull. Inst. Combin. Appl., 1993. – Vol.9. – P.27–39.
10. **Bernstein P.A., Goodman N.** Power of natural semijoints. – SIAM. J. Comput., 1981. – Vol.10. – P.751-771.
11. **Barrett W.W., Johnson C.R., Lundquist M.** Determinantal formulae for matrix completions associated with chordal graphs. – Linear Algebra Appl., 1989. – Vol.121. – P.265–289.
12. **Ho C.-W., Lee R.C.T.** Counting clique trees and computing perfect elimination schemes in parallel. – Inform. Process. Lett., 1989. – Vol.31. – P.61–68.
13. **Dragan F.F., Voloshin V.I.** Incidence graphs of biacyclic hypergraphs. – Discrete Applied Mathematics, 1996. – Vol.68. – P.259–266.
14. **Farber M.** Characterization of strongly chordal graphs. – Discrete Math., 1983. – Vol.43. – P.173–189.
15. **Branstadt A., Dragan F.F., Chepoi V.D., Voloshin V.I.** Dually chordal graphs. – SIAM J. Discrete Mathematics, 1998. – Vol.11. – P.437–455.
16. **Golumbic M.C., Goss C.F.** Perfect elimination and chordal bipartite graphs. J.Graph Theory, 1978. – Vol.2. – P.155–163.

СОДЕРЖАНИЕ

CONTENTS

УДК 519.68 + 681.3.06

Индексный анализ зависимостей по данным в Sisal-программах / Арапбаев Р.Н., Стасенко А. П. // Конструирование и оптимизация параллельных программ. – Новосибирск, 2008. – С. 8–20.

Работа посвящена компиляторной технологии, называемой индексным анализом, которая позволяет предоставить информацию о существовании зависимости между операциями, обращающимися к памяти и находящимися в цикле. Такая информация используется многими оптимизациями, особенно циклическими. В данной работе для анализа зависимости по данным в SISAL-программах применяется алгоритм индексного анализа на основе новой стратегии тестирования. Для формирования этой системы требуется выполнить подготовительные действия по поиску гнезд циклов и индуктивных переменных и непосредственному построению уравнений зависимости и систем неравенств. – Библиогр.: 12 назв.

Data dependency index analysis in Sisal programs / Arapbaev R.N., Stasenko A.P. // Parallel programs construction and optimization. – Novosibirsk, 2008. – P. 8–20.

The paper describes a compiler technology called index analysis which allows the presentation of information about the existence of cycled intra-operational depenencies referring to memory. Such information is used in many optimizations, especially the cyclic ones. The formation of this system takes several preparatory steps of searching for cycle nests and inductive variables, and the direct construction of dependency equations and inequality systems. – Refs: 12 titles.

УДК 519.68 + 681.3.06

Расчет интегралов поперечных мер Минковского, сумм Минковского и построение диаграммы Бляшке для выпуклых многогранников в евклидовом пространстве R^3 / Бутовский М. М. // Конструирование и оптимизация параллельных программ. – Новосибирск, 2008. – С. 21–32.

В статье даётся описание основных характеристик выпуклых многогранников. Основным содержанием статьи является задача построения диаграммы Бляшке. Достижение этой цели подразумевает выполнение множества промежуточных задач. При оптимальном подборе алгоритмов суммарное время расчетов сводится к минимуму. Также в статье приводится описание программного продукта, реализующего все требуемые вычисления. – Библиогр.: 10 назв.

Calculation of Minkowsky integrals of cross measures, Minkowsky sums and building of Blaschke diagram for convex polyhedrons in euclidean space R^3 / Butovsky M.M. // Parallel programs construction and optimization. – Novosibirsk, 2008. – P. 21–32.

The article presents a description of key features of convex polyhedrons. The main content of the article is the building of Blaschke diagram. Achieving this includes solving numerous intermediate tasks. The total time of computation reduces to a minimum because of the optimal selection of algorithms. The article also contains a description of a software product created for performing all required computations. – Refs: 10 titles.

УДК 519.68 + 681.3.06

Визуализация внутреннего представления программ в системе функционального программирования SFP / Гордеев Д. С. // Конструирование и оптимизация параллельных программ. – Новосибирск, 2008. – С. 33–47.

В статье представлен алгоритм визуализации внутреннего представления программ. Внутреннее представление может быть представлено в виде иерархического ациклического графа, каждая вершина которого содержит два упорядоченных множества входов и выходов. Алгоритм основан на известном алгоритме Сугиямы. На основе предложенного алгоритма создан компонент визуализации. В статье представлены варианты изображений, которые можно получить с помощью созданного компонента. – Библиогр.: 11 назв.

Visualization of internal representation in the SFP functional programming system / Gordeev D.S. // Parallel programs construction and optimization. – Novosibirsk, 2008. – P. 33–47.

The paper introduces an algorithm of visualization of internal representation of programs. Internal representation can be viewed as a hierarchical directed acyclic graph, each vertex of which contains two ordered sets of input and output points. The algorithm is based on the Sugiyama algorithm. The algorithm was used to create a visualization component. The paper shows what kind of images can be generated. – Refs: 11 titles.

УДК 519.68 + 681.3.06

Технологии обработки изображения со спутника / Гужавина И.В. // Конструирование и оптимизация параллельных программ. – Новосибирск, 2008. – С. 48–70. В статье даётся описание методов обработки изображений, получаемых при съемке со спутника. Предлагается алгоритм пересчета яркости и создания панорамного изображения. Приведены некоторые детали программной реализации описанных и предложенных алгоритмов и результаты работы программы. – Библиогр.: 16 назв.

Technologies of satellite image processing / Guzhavina I.V. // Parallel programs construction and optimization. – Novosibirsk, 2008. – P. 48–70.

The article presents a description of image processing technology applied to images received from satellite. The algorithms of brightness recalculation and panorama stitching are suggested. At the end of the article details of program implementation and results are brought. – Refs: 16 titles.

УДК 519.68 + 681.3.06

Алгоритмы выделения особенностей на изображениях с целью классификации заболеваний растений / Денисюк В.С. // Конструирование и оптимизация параллельных программ. – Новосибирск, 2008. – С. 71–82.

Статья посвящена алгоритмам анализа изображений применительно к задаче выделения особенностей с целью диагностики болезней растений по фотографиям. Рассматриваются статистические и текстурные признаки, которые помогают определить тип заболевания растения. Накопленные данные используются в дальнейшем для поиска похожих изображений. В статье описаны различные алгоритмы выделения контура контрастных объектов, и особенностей текстур. – Библиогр.: 4 назв.

Algorithms of image analysis for feature detection problem in order to diagnose plants diseases / Denisyuk V.S. // Parallel programs construction and optimization. – Novosibirsk, 2008. – P. 71–82.

This article is dedicated to algorithms of image analysis for feature detection problem in order to diagnose plants diseases. Statistical and textural features are analyzed, which helps to define the disease. The data gathered is used in future analyses to detect similar images. Several algorithms of contrast objects contour and texture features detection are described in this article. – Refs: 4 titles.

УДК 519.68 + 681.3.06

Протягивание констант в графе IR2 внутреннего представления языка Sisal / Идрисов Р. И. // Конструирование и оптимизация параллельных программ. – Новосибирск, 2008. – С. 83–94.

Потоковый язык SISAL 3.2, разрабатываемый в Институте систем информатики им. А. П. Ершова, является функциональным языком однократного присваивания. Основная часть оптимизаций в компиляторе языка SISAL выполняется на уровне представления IR2, которое является иерархическим потоковым бесконтурным графом программы. Представление состоит из вершин и портов, где вершины обозначают операции, а порты используются для обозначения начала и окончания дуги передачи значения между операциями.

Эта статья о протягивании значений, алгоритме оптимизации, направленном на подстановку значений, которые могут быть вычислены до исполнения программы. Рассмотрены не только случаи, в которых значение может быть однозначно определено, но и случаи, в которых возможен некоторый набор или диапазон возможных значений. – Библиогр.: 4 назв.

Constant propagation for the Sisal internal representation graph IR2 / Idrisov R.I. // Parallel programs construction and optimization. – Novosibirsk, 2008. – P. 83–94.

Sisal is the functional data flow single assignment language, developed at A. P. Ershov Institute of Informatics Systems. In a SISAL compiler, most optimizations are performed on the intermediate representation IR2, which is the hierarchical acyclic data flow graph of the program. IR2 consists of vertices and ports. Vertices correspond to operations in the source program. Ports are used to mark the beginning and end of the value transfer edge between two operations.

This paper describes constant propagation optimization algorithm used to substitute the values, which could be evaluated before the execution of the program. Multiple value cases are also examined. – Refs: 4 titles.

