В. А. Маркин, С. А. Маркина

ПРОЕКТ СИСТЕМЫ ДЛЯ БЫСТРОГО ПРОТОТИПИРОВАНИЯ РАСПАРАЛЛЕЛИВАЮЩЕГО КОМПИЛЯТОРА. УНИВЕРСАЛЬНОЕ ВНУТРЕННЕЕ ПРЕДСТАВЛЕНИЕ СИСТЕМЫ*

1. ВВЕДЕНИЕ

Развитие ЭВМ с параллельными архитектурами и высокопроизводительных вычислительных систем ставит перед системными программистами задачи по созданию новых технологических подходов и их эффективному использованию. Одним из направлений является создание языков программирования с явно определенными параллельными семантическими конструкциями, что, впрочем, не облегчает процесс написания параллельных программ. Процесс человеческого мышления ближе подходит к последовательной модели вычислений, чем можно объяснить, почему до сих пор разрабатывается большое количество последовательных алгоритмов любого уровня сложности и в различных прикладных областях. Поэтому вторым основным направлением по созданию программных систем для параллельных архитектур является разработка распараллеливающих и оптимизирующих компиляторов.

В отличие от явного параллелизма, связанного с расширением существующих языков средствами управления параллельными вычислениями, автоматическое распараллеливание программ свободно от многих недостатков первых систем, но имеет свои собственные, связанные с ориентированностью на определенную «среднюю» программу. Различные распараллеливающие системы включают в себя различные наборы оптимизирующих и реструктурирующих преобразований и средства распараллеливания вычислений. Каждая система определяет свой класс входных программ, в которых она может выявить параллелизм. Также не последнюю роль играют конечные параллельные архитектуры, на которые ориентированы распараллеливающие системы [3].

Описываемая в данной работе система для быстрого прототипирования распараллеливающего компилятора выполняется в рамках проекта ПРО-

* Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (грант № 01-01-794) и Министерства образования РФ.

ГРЕСС [1, 2], проводимого в лаборатории конструирования и оптимизации программ ИСИ СО РАН. Началом работ по созданию этой системы для авторов послужили уже полученные программные наработки в области исследования свойств программ в рамках проекта ПРОГРЕСС (в частности, по анализу программных зависимостей; исследованию различных теоретико-графовых внутренних представлений программ и их эффективности для оптимизирующих преобразований). Одной из целей проекта ПРОГРЕСС является исследование свойств различных классов языков программирования, позволяющее выявлять внутренний параллелизм входных программ. В связи с этим накапливается большая база программных библиотек, решающих задачи теории построения трансляторов для высокопроизводительных архитектур, которые пока трудно совместимы между собой, что значительно усложняет их совместное использование. Данная проблема и требует создания подобной системы.

Далее в разд. 2 описывается проект системы по созданию прототипа распараллеливающего компилятора. Даются формальные определения составных частей системы, ее наполнение и содержание. В разд. 3 приводится описание расширяемого универсального внутреннего представления, которое является стержнем всей системы. Определяются структуры внутреннего представления, описываются его компоненты и их взаимосвязь. В заключении приведен перечень проделанной работы и представлены планы на будущее.

2. ОПИСАНИЕ ПРОЕКТА СИСТЕМЫ

Работа с новыми распараллеливающими программными комплексами требует определенных знаний и навыков. Следовательно, возникает необходимость в создании различных программных систем для обучения параллельному программированию. К тому же при построении эффективных распараллеливающих компиляторов для конкретной целевой архитектуры необходимо определять оптимальное множество средств анализа свойств программ, множество оптимизирующих преобразований и т.п. В теории распараллеливания и оптимизации постоянно возникают новые подходы и алгоритмы, анализ которых необходимо проводить в контексте уже существующих, что позволяет определить их оптимальность и эффективность.

