УДК 004.4’416

На правах рукописи

Филиппов Александр Николаевич

МЕТОДЫ УДАЛЕНИЯ ИЗБЫТОЧНОСТЕЙ НА ЭТАПЕ КОМПИЛЯЦИИ ПРОГРАММ

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

АВТОРЕФЕРАТ

диссертации на соискание ученой степени

кандидата технических наук

Москва-2009

Работа выполнена в *ЗАО «МЦСТ»* и *ОАО «ИНЭУМ им. И.С.Брука»*

Научный руководитель: кандидат технических наук,
старший научный сотрудник
Волконский Владимир Юрьевич

Официальные оппоненты: доктор физико-математических наук
Петренко Александр Константинович
кандидат технических наук
Серебряный Константин Сергеевич

Ведущая организация: ОАО «Институт точной механики и вычислительной техники им. С.А. Лебедева»

Защита состоится «___» _____2009г. в _____ часов на заседании диссертационного совета Д.409.009.01 при **ОАО «Институт электронных управляющих машин им. И.С.Брука»**, расположенном по адресу : 119334, Москва, ул. Вавилова, д.24

С диссертацией можно ознакомиться в библиотеке **ОАО «ИНЭУМ им. И.С. Брука»**

Автореферат разослан «___» _____2009 г.

Ученый секретарь
диссертационного совета,
кандидат технических наук, профессор Красовский В.Е.

Общая характеристика работы

Актуальность работы. Одной из причин неполного использования производительности вычислительных систем является наличие в программах избыточных вычислений, называемых в дальнейшем просто «избыточностями». Вне зависимости от того, присутствуют ли избыточности в исходном коде программы или возникают в процессе перевода в код целевой платформы, их устранение является одной из наиболее приоритетных задач оптимизирующей компиляции.

Из всего многообразия избыточностей, характерных для той или иной архитектуры процессоров, выделяют отдельный класс, в который входят избