На правах рукописи

УДК 004.4'416

Макошенко Денис Валентинович

АНАЛИТИЧЕСКОЕ ПРЕДСКАЗАНИЕ ВРЕМЕНИ ИСПОЛНЕНИЯ ПРОГРАММ И ОСНОВАННЫЕ НА НЕМ МЕТОДЫ ОПТИМИЗАЦИИ

05.13.11 – математическое и программное обеспечение
вычислительных машин, комплексов и компьютерных сетей

АВТОРЕФЕРАТ

диссертации на соискание ученой степени
кандидата физико-математических наук

Новосибирск – 2011

Работа выполнена на кафедре алгебры и дискретной математики факультета математики, механики и компьютерных наук Южного федерального университета

Научный руководитель: Штейнберг Борис Яковлевич,
доктор технических наук

Официальные оппоненты: Плоткин Арнольд Леонидович
доктор технических наук

Идрисов Ренат Искандерович,
кандидат физико-математических наук

Ведущая организация: Ведущая организация: Институт точной механики и вычислительной техники им. С. А. Лебедева РАН

Защита состоится 26 декабря 2011 г. в 16 ч. 00 мин. на заседании диссертационного совета ДМ 003.032.01 в Институте систем информатики им. А.П. Ершова Сибирского отделения РАН по адресу: 630090, г. Новосибирск, пр. ак. Лаврентьева, 6.

С диссертацией можно ознакомиться в читальном зале ИСИ СО РАН (г. Новосибирск, пр. ак. Лаврентьева, 6).

Автореферат разослан _____ ноября 2011г.

Ученый секретарь
Диссертационного совета,

к.ф.–м.н. Мурзин Ф.А.

ОБЩАЯ ХАРАКТЕРИСТИКА РАБОТЫ

Актуальность проблемы. Современные научные исследования, разработка оборонных технологий, растущая функциональность мобильных устройств требуют все более высокой производительности программного обеспечения.

Помимо аппаратной составляющей наибольший вклад в повышение производительности дают операционная система и оптимизирующий компилятор. Причем, согласно измерениям, приведенным на www.spec.org, компилятор вносит значительно больший вклад в производительность программы, чем операционная система.

Следует отметить, что постоянное повышение производительности вычислительных архитектур, как правило, влечет за собой усложнение этих архитектур и, как следствие, усложнение их эффективного использования. Для эффективного исполнения на конкретной вычислительной архитектуре программа должна быть оптимизирована с учетом особенностей этой архитектуры. Обычно это достигается либо ручной оптимизацией текста программы на ассемблере, либо использованием компилятора с языка высокого уровня, автоматически оптимизирующего программу под выбранную архитектуру. Последний способ более распространен, поскольку он значительно сокращает время разработки программного продукта и не требует от программистов детальных знаний о целевой архитектуре.

Процесс повышения производительности программы компилятором может быть представлен как некая комбинация преобразований, выбранная из заданного набора. Одной из задач построения оптимизирующего компилятора является задача выбора для каждой оптимизируемой программы такой комбинации преобразований из имеющегося набора, чтобы после ее применения результирующая программа наиболее эффективно

использовала аппаратные ресурсы архитектуры. Сложность задачи обусловлена двумя факторами:

- Преобразование может влиять, как на эффективность использования процессора, так и на эффективность использования подсистемы памяти. При этом преобразование может оказать положительное влияние на оптимальность использования одного ресурса, и, одновременно, отрицательное влияние на оптимальность использования другого ресурса.
- Преобразования могут зависеть друг от друга: без применения одного преобразования невозможно применение другого; наоборот, применение какого-то преобразования может блокировать применение другого преобразования; наконец, примененные по отдельности, преобразования могут приводить к понижению оптимальности кода, в то время как их комбинация может улучшать его оптимальность.

Широко используемые компиляторы, например, gcc, MSVS, Intel compiler, проводят комбинации преобразований, которые жестко подчиняются опциям, специфицированным в командной строке. Для некоторых конкретных оптимизируемых программ такие жестко зафиксированные комбинации преобразований могут привести к неоптимальному коду. Разработчики компиляторов зачастую выбирают некоторый пакет программ, на котором путем ручного подбора достигается хорошее взаимодействие преобразований.

Некоторыми авторами исследовался метод перебора различных сочетаний параметров командной строки компилятора и машинное обучение, которые влияют на выбор проводимых преобразований. Однако задачу можно считать нерешенной для промышленных компиляторов.

Большое подмножество преобразований, проводимых компилятором, решают задачи, которые являются NP-полными. Поэтому, зачастую, при разработке компилятора оптимальные алгоритмы заменяются эвристическими, чтобы сократить время компиляции до приемлемого.

Возможность контролировать эвристики, управляющие качеством преобразований, дает пользователю компилятора свободу в вопросе выбора, что в данный момент является более важным: время компиляции программы или качество проведенной оптимизации. Современные компиляторы дают возможность контролировать набор проводимых преобразований, но не оптимальность каждого конкретного преобразования.

В рамках диссертационной работы предложены новые методы оптимизации программ, позволяющие сделать выбор между временем, затраченным на компиляцию, и быстродействием полученного кода.

Цель и задачи работы. Целью данной диссертации является повышение эффективности использования системы процессор + память. Достижение цели связано с решением следующих задач:

1. Построение модели, позволяющей на этапе компиляции программы оценивать время ее исполнения.
2. Разработка семейства алгоритмов оптимизации распределения регистров, позволяющих с помощью выбора значений параметров оптимизировать быстродействие кода, не превышая лимита времени компиляции.
3. Разработка семейства алгоритмов выбора оптимальной комбинации преобразований программы, также позволяющих с помощью выбора значений параметров оптимизировать быстродействие кода, не превышая лимита времени компиляции.

Методы исследования. В диссертационной работе использовались различные методы и математические инструменты, такие как: теория графов, теория алгоритмов, элементы теории множеств, теория преобразования и оптимизации программ и др.

Научная новизна. Предложены новые алгоритмы, позволяющие более оптимально использовать иерархию памяти путем более эффективного распределения регистров.

Построена теоретико-графовая модель программы, описывающая зависимости между возможными преобразованиями.

Разработаны новые алгоритмы выбора оптимальной комбинации преобразований, учитывающие взаимное влияние преобразований друг на друга, влияние преобразований на использование ресурсов процессора и подсистемы памяти.

В частности, алгоритмы распределения регистров и выбора оптимальной комбинации преобразований имеют параметр, позволяющий минимизировать время исполнения программы, не превышая лимита времени компиляции.

Практическая ценность. Результаты диссертации могут быть использованы при разработке оптимизирующих компиляторов нового поколения, которые предоставят программисту возможность делать выбор между качеством оптимизации программы и временем, затрачиваемым на оптимизацию. Также предложенные методы оптимизации являются хорошо распараллеливаемыми, что позволяет значительно сократить время оптимизации программы за счет использования современных многоядерных архитектур.

Результаты диссертации использовались при обучении студентов кафедры микропроцессорных технологий факультета РТК Московского физико-технического института современным методам программирования и оптимизации для различных вычислительных архитектур.

Апробация работы. Основные положения диссертации обсуждались на следующих конференциях и семинарах:

1. Интеллектуальные и многопроцессорные системы ИМС-2003, Международная научно-техническая конференция, Дивноморское, Россия, 22-27 сентября 2003.
2. IEEE EAST-WEST DESIGN & TEST SYMPOSIUM 2009 MOSCOW, RUSSIA, September 18–21, 2009.

3. Современные проблемы фундаментальных и прикладных наук, 52-я научная конференция МФТИ, Долгопрудный, 27-28 ноября, 2009.
4. «Автоматическое распараллеливание программ», семинар кафедры алгебры и дискретной математики мехмата Южного федерального университета, 12 сентября 2005.

Публикации. Основные результаты диссертации опубликованы в семи работах, среди них четыре статьи, из которых две опубликованы в журналах из перечня ВАК [1], [2], одна в зарубежном журнале [3], одна в сборнике научных трудов [4], и три публикации в сборниках тезисов докладов конференций [5], [6], [7].

Структура и объем работы. Диссертационная работа состоит из введения, четырех глав, заключения и списка литературы. Объем диссертации – 120 стр. Список литературы содержит 109 наименований.

СОДЕРЖАНИЕ РАБОТЫ

Во введении обоснована актуальность проводимых исследований, сформулирована цель диссертационной работы, показана новизна и практическая значимость результатов, указаны положения, выносимые на защиту, и кратко аннотировано содержание глав.

В главе 1 строится модель времени вычислений, позволяющая оценивать время исполнения процедуры программы во время ее компиляции.

Пусть G - это граф потока управления (ГПУ) процедуры P. Рассмотрим некоторый простой путь *path* в графе G и его множество вершин *BB*. Обозначим через *enter* начало пути *path*, через *exit* конец пути *path*. Множество вершин *BB* называется линейным участком, если в процедуре нет инструкций передачи управления в инструкции, соответствующие вершинам *BB,* кроме, быть может, *enter*, и ни одна инструкция, соответствующая вершинам *BB,* не является инструкцией перехода, кроме, быть может, *exit*. Пусть *resource(BB)* – это оценка минимального времени линейного участка

BB при данных ресурсах вычислительной системы, являющаяся модификацией формулы, предложенной Рау.

Обозначим через *ГПЗ(BB)* подграф графа программных зависимостей (ГПЗ), порожденный объединением множества вершин *BB* и вершин, из которых в вершины *BB* ведут дуги. Обозначим через *weight(BB)* сумму весов дуг критического пути в *ГПЗ(BB)*.

Пусть *NumExec(BB)* – это количество исполнений линейного участка во время исполнения программы. Тогда *ETM(BB) = max(resource(BB), weight(BB)) * NumExec(BB)* – это время вычисления *BB*.