Данный проект направлен на создание системы для построения прототипа распараллеливающего компилятора, а также изучения свойств про-

грамм, исследования алгоритмов распараллеливания и оптимизации. Предполагается, что система будет содержать:

1) инструменты для быстрого прототипирования распараллеливающих компиляторов и оптимизирующих компиляторов для различных целевых архитектур, таких как VLIW [14], суперскалярных и мультипроцессорных с разделенной памятью (NUMA [15] и др.). Система должна позволять из имеющегося набора алгоритмов создавать, исследовать и исполнять прототип компилятора, функциональная наполняемость которого будет задаваться пользователем;
2) средства и инструменты для унификации программирования различных подходов и алгоритмов, в частности, универсальное расширяемое внутреннее представление и библиотеку работы с ним;
3) инструменты для подключения к системе новых алгоритмов и преобразований; для проведения исследований эффективных форм промежуточных представлений программ, оптимизирующих и реструктурирующих преобразований (оптимальных условий и стратегий их использования для различных классов программ и вычислительных систем);
4) средства визуального проектирования и проведения обучения студентов методам программирования и оптимизации для параллельных архитектур.

Для достижения данных целей система создается как инструмент для манипулирования программами. Это означает пошаговое преобразование программ, начиная с исходного текста программы на одном из языков высокого уровня (например, C, Паскаль), и далее, пропуская программу через различные блоки, получаем текстовое представление распараллеленной программы либо исполняемый код для запуска программы на одной из целевых архитектур. Центральным понятием работы системы является СЦЕНАРИЙ прототипа создаваемого компилятора. В сценарии определяются множество и порядок исполнения компонент системы, которые делятся на два вида:

- *функциональные компоненты* — реализованные алгоритмы, связанные с преобразованием внутреннего представления;
- *инструментальные компоненты* — интерактивные компоненты, ориентированные на взаимодействие с пользователем, внешние графические интерфейсы.

Однотипные функциональные компоненты объединяются в *функциональные блоки*, такие как блок синтаксического разбора (перевод во внутреннее представление), блок внутреннего представления программ, блок

анализа программных зависимостей, блок оптимизирующих и распараллеливающих преобразований, блок кодогенерации. К инструментальным компонентам относятся: текстовый редактор, графический редактор, редактор внешних связей и т.д. Функциональные и инструментальные компоненты реализуются как внешние библиотеки (dll либо элементы ActiveX) с учетом заданных интерфейсов взаимодействия с системой. Регистрация либо загрузка соответствующих компонент происходит во время запуска системы либо во время начала работы сценария.

Для задания сценария в систему вводится *язык описания сценариев* и, соответственно, интерпретатор для данного языка. Основными объектами языка сценариев будут служить библиотеки, вложенные в них запрограммированые алгоритмы и методы, а также типы объектов внутреннего представления системы. Весь сценарий разбивается на участки исполнения, на каждом из которых определены множество подключаемых компонент системы, последовательность и условия их вызовов. Языку сценариев дается одна из основных ролей в разрабатываемой системе. По сути дела, создание прототипа компилятора будет заключаться в составлении сценария системы и его дальнейшего интерпретирования. На данный момент механизм сценария находится на стадии проработки.

С учетом выше сказанного, система рассматривается как КОНСТРУКТОР для построения прототипа компилятора, кирпичиками которого являются различные функциональные и инструментальные компоненты. С помощью заданного сценария пользователь сможет построить, исполнить и исследовать прототип распараллеливающего компилятора.

Описание современных подходов к построению распараллеливающих компиляторов можно найти в [3]. Также общую теорию по созданию трансляторных систем и подходы к их реализации можно найти в [4–6]. Смотри также [11, 12].

Система условно разбивает процесс трансляции на следующие этапы: разбор входной программы и перевод во внутреннее представление (front-end); оптимизирующие и реструктурирующие преобразования (трансформация); кодогенерация.

Для реализации каждого этапа необходимо разработать следующие компоненты.