УДК 519.68 + 681.3.06

Интегрированная визуальная среда поддержки конструирования параллельных программ / Касьянов В. Н. // Конструирование и оптимизация параллельных программ. – Новосибирск, 2008. – С. 95–112.

Статья посвящена системе функционального программирования SFP, создаваемой в Институте систем информатики СО РАН при финансовой поддержке РФФИ. Система SFP предназначена для поддержки разработки высококачественного переносимого программного обеспечения для параллельных

вычислителей на недорогих персональных компьютерах. – Библиогр.: 18 назв.

Integrated visual environment for supporting of parallel program construction / Kasyanov V.N. // Parallel programs construction and optimization. – Novosibirsk, 2008. – P. 95–112.

The paper is devoted to the functional programming system SFP that is under development at the Institute of Informatics Systems in Novosibirsk with financial support from RFBR. The SFP system is intended to support construction of high quality portable software for parallel computers on low cost sequential computers (PC). – Refs: 18 titles.

УДК 519.68 + 681.3.06

Всемирные компьютерные конгрессы ИФИП / Касьянов В. Н. // Методы и инструменты конструирования программ. – Новосибирск, 2007. – С. 113–145.

Данная статья является расширенным текстом доклада автора на заседании N 696 от 28 октября 2008 г. Объединенного семинара ИСИ СО РАН и НГУ «Конструирование и оптимизация программ», посвященном 50-летию со дня образования Отдела программирования Института математики СО АН СССР. – Библиогр.: 47 назв.

IFIP World Computer Congresses / Kasyanov V.N. // Parallel programs construction and optimization. – Novosibirsk, 2008. – P. 113–145.

The paper is an extended text of paper which was presented by the author in 28 October 2008 at the meeting N 696 of the United seminar “Program construction and optimization” of the Institute of Informatics Systems and the Novosibirsk State University, and was devoted to the 50th anniversary of the Programming Department of the Institute of Mathematics, SB AS USSR. – Refs: 47 titles.

УДК 519.68 + 681.3.06

Использование кластеров при вычислении преобразования Меллина для функций в задачах томографии / Касьянова С. Н. // Конструирование и оптимизация параллельных программ. – Новосибирск, 2008. – С. 146–154.

В данной работе описывается эксперимент восстановления функции по её интегралам на семействе прямых, которые пересекают отрезок $[-1, 1]$ на оси x. Носитель интегрируемой функции двух переменных $f(x, y)$ сосредоточен в полосе $-1 \leq x \leq 1$. В алгоритме восстановления используются двумерное прямое и обратное преобразования Меллина, и их вычисления осуществляются параллельно. – Библиогр.: 7 назв.

Using the clusters for computing the Mellin transform of functions in tomography tasks / Kasyanova S. N. // Parallel programs construction and optimization. – Novosibirsk, 2008. – P. 146–154.

An experiment of function reconstruction from its integrals on the family of straight lines cross the segment $[-1, 1]$ on the x-axis is described. The carrier of intergrable function of two variables $f(x, y)$ is concentrated in the region $-1 \leq x \leq 1$. Two-dimensional direct and reverse Mellin transformations are used in the algorithm of reconstruction, and their computations are performed in parallel. – Refs: 7 titles.

УДК 519.68 + 681.3.06

Автоматическое выявление таксономии в области преобразований программ на основе анализа семантических связей в публикациях / Малинина Ю. В. // Конструирование и оптимизация параллельных программ. – Новосибирск, 2008. – С. 155–163.

Данная статья является развитием подхода, описанного в [1] и уточняет предложенную модель, которая позволяет решать ряд прикладных задач, особенно связанных с построением автоматической классификаций предметной области на основе выборки публикаций. – Библиогр.: 21 назв.

Automatic taxonomy detection in the field of program transformations based on the analysis of semantic connections in publications / Malinina Yu.V. // Parallel programs construction and optimization. – Novosibirsk, 2008. – P. 155–163.

This article continues the approach described in [1] and extends the offered model, which allows solving several applied problems, particularly those related to building automatic taxonomy of application domain based on the set of publication. – Refs: 21 titles.

УДК 519.68 + 681.3.06

О семинаре Зыкова в Новосибирске / Мельников Л.С. // Конструирование и оптимизация параллельных программ. – Новосибирск, 2008. – С. 164–173.

Цель настоящих заметок - дать хотя бы мимолетный взгляд на семинар Зыкова с точки зрения студента Новосибирского государственного университета (НГУ), позднее стажера-исследователя Института математики СО АН (ИМ). Личные впечатления автора о семинаре относятся к периоду с 1964 по 1969. Таким образом, это взгляд из того времени. – Библиогр.: 7 назв.

About Zykov's seminar in Novosibirsk / Mel'nikov L.S. // Parallel programs construction and optimization. – Novosibirsk, 2008. – P. 164–173.

The aim of these notes is to present a brief glance upon the seminars held by Zykov from the point of view of a student of the Novosibirsk State University, later a research intern of the Institute of Mathematics of the SB AS USSR. The article contains personal impressions of the author in 1964-1969. In a way, it?s a view upon that period. – Refs: 7 titles.

УДК 519.68 + 681.3.06

История развития суперкомпьютерной вычислительной техники / Мурзин Ф. А., Полетаев С. А. // Конструирование и оптимизация параллельных программ. – Новосибирск, 2008. – С. 174–215.

В статье приведен исторический обзор развития суперкомпьютеров, и кратко рассматриваются статистика и анализ тенденций в области высокопроизводительных вычислений. – Библиогр.: 14 назв.

History of development of supercomputers / Murzin F.A., Poletaev S.A. // Parallel programs construction and optimization. – Novosibirsk, 2008. – P. 174–215.

In the paper, the historical review of development of supercomputers is given, and statistics and analysis of tendencies in the area of high-performance computations are briefly considered. – Refs: 14 titles.

УДК 519.68 + 681.3.06

Пособие по написанию разного рода деловых текстов / Несговорова Г.П. // Конструирование и оптимизация параллельных программ. – Новосибирск, 2008. – С. 216–259.

Данное пособие написано в помощь студентам, которые испытывают трудности при написании разного рода документации, а именно: курсовых и дипломных работ, статей, отчетов, инструкций к программам, различных документов, деловых писем и т.д. Как правило, это студенты негуманитарных специальностей, у которых возникают сложности при создании такого рода текстов.

Пособие состоит из 2-х частей: теоретической и практической, каждая из которых включает вопросы для самоконтроля после каждого раздела. В конце каждой части прилагается список литературы, а в приложении ко 2-ой части предлагаются упражнения, снабженные справками и пояснениями. – Библиогр.: 14 назв.

Manual in writing different kind of business texts / Nesgovorova G.P. // Parallel programs construction and optimization. – Novosibirsk, 2008. – P. 216–259.

This paper aims to help students in writing different kinds of documents, such as term papers and research works, accounts, program instructions, letters and so on. As a rule, students facing such tasks are those specializing in science, and not humanities. They may experience certain difficulties in creating such texts.

The paper consisits of two parts: theory and application. There are questions for self-control after each section. In the end of each part there is a list of literature. Exercises with explanations and references are given in the appendix of the second part. – Refs.: 14 titles.

УДК 519.68 + 681.3.06

Поисковая система с элементами лингвистического анализа / Перфильев А. А. // Конструирование и оптимизация параллельных программ. – Новосибирск, 2008. – С. 259–269.

Эта работа посвящена актуальной проблеме эффективного Интернет-поиска. Основная цель – разработать эффективные алгоритмы сравнения предложений, основанные на схемах синтаксического анализа, и в результате разработать поисковую систему, основанную на таком подходе. Интересно, что достаточно оставаться на уровне синтаксиса и грамматики, осуществляя оценку релевантности шаблонов в тексте. Синтаксические диаграммы задают первичную структуру в тексте, которая позволяет нам выбирать и оценивать предложения и фразы из текста, имеющие соответствующие синтасемантические связи между ключевыми словами. В результате была разработана поисковая система, которая автоматизирует поиск в Интернете. Проведенные испытания этой системы показали хорошую релевантность. – Библиогр.: 3 назв.

Searching system with elements of linguistic analysis / Perfiliev A.A. // Parallel programs construction and optimization. – Novosibirsk, 2008. – P. 259–269.

This work is dedicated to the relevant problem of effective context-related text search in the Internet. The main goal is to develop new efficient algorithms of sentence comparison based on syntactic analysis schemes and consequently

develop a search system based on this approach. Worth mentioning is that it suffices to stay on the level of syntax and grammar constructions while evaluating the relevance of templates in the text. Syntactic diagrams set up a primary structure in the text which allows to select and evaluate sentences and phrases from the text which have corresponding semantic links between keywords. As the result a search system which automates Internet searching was developed. Test have shown good result relevance. – Refs.: 3 titles.

УДК 519.68 + 681.3.06

Параллельные вычисления на графических процессорах / Полетаев С. А. // Конструирование и оптимизация параллельных программ. – Новосибирск, 2008. – С. 270–300.

CUDA (*Compute Unified Device Architecture*) – технология, разработанная компанией Nvidia, позволяющая программистам реализовывать на языке программирования Си алгоритмы, выполнимые на графических процессорах GeForce восьмого поколения и старше (GeForce 8 Series, GeForce 9 Series, GeForce 200 Series), Nvidia Quadro и Tesla компании Nvidia. В статье приведен аналитический обзор технологии CUDA; описано оборудование и особенности программирования. – Библиогр.: 3 назв.

Parallel calculations on the graphic processors / Poletaev S.A. // Parallel programs construction and optimization. – Novosibirsk, 2008. – P. 270–300.

CUDA (*Compute Unified Device Architecture*) is the technology, developed by the company Nvidia, which makes possible for programmers to realize in the programming language C the algorithms, feasible on the graphic processors GeForce of the eighth generation and older (GeForce of 8 Series, GeForce of 9 Series, GeForce of 200 Series), Nvidia Quadro and Tesla of the company Nvidia. The article presents an instant analysis of the CUDA technology; equipment and special features of programming are described. – Refs.: 3 titles.

УДК 519.68 + 681.3.06

Генерация исполняемых тестов для компилятора / Стасенко А. П. // Конструирование и оптимизация параллельных программ. – Новосибирск, 2008. – С. 301–313.

В статье описывается генератор, основанный на формализме параметрической контекстно-свободной грамматики и допускающий генерацию контекстно-зависимых языков, достаточных для автоматического построения компилируемых программ, имеющих детерминированное и конечное исполнение. Путём анализа различий вывода полученных программ, скомпилированных с оптимизациями и без них, были найдены трудно обнаруживаемые ошибки в бета версии компилятора промышленного уровня. – Библиогр.: 5 назв.

Generation of executable tests for compiler / Stasenko A.P. // Parallel programs construction and optimization. – Novosibirsk, 2008. – P. 301–313.

The paper describes a generator that uses the formalism of parametric context-free grammar and allows generation of context-sensitive languages suitable for automatic construction of programs that can be compiled and have deterministic and finite execution. By analyzing differences in the output of these programs compiled with optimizations and without them, subtle errors in beta version of the industrial level compiler were found. – Refs.: 5 titles.

УДК 519.68 + 681.3.06

Деревья клик хордального графа и деревья подграфов в теории графов / Турсунбай кызы Ы. // Конструирование и оптимизация параллельных программ. – Новосибирск, 2008. – С. 314–321.

Обобщение деревьев клик посредством выбора других видов индуцированных подграфов позволяет определенные понятия и результаты хордальных графов переносить на другие классы графов, такие как строго хордальные графы, полностью сбалансированные гиперграфы, хордально двудольные графы и др. Также в работе приведены основные утверждения, позволяющие распознать и построить дерево клик для графа G. – Библиогр.: 16 назв.

The clique tree of chordal graph and subgraph trees in graph theory / Tursunbay kyzy Y. // Parallel programs construction and optimization. – Novosibirsk, 2008. – P. 314–321.

Generalizing clique trees by selecting other sorts of induced subgraphs allows certain concepts and results of chordal graphs to be transferred to other classes of graphs, such as strongly chordal graphs, totally balanced hypergraphs, chordal bipartite graphs etc. – Refs.: 16 titles.

КОНСТРУИРОВАНИЕ И ОПТИМИЗАЦИЯ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ

Под редакцией
проф. Виктора Николаевича Касьянова

Рукопись поступила в редакцию 15. 10. 2008
Ответственный за выпуск Г. П. Несговорова
Редактор Т. М. Бульонкова

Подписано в печать 27. 12. 2008
Формат бумаги 60 × 84 1/16 Объем 19,0 уч.-изд.л., 20,75 п.л.
Тираж 75 экз.

Центр оперативной печати "Оригинал 2",
г.Бердск, ул. Островского, 55, оф. 02, тел. (383) 214-45-35