Разобьем вершины *G* на множество линейных участков *BBlocks(G)*. Тогда $ETM(P) = \sum_{BB \in BBlocks(G)} ETM(BB)$.

В главе 2 рассматривается задача размещения виртуальных регистров в физических регистрах, возникающая при трансляции программы с языка высокого уровня в машинные коды.

Каждой программе может быть поставлен в соответствие граф несовместимости (ГН), моделирующий распределение физических регистров. Если хроматическое число ГН не превосходит количество доступных в архитектуре физических регистров, то виртуальные регистры могут быть размещены в физических без добавления инструкций обращения к памяти.

В данной главе предлагаются новые алгоритмы раскраски ГН.

Рассмотрим ГН *G*. Заметим, что если граф *G* полный, то его единственная возможная раскраска – каждая вершина - получает свой собственный цвет. Если граф *G* не полный, то для его раскраски воспользуемся методом, предложенным Зыковым. Выберем две несмежные вершины *u* и *v*. Обозначим *(G;<u-v>)* граф, в котором вершины *u* и *v* соединены ребром. Обозначим *(G:<u=v>)* граф, в котором вершины *u* и *v* стянуты в одну.

Рассмотрим множество *M(G)* всех раскрасок ГН *G*. Множество *M(G)* представимо в виде объединения двух непересекающихся множеств:

$$M(G) = M((G;<u\text{-}v>)) \cup M((G:<u=v>)) \qquad (1)$$

Пользуясь соотношением (1), построим дерево T_c перебора всех раскрасок G. Каждая вершина дерева - это некоторый граф G', корень дерева - граф G. Если G' – не полный граф, то у него есть два сына – l и r, причем, $l=(G';<u-v>)$, $r=(G':<u=v>)$, где u и v некоторые вершины, не связанные ребром. Если G' – полный граф, то он является листом.

Теперь сопоставим каждой вершине t некоторое множество $X(t) \subset M(G)$, корню дерева сопоставим множество $M(G)$. Если у вершины t есть сыновья l и r, то им сопоставляются множества $M(l)$ и $M(r)$ соответственно.

Известно, что, используя раскраски графов, являющихся вершинами пути от листа к корню T_c, можно восстановить множество раскрасок графа G.

Пусть T_c' - некоторое поддерево дерева T_c всех вариантов раскраски, причем T_c' содержит корень T_c. Пусть величина $Built(t)$ – это количество левых сыновей на пути от корня дерева T_c' к вершине t. Пусть параметр D – это максимальное количество левых сыновей на любом пути от корня к листу в дереве T_c'. Обозначим через $T_c(D)$ семейство всех возможных поддеревьев T_c' дерева T_c, таких, что для любой их вершины n выполняется $Built(n) < D$.

Пусть $\Gamma(v)$ – это множество всех вершин смежных вершине v графа G.

Лемма 1.[1] Возьмем две произвольные вершины u и v графа G. Пусть выполняется $\Gamma(v) \subset \Gamma(u)$ или $\Gamma(u) \subset \Gamma(v)$. Тогда хроматическое число графа $(G:<u=v>)$ не больше, чем хроматическое число графа G.

Рассмотрим перебор множества раскрасок ГН G с помощью обхода поддеревьев из семейства $T(LeftChilds)$. Рассмотрим текущую при данном обходе вершину t дерева $T_c' \in T(LeftChilds)$. У вершины t есть поддеревья, нуждающиеся в обходе, если она является не полным графом G'. Будем всегда обходить правое поддерево вершины t. Затем выберем пару не соединенных ребром вершин u и v графа G'. Будем обходить левое поддерево t, если предикат $I(u,v)=(\Gamma(v) \subset \Gamma(u)==TRUE) \vee (\Gamma(u) \subset \Gamma(v)==TRUE)$ ложен.

[1] Номера лемм и теорем соответствуют номерам в тексте диссертации.

Согласно лемме 1 истинность этого предиката означает, что обход левого поддерева излишен, поскольку хроматическое число графа левого сына не меньше хроматического числа графа правого сына.

Параметрический алгоритм перебора раскрасок ГН *G* заключается в предложенном обходе дерева $T_c' \in T(LeftChilds)$.

Глава 3 посвящена уменьшению времен жизни виртуальных регистров, которое производится с целью уменьшения хроматического числа ГН программы. Предлагается древовидный алгоритм преобразования линейного участка, уменьшающий соответствующее хроматическое число.

Будем называть генератором виртуального регистра *v* такую инструкцию, которая присваивает значение *v*; использованием *v* такую инструкцию, потребляющую *v* в качестве операнда, что существует генератор *v*. Будем говорить, что инструкция *inst* потребляет значение *v*, присвоенное *v* генератором *gen*, если на пути в ГПУ между вершинами, соответствующими *gen* и *inst*, нет генераторов *v*.

Определим множество *Access(v)* следующим образом.

- Если вершина *x* в ГПУ является использованием *v*, то множество *Access(v)* содержит пару <*x,use*>. Символ *use* означает, что инструкция, соответствующая вершине *x,* рассматривается как использование.
- Если вершина *x* в ГПУ является генератором *v*, то множество *Access(v)* содержит пару <*x,def*>. Символ *def* означает, что инструкция, соответствующая вершине *x,* рассматривается как определение.
- Множество *Access(v)* не содержит никаких других пар.

Рассмотрим некоторое множество $U \subset Access(v)$. Будем говорить, что использование *x* виртуального регистра *v* относительно *U* является использованием типа *Common*, если выполнены следующие условия.

- Пара $<x, use> \notin U$.
- Существует путь *P* от некоторой вершины $y \in V(U)$ к использованию *x*.
- На пути *P* нет генераторов *v* и других вершин $z \in V(U)$.

Пусть *uses(U)* – это все вершины типа *Common* относительно данного множества *U*. Путь в ГПУ является путем типа *Common* относительно множества *U*, если существует $x \in V(U)$, являющаяся началом этого пути к использованию типа *Common*. Обозначим через *Starts(U)* множество всех вершин, в которых начинается хотя бы один путь типа *Common*.

Частичный сброс виртуального регистра *v* – это пара <*U,m*>, где *X* – подмножество множества *Access(v)*; *m* – некоторая ячейка памяти.

При проведении частичного сброса согласно <*U,m*> программа изменяется в четыре шага следующим образом.

1. Для каждой пары вида <*x, def*>, входящей в *U*, после вершины *x* вставляется инструкция, записывающая в ячейку памяти *m* значение виртуального регистра *v*, определенное этим генератором.
2. Ищется незанятый виртуальный регистр *v′*.
3. Для каждой пары вида <*x,use*> из *Starts(U)* перед вершиной *x* вставляется инструкция, считывающая значение из *m* в *v′*; в каждом использовании, принадлежащем множеству $uses(U) \cup Starts(U)$ каждый потребляемый (определяемый остается без изменений) виртуальный регистр *v* заменяется на *v′*.
4. Рассматриваем все пары вида <*x, use*>, входящие в *U*, такие, что первые элементы этих пар не входят в *Starts(U)*. Перед каждым таким использованием *x* вставляется инструкция, считывающая значение из *m* в *v′*, а в использовании *x* потребляемый (определяемый остается без изменений) виртуальный регистр *v* заменяется на *v′*.

Назовем множество $U \subset Access(v)$ корректным для сброса, если частичный сброс *v* согласно паре <*U, m*> приводит к эквивалентной процедуре.

Обозначим через *Gen(B,v)* множество генераторов, определяющих виртуальный регистр *v*, потребляемый использованием *B*.

Приведем наиболее важные теоремы, доказанные в третьей главе.

Теорема 1. Пусть имеется процедура *P* и ячейка памяти *m*. Рассмотрим некоторый виртуальный регистр *v* и некоторое множество $U \subset Access(v)$.

Пусть для каждой пары $<B, use>$ из множества U и для каждого генератора A из множества $Gen(B, v)$ выполняется хотя бы одно из двух условий:

- значение ячейки m совпадает со значением v после исполнения A;
- пара $<A, def>$ входит во множество U.

Тогда подмножество U является множеством, корректным для сброса.

Обозначим $Def(x)$ множество виртуальных регистров, которые определяются инструкцией, соответствующей вершине x. Обозначим $LastUse(x)$ множество виртуальных регистров, которые живы в вершине x, но не являются живыми ни в одной из вершин, в которые входят дуги из x. Для вершины $x \in V(G)$ определим величину регистрового давления $Pressure(x)$ следующим образом

$$Pressure(x)=\sum_{i\in\{1,\dots,n\}}|\{x\}\cap LifeTime(v_i)|-min(|Def(x)|,|LastUse(x)|) \qquad (2)$$

Здесь и далее будем считать, что N – это количество физических регистров, доступных в архитектуре. Обозначим через $V(G)$ множество вершин ГПУ G. Виртуальные регистры $v_1,v_2,\dots,v_n$ могут быть размещены в N физических регистрах, если для каждой вершины $x \in V(G)$ выполняется

$$Pressure(x) \le N \qquad (3)$$

Обозначим через $D(BB)$ такое разбиение линейного участка BB на линейные участки, что для любого линейного участка $Q \in D(BB)$ и любых двух вершин x и y линейного участка Q выполняется $Pressure(x)=Pressure(y)$.

Для любого линейного участка $Q \in D(BB)$ можно определить величину регистрового давления на этом участке как величину $Pressure(Q)=Pressure(v)$, где v – это любая вершина Q. Тогда перепишем формулу (3) следующим образом: $\forall Q \in D(BB), Pressure(Q) \le N$.

Пусть линейный участок Q' принадлежит разбиению $D(BB')$, где BB' – это линейный участок, полученный из линейного участка BB после

проведения частичного сброса. Тогда введем множество $D_{spill}(BB) = \{Q' \setminus S \mid Q' \in D(BB')\}$, $S = BB' \setminus BB$.

Теорема 3. Множество $D_{spill}(BB)$ – это одно из разбиений вида $D(BB)$.

Теорема 5. Если для любого $Q \in D_{spill}(BB)$ выполняется $Pressure(Q) \le N$, то для BB' все виртуальные регистры могут быть размещены в N физических.

Пусть $Num(v)$ - это номер вершины v линейного участка BB. Пусть линейный участок $Q \in D(BB)$, $first(Q)$ – это вершина Q с наименьшим номером, а $last(Q)$ – это вершина Q с наибольшим номером. Тогда Q можно сопоставить сегмент $I = [Num(first(Q)), Num(last(Q))]$.

Определим $Pressure([Num(first(Q)), Num(last(Q))]) = Pressure(Q)$. Тогда вместо множества $D(BB)$ можно оперировать множеством пар $DI(BB) = \{<I, Pressure(I)> \mid I = [Num(first(Q)), Num(last(Q))], Q \in D(BB)\}$.

Операции над числовыми интервалами программно реализуются значительно эффективнее, чем операции над множествами.

Рассмотрим виртуальный регистр v_i и некоторую ячейку памяти m_i, где $i \in \{1,2,\ldots,n\}$. Обозначим $AllSpill(v_i) = \{<E, m_i> \mid E \subseteq Access(v_i)$, причем E является множеством корректным для сброса$\}$. Пусть $S_i \in AllSpill(v_i)$, где $i \in \{1, 2, \ldots, n\}$, тогда обозначим $E(S_i)$ первый элемент пары S_i.

Будем называть множеством всех комбинаций частичных сбросов декартово произведение $AllSpill(v_1) \times AllSpill(v_2) \times \ldots \times AllSpill(v_n)$.

Проведением комбинации частичных сбросов относительно точки декартова произведения $(S_1, S_2, \ldots, S_n)$ называется применение к программе частичных сбросов согласно каждому S_i, где $i \in \{1, 2, \ldots, n\}$.

Пусть L – подграф ГПУ, порожденный линейным участком BB. Обозначим через $S(L)$ такое подмножество множества всех комбинаций частичных сбросов, что для любого его элемента $(S_1, S_2, \ldots, S_n)$ выполняется условие: для любого $i \in \{1,2,\ldots,n\}$ если пара $<A, def>$ принадлежит множеству $E(S_i)$, то минимум одна пара $<B, use>$, потребляющая значение, вычисленное генератором A, также принадлежит $E(S_i)$.

Рассмотрим некоторое подмножество X множества $S(L)$. Пусть существует $(S_1,S_2,...,S_n) \in X$, и выберем некоторый частичный сброс S_i, где $i \in \{1,2,...,n\}$. Выберем пару $<A, def> \in E(S_i)$.

Обозначим через $Spill(L, A) \subset X$ множество всех таких комбинаций частичных сбросов, что для любого $(S_1,...,S_i,...,S_n) \in Spill(L, A)$ выполняется $<A, def> \in E(S_i)$.

Обозначим через $NoSpill(L, A) \subset X$ множество всех таких комбинаций частичных сбросов, что для любой $(S_1,...,S_i,...,S_n) \in NoSpill(L, A)$ выполняется $<A, def> \notin E(S_i)$. Тогда множество X можно представить в виде двух непересекающихся подмножеств:

$$X = Spill(L, A) \cup NoSpill(L, A) \quad (4)$$

Пусть существует $(S_1,S_2,...,S_n) \in X$, тогда выберем некоторый частичный сброс S_i, где $i \in \{1,2,...,n\}$. Выберем пару $<B,use> \in E(S_i)$. Поскольку L порожден линейным участком, можно выбрать единственный генератор A такой, что B является его использованием.

Обозначим через $Reload(L,A,B) \subset X$ множество таких комбинаций частичных сбросов, что для любой $(S_1,...,S_i,...,S_n) \in Reload(L, A)$ выполняется $<B, use> \in E(S_i)$.

Обозначим через $NoReload(L, A, B) \subset X$ множество таких комбинаций частичных сбросов, что для любой $(S_1,...,S_i,...,S_n) \in NoReload(L, A, B)$ выполняется $<B, use> \notin E(S_i)$. Тогда множество X можно представить в виде двух непересекающихся подмножеств:

$$X = Reload(L, A, B) \cup NoReload(L, A, B) \quad (5)$$

Пользуясь соотношениями (4) и (5), можно построить дерево T_s, описывающее все возможные комбинации частичных сбросов для одного линейного участка. Корень этого дерева – это множество *S(L)*.

Каждая вершина *t* дерева T_s – это множество $X \subset S(L)$.

Если *X* разбиваемо согласно формуле (4), причем $Spill(L,A) \neq \varnothing$ и $NoSpill(L, A) \neq \varnothing$, то у вершины *t* есть два сына *y* и *z*, вершине *y* ставится в соответствие множество *Spill(L, A)*, вершине *z* ставится в соответствие множество *NoSpill(L, A)*.

Пусть у вершины *t* нет сыновей *y* и *z*. Тогда, если *X* разбиваемо согласно формуле (5), причем $Reload(L, A, B) \neq \varnothing$ и $NoReload(L, A, B) \neq \varnothing$, у вершины *t* есть два сына *w* и *v*, то вершине *w* ставится в соответствие множество *Reload(L, A, B)*, вершине v ставится в соответствие множество *NoReload(L, A, B)*.

В противном случае, сыновей у вершины *t* нет, и мощность $|t|=1$.

Пусть *Leaves* – это множество листьев дерева T_s достижимых из вершины *t*. Путь от корня T_s к его вершине *t* определяет комбинацию частичных сбросов *S(t)*, описывающую принятые на этом пути решения о том, какие частичные сбросы войдут в комбинации частичных сбросов принадлежащие *Leaves*. Комбинация *S(t)* определяет линейный участок *BB´(t)*, получаемый после ее применения к *BB*. Оценив эффект от проведения комбинации *S(t)* над *BB*, можно оценить эффект от проведения комбинаций из *Leaves* на *BB*.

Множеству листьев T_s соответствует множество линейных участков, получаемых после применения всех возможных комбинаций из *S(L)*.

Обозначим через *BuiltNoSpill(t)* множество вершин дерева T_s такое, что:

- $\forall z \in BuiltNoSpill(t)$ выполняется $z \in P(t)$;
- $\forall z \in BuiltNoSpill(t)$ у родителя *z* есть другие сыновья кроме *z*;
- $\forall z \in BuiltNoSpill(t)$ соответствует множество типа *NoSpill(L, A)*.

Обозначим через *BuiltNoReload(t)* множество вершин дерева T_s, такое, что:

- $\forall z \in BuiltNoReload(t)$ выполняется $z \in P(t)$;

- $\forall z \in BuiltNoReload(t)$ у родителя z есть другие сыновья кроме z;
- $\forall z \in BuiltNoReload(t)$ соответствует множество типа $NoReload(L, A)$.

Пусть $D1$, $D2$ – это некоторые неотрицательные целые числа. Семейство поддеревьев $T_s(D1,D2)$ – это семейство, которому принадлежат все поддеревья T_s такие, что для любой их вершины t величины $|BuiltNoSpill(t)|$ и $|BuiltNoReload(t)|$ не превышают значений $D1$ и $D2$ соответственно.

Осуществим перебор множества комбинаций частичных сбросов с помощью обхода дерева $T_s' \in T_s(D1,D2)$. При обходе дерева будем использовать метод ветвей и границ для отсечения веток обхода, которые заведомо не находят наилучшей комбинации. Для этого введем переменную *Record* и положим ее равной бесконечности. В дальнейшем будем присваивать переменной *Record* значение $ETM(BB'(t))$ для некоторой вершины t дерева T_s, если выполняются следующие два условия.

- Текущее значение *Record* больше величины $ETM(BB'(t))$.
- Для каждого $<I, Pressure(I)> \in DI(BB)$, $Pressure(I)<N$.

Заметим, что на пути от корня дерева T_s к его листу функция $ETM(BB'(t))$ монотонно не убывает. Поэтому метод ветвей и границ, реализуемый с помощью переменной *Record*, корректен.

Рассмотрим текущую при данном обходе вершину t дерева T_s'. Если у вершины t есть сын z, то он обходится, если предикат $Record>ETM(BB'(z))$ истинен. Пара $<I(child),Pressure(I(child))>$ строится по паре $<I(t), Pressure(I(t))>$ с помощью операций над числовыми интервалами.

Параметрический алгоритм перебора комбинаций частичных сбросов заключается в предложенном обходе дерева $T_s' \in T(D1, D2)$.

В главе 4 рассматривается задача нахождения оптимальной комбинации преобразований для минимизации времени исполнения программы. В качестве инструмента для выбора наиболее хорошей комбинации оптимизаций предлагается бинарное дерево перебора таких комбинаций.

Предлагается алгоритм, который для циклов программы, являющихся линейными участками, и конкретного набора преобразований выбирает наилучшую комбинацию этих преобразований.

Будем называть элементарным преобразованием такое, которое является не разложимым на более мелкие. Будем говорить, что элементарное преобразование E' зависит от преобразования E'', если E' не может быть проведено без предшествующего проведения E''. Будем называть элементарное преобразование E' несовместимым с преобразованием E'', если E' не может быть проведено, если проведено E'', и E'' не может быть проведено, если проведено E'.

Для нахождения оптимальной комбинации преобразований введем смешанный (с ориентированными дугами и неориентированными ребрами) граф зависимостей между возможными преобразованиями (ГЗВП) [2].

Вершины ГЗВП – это виртуальные регистры, константы и элементарные преобразования. Ребро соединяет пару вершин, соответствующих несовместимым преобразованиям. Пару вершин u и v соединяет дуга (u, v), если выполняется одно из двух условий.

- Вершина u – это виртуальный регистр, вершина v – это преобразование над инструкциями, использующими или определяющими виртуальный регистр, соответствующий u.
- Вершина u – это преобразование E_u, вершина v – это преобразование E_v, преобразование E_v зависит от преобразования E_u.

Пусть элементарные преобразования $E_1, E_2, \ldots, E_n$ принадлежат множеству $Trans(F)$, эквивалентных элементарных преобразований процедуры F. Комбинацией элементарных преобразований будем называть упорядоченную последовательность $<E_l, E_k, \ldots, E_m>$, где $1 \le l < k < m \le n$.

Обозначим через $AllTransComb(F)$ все комбинации элементарных преобразований из множества $Trans(F)$ такие, что они приводят к эквивалентной процедуре F'. Рассмотрим некоторое подмножество

$X \subset AllTransComb(F)$. Пусть существует комбинация $Comb \in X$, выберем некоторое элементарное преобразование E_i, входящее в $Comb$.

Обозначим через $Present(F,E_i) \subset X$ множество всех комбинаций элементарных преобразований, содержащих E_i, и не содержащих элементарных преобразований, несоместимых с E_i. Обозначим через $NotPresent(F,E_i) \subset X$ множество всех комбинаций элементарных преобразований, не содержащих E_i и не содержащих элементарных преобразований, зависящих от E_i.

Множество X можно представить в виде разбиения

$$X = Present(F, E_i) \cup NotPresent(F, E_i) \quad (6)$$

Пользуясь соотношением (6), можно построить дерево T_a, перебирающее все возможные комбинации элементарных преобразований для процедуры F. Корень этого дерева – это $AllTransComb(F)$. Каждая вершина t дерева T_a – это некоторое подмножество $X \subset AllTransComb(F)$.

Если X разбиваемо согласно формуле (6) относительно некоторого E_i из комбинации $Comb$, где $Comb \in X$, то у вершины t существует два сына $y=Present(F, E_i)$ и $z=NotPresent(F, E_i)$.

Путь от корня T_a к его вершине t определяет комбинацию элементарных преобразований $Comb(t)$, состоящую из преобразований разбивающих предшественника каждой вершины на этом пути согласно формуле (6). Комбинации $Comb(t)$ соответствует процедура $F'(t)$, которая получается из процедуры F последовательным применением преобразований $E_i \in Comb(t)$.
Заметим, что множество листьев дерева Та – это множество $AllTransComb(F)$.

Пусть D – некоторое неотрицательное целое число. Рассмотрим семейство поддеревьев $T_a(D)$ дерева T_a. Обозначим через $NumOfNotPresent(T_a')$ количество вершин типа $NotPresent(L, E_i)$ в поддереве $T_a' \in T_a(D)$. Семейству

поддеревьев $T_a(D)$ принадлежат все поддеревья дерева T_a, для которых выполняется $NumOfNotPresent(T_a') < D$.

Пусть *Loop* – это цикл исходной программы, состоящий из одного линейного участка *BB*. Вершине *t* соответствует цикл $Loop'(t)$, получаемый после проведения комбинации *Comb(t)* над циклом *Loop*.

Рассмотрим перебор множества комбинаций элементарных преобразований с помощью построения дерева $T_a' \in T_a(D)$. Обозначим через $Loop'(t+E_i)$ цикл, получаемый после проведения преобразования E_i над циклом $Loop'(t)$, где *t* – это вершина дерева T_a'. Обозначим через $\{E_{jj}\}$ множество допустимых к выбору преобразований для цикла $Loop'(t)$. Положим $MaxPressure(BB)=max_{v \in BB}(Pressure(v))$. Рассмотрим возможную стратегию выбора сыновей для вершины *t*.

Пусть верно, что $\exists E_i \in \{E_{jj}\}$ такое, что $MaxPressure(Loop'(t+E_i)) \leq MaxPressure(Loop'(t))$. Тогда левый сын вершины *t* – это множество $Present(L, E_i)$. Если $NumOfNotPresent(T_a') < D$, то правый сын вершины *t* – это множество $NotPresent(L, E_i)$.

В противном случае, левый сын вершины *t* – это множество $Present(L, E_k)$, такое, что $\forall E_l \in \{E_{jj}\}$ выполняется $MaxPressure(Loop'(t+E_k)) \leq MaxPressure(Loop'(t+E_l))$. Если $NumOfNotPresent(T_a') < D$, то правый сын вершины *t* – это множество $NotPresent(L, E_k)$.

Для выбора наилучшей комбинации преобразований необходимо вычислять $ETM(Loop'(t))$ для каждой вершины *t* дерева T_a', потому, что множество листьев дерева T_a' не описывает все возможные комбинации преобразований. Эффективная оценка времени исполнения цикла $Loop'(t)$, соответствующего вершине *t*, возможна, если *Trans(Loop)* содержит: вынос инвариантного выражения из цикла, подстановку вперед, понижение силы операций и удаление общих подвыражений.

В главе 5 проводится анализ распараллеливаемости алгоритмов построения деревьев из семейств $T_c(D)$, $T_s(D1,D2)$ и $T_a(D)$. Показывается, что параллельные варианты алгоритмов построения требуют разное количество

пересылок данных и синхронизаций. С другой стороны, устанавливается, что число таких пересылок и синхронизаций невелико, поэтому предложенные методы целесообразно распараллеливать. Таким образом, время компиляции программы может быть существенно сокращено путем распараллеливания предложенных алгоритмов.

В заключении изложены основные результаты работы и выводы по диссертации в целом.

ОСНОВНЫЕ РЕЗУЛЬТАТЫ

Проведены комплексные теоретические исследования и разработаны методы для проведения оптимизации программ написанных на таких современных языках программирования как C, C++, Fortran, Java и C#.

1. Построена модель времени вычислений, позволяющая на этапе компиляции программы оценивать время ее исполнения.
2. Разработан древовидный параметрический алгоритм раскраски графа несовместимости процедуры для оптимизации быстродействия кода в рамках лимита времени компиляции.
3. Разработан древовидный параметрический алгоритм выбора оптимального множества сбросов для понижения хроматического числа графа несовместимости линейного участка для оптимизации быстродействия кода в рамках лимита времени компиляции.
4. Разработан алгоритм выбора оптимальной комбинации преобразований программы, позволяющих с помощью выбора значения параметра оптимизировать быстродействие кода, не превышая лимита времени компиляции.
5. Проведен анализ предложенных алгоритмов и показана их распараллеливаемость, позволяющая значительно сократить время компиляции программ.

Личный вклад соискателя заключается в постановке задач, обсуждении методов их решения, разработке алгоритмов, реализующих выбранные методы, и теоретическом обосновании оптимальности предложенных алгоритмов. Все выносимые на защиту результаты принадлежат лично автору.

ПУБЛИКАЦИИ ПО ТЕМЕ ДИССЕРТАЦИИ

1. Макошенко Д.В. Математическая модель времени вычислений для оптимизации программ // Известия Вузов / Издательство ЮФУ. — Ростов-на-Дону, 2009. — №2. — С. 10-13.
2. Макошенко Д.В. Назначение переменных на регистры с помощью древовидного параметрического алгоритма раскраски графа // Информационные Технологии / Москва, 2010. — № 6 — С. 41 – 46.
3. Штейнберг Б.Я., Макошенко Д.В., Черданцев Д.Н., Шульженко А.М. Внутреннее представление в открытой распараллеливающей системе // Искусственный интеллект / Институт проблем искусственного интеллекта НАНУ. — Украина, Донецк, 2003. — № 3. — С. 89-96.
4. Макошенко Д.В. Нахождение оптимальной комбинации преобразований для минимизации времени исполнения программы // Труды научной школы И.Б. Симоненко / Издательство ЮФУ. — Ростов-на-Дону, 2010. — С. 163-167.
5. Steinberg B., Abramov A., Alymova E., Baglij A., Guda S., Demin S., Dubrov D., Ivchenko A., Kravchenko E., Makoshenko D., Molotnikov Z., Morilev R., Nis Z. , Petrenko V., Povazhnij A., Poluyan S., Skiba I., Suhoverkhov S., Shapovalov V., Steinberg O., Steinberg R. Dialogue-Based Optimizing Parallelizing Tool and C2HDL Converter // Proceedings of IEEE EAST-WEST DESIGN & TEST SYMPOSIUM 2009 / MOSCOW, 2009. — P. 216–218.
6. Макошенко Д.В. Назначение переменных на регистры с помощью новых алгоритмов раскраски графа // Труды 52-й научной конференции МФТИ «Современные проблемы фундаментальных и прикладных наук». — Долгопрудный, 2009. — Часть I. Радиотехника и кибернетика. Том 1. — С. 96-98.
7. Штейнберг Б.Я., Макошенко Д.В., Черданцев Д.Н., Шульженко А.М. Внутреннее представление в открытой распараллеливающей системе // Интеллектуальные и многопроцессорные системы ИМС-2003, материалы Международной научно-технической конференции. — Дивноморское, 2003. — Т. 2. — С. 47-50.

Макошенко Д.В.

АНАЛИТИЧЕСКОЕ ПРЕДСКАЗАНИЕ ВРЕМЕНИ ИСПОЛНЕНИЯ ПРОГРАММ И ОСНОВАННЫЕ НА НЕМ МЕТОДЫ ОПТИМИЗАЦИИ

Автореферат

Подписано в печать Объем 1,1 уч.-изд. л.

Формат бумаги 60 × 90 1/16 Тираж 100 экз.

Отпечатано в ЗАО РИЦ «Прайс-курьер»

630090, г. Новосибирск, пр. ак. Лаврентьева, 6, тел. 330-72-02

Заказ №135