Для этапа front-end:

- расширяемое универсальное внутреннее представление (ВП), а также библиотеку для работы с ним;

- разбор командной строки и настройку среды, библиотеку работы с конфигурационными файлами;
- библиотека парсеров для перевода текста входной программы во ВП. Данная библиотека необходима для создания многоязыковой системы. На текущий момент предполагается реализация поддержки программ на языках С и Паскаль, в дальнейшем множество языков может быть расширено путем написания дополнительных библиотек и подключения их к системе. Такой же принцип расширяемости будет использоваться и для других компонент;
- библиотека алгоритмов анализа программных зависимостей [3, 16]. К настоящему времени реализованы несколько алгоритмов анализа зависимостей по данным (НОД-тест, тест Банержи, модифицированный алгоритм Шостака [7]) и алгоритм построения подграфа зависимостей по управлению. В данную библиотеку также будут включены алгоритмы построения SSA-формы и теоретико-графовых промежуточных представлений программ (ГПЗ, ИГЗ, идеограф).

Для этапа трансформации:

- библиотека оптимизирующих преобразований;
- библиотека реструктурирующих (в т.ч. и распараллеливающих) преобразований.

Для этапа кодогенерации:

- библиотека кодогенераторов с целью получения объектного кода для различных целевых архитектур.

В систему предполагается включить множество программных библиотек и интерфейсы взаимодействия с системой, используя которые, пользователь сможет создать новые библиотеки алгоритмов и подключить их к системе. Система будет иметь дружественный и понятный интерфейс, что позволит проводить образовательный процесс по вопросам теории трансляции программ и их распараллеливания.

3. РАСШИРЯЕМОЕ УНИВЕРСАЛЬНОЕ ВНУТРЕННЕЕ ПРЕДСТАВЛЕНИЕ СИСТЕМЫ

Основной компонентой системы является расширяемое универсальное внутреннее представление программы, которое служит для связи между блоками системы. Главными целями при его проектировании являлись универсальность (общность), расширяемость и гибкость. Данное представление должно было позволить переводить во внутреннее отображение многие ши-

роко распространенные языки программирования, а также обобщить различные теоретико-графовые внутренние представления, разработанные за последнее время [8–10, 13].

Модель разработанного внутреннего представления можно определить следующей схемой, как показано на рис. 1.

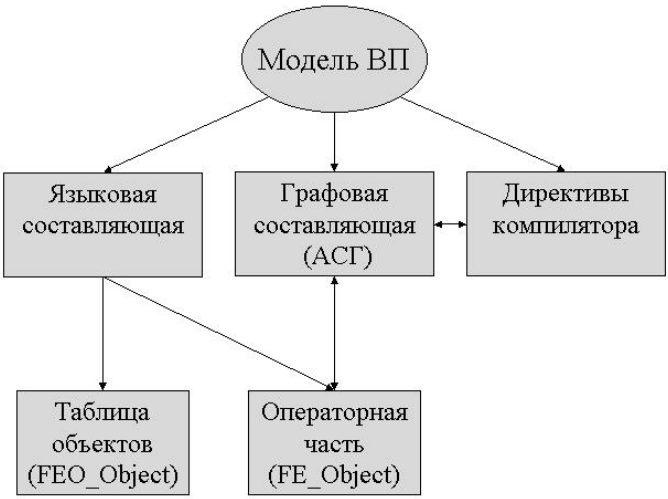

Рис. 1

Языковая составляющая ВП — это множество синтаксических и семантических объектов, иерархически упорядоченных, отображающих большинство общих конструкций входных языков. В качестве базового класса для всех объектов ВП системы используется абстрактный класс OM_object, на базе которого реализован механизм для получения информации об объекте в режиме исполнения (RTTI). Внутренняя реализация данного класса позволяет вести регистрацию всех наследуемых от него типов, динамического создания объектов и контроль типов (рис. 2).

Все объекты распределяются на области видимости, тем самым ВП включает в себя таблицы объектов для каждой области видимости и операторную часть.

Таблицы объектов включают в себя множество определений типов и внутренних объектов. Типы делятся на простые (такие как целый, вещест-

венный, символьный), составные (такие как массив, возможно, динамический, структура) и указатель. Внутренние объекты — это константы, метки и переменные (см. FEO_object иерархию, рис. 2).

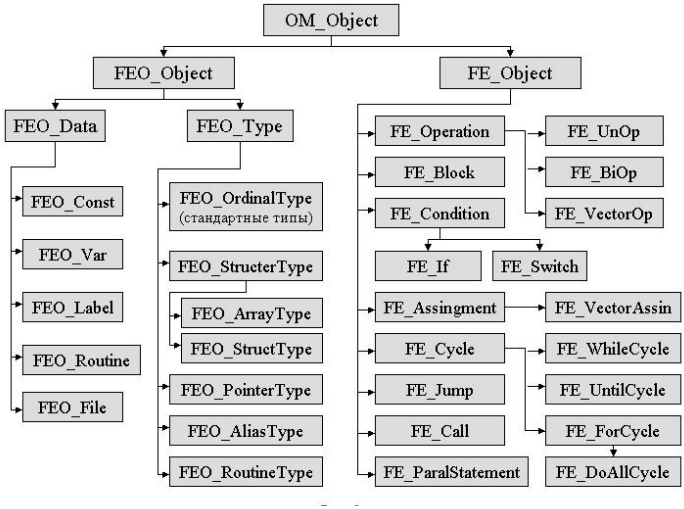

Рис. 2

Операторная часть представлена в виде абстрактного синтаксического графа (который является частью графовой составляющей), узлы которого делятся на семантические и управляющие вершины (см. FE_object иерархию, рис.2). К семантическим вершинам относятся if-вершины, loop-вершины, или операторные вершины. К управляющим относятся вершины-директивы компилятора, или дополнительные вершины ВП, такие как R-вершины графа программных зависимостей. Семантические вершины делятся на вершины высокого и низкого уровней. Вершины операторов низкого уровня можно сравнить с командами модели какой-либо вычислительной машины, на последовательность которых можно разбить операторы вершин высокого уровня и проводить какие-либо оптимизирующие преобразования, такие как вычисление подвыражений, протягивание констант и т.д. Множество конструкций, соответствующих большинству конструкций императивных языков программирования, расширено операторами с параллельной семантикой исполнения.

Любой объект ВП содержит атрибуты, необходимые для визуализации программы. Для каждого объекта хранятся координаты его текстового про-

образа — это номер строки и номер литеры в строке по текстовому файлу для начальной и конечной литер конструкции в тексте исходной программы. К атрибутам визуализации также можно отнести идентификатор исходной программы; вид объекта, цикла, безусловного перехода; информацию о переименовании объектов в тексте исходной программы.

Помимо основных объектов в ВП введены системно-зависимые объекты, так называемые директивы компилятора. Они могут быть установлены пользователем между запуском различных частей компилятора и указывать системе на вызов каких-либо библиотечных алгоритмов, функций и т.д.

Поскольку одной из задач при проектировании системы являлось обобщение и реализация различных теоретико-графовых представлений, была выделена отдельно графовая составляющая ВП — абстрактный синтаксический граф (АСГ) — для отображения графовой структуры программы. На первом этапе трансляции (front-end) строится абстрактное синтаксическое дерево (АСД). Затем в зависимости от сценария работы компилятора строятся так называемые графовые тени (ГТ), ориентированные на явные описания свойств транслируемых программ, например: граф зависимостей по данным, граф программных зависимостей, SSA-форма и т.д. Абстрактный синтаксический граф является объединением АСД и графовых теней, при котором к АСД добавляются новые графовые объекты (вершины и ребра) с особыми метками, сигнализирующими о принадлежности данного объекта некоторой графовой тени. При программной реализации ГТ должен быть предоставлен метод ее восстановления из АСГ.

Основной целью при проектировании ВП являлась его расширяемость. Поэтому ВП организовано в виде иерархии классов и библиотеки работы с ними. При разработке новых функциональных блоков системы пользователь может либо по-своему интерпретировать зарезервированное поле, присутствующее в любом из классов, либо наследовать от уже существующих классов новые типы объектов.

Данное ВП спроектировано в объектно-ориентированном стиле, библиотека работы с ними реализована на языке C++.

4. ЗАКЛЮЧЕНИЕ

На данном этапе авторами проекта реализовано ВП системы, а также переработана часть алгоритмов потокового анализа и построения теоретико-графовых представлений программ. Реализованы некоторые инструментальные компоненты: текстовый и графовый редакторы. Дальнейший этап

работ будет связан с формализацией и реализацией языка сценариев системы, с учетом тех целей, которые были ему заданы, а именно — поддержка *динамичности* по отношению к компонентам системы и элементам внутреннего представления. Выход предварительной версии системы предполагается к началу второго полугодия 2002 года. Она должна быть доступна через сайт лаборатории.

Особые слова благодарности хочется выразить научному руководителю д.ф-м.н. Евстигнееву В.А.

СПИСОК ЛИТЕРАТУРЫ

1. **Kasyanov V.N., Evstigneev V.A.** The PROGRESS program manipulation system // Parallel Computer Technologies/ Proc. Intern. Conf., Obninsk, 1993. — Vol. 3. — P. 651–656.
2. **Kasyanov V.N., Evstigneev V.A., Gorodniaia L.** The system PROGRESS is a tool for parallelizing compiler prototyping // Proc. of Eighth SIAM Conf. on Parallel Processing for Scientific Computing (PPSC-97), Minneapolis, 1997. — P. 301–306.
3. **Wolfe M.** High performance compilers for parallel computing. — Addison-Wesley, 1996. — P. 570.
4. **Aho A.V., Sethi R., Ullman J.D.** Compilers: Principles, Techniques and Tools. – Addison-Wesley, 1986.
5. **Wirth N.** Compiler construction. — Addison-Wesley, 1996. — P. 176.
6. **Касьянов В.Н. , Поттосин И.В.** Методы построения трансляторов. — Новосибирск: Наука, 1986. — 344 с.
7. **Логачева С.А.** Анализ зависимостей по данным на базе алгоритма Шостака // Поддержка супервычислений и интернет-ориентированные технологии. — Новосибирск, 2001. — С. 31–43.
8. **Шелехов В.И.** Внутреннее представление программ в системе Сократ. — Новосибирск, 1993. — 44 с. — (Препр./ РАН. Сиб. отд-ние. ИСИ; № 15.)
9. **Евстигнеев В.А.** О некоторых формах промежуточного представления программ //Конструирование и оптимизация программ. — Новосибирск, 1993. — С. 60–68.
10. **Маркин В.А.** Промежуточное представление программ в распараллеливающих компиляторах // Проблемы систем информатики и программирования. — Новосибирск, 1999. — С. 163–182.
11. **Евстигнеев В.А., Касьянов В.Н.** Сводимые графы и граф-модели в программировании. — Новосибирск: ИДМИ, 1999. — 288 с.
12. **Евстигнеев В.А., Касьянов В.Н.** Теория графов: алгоритмы обработки бесконтурных графов. — Новосибирск: Наука, 1998. — 385 с.
13. **Маркина С.А., Маркин В.А.** Проект системы для создания прототипа распараллеливающего компилятора. Расширяемое внутреннее представление системы

// Материалы конф. молодых ученых по математике, математическому моделированию и информатике, 4-6 декабря 2001 г. — Новосибирск, 2001. — С. 47–48.

14. **Евстигнеев В.А.** VLIW-машины: развитие архитектуры и принципов построения программного обеспечения // Системная информатика. Вып. 4: Методы теоретического и системного программирования. — Новосибирск: Наука, 1995. — С. 304–333.
15. **Евстигнеев В.А.** NUMA-архитектура: некоторые особенности компиляции и генерации кода // Поддержка супервычислений и интернет-ориентированные технологии. — Новосибирск, 2001. — С. 44–53.
16. **Евстигнеев В.А.** Анализ зависимостей: состояние проблемы // Системная информатика. Вып. 7: Проблемы теории и методологии создания параллельных и распределительных систем. — Новосибирск: Наука, 2000. — С. 112–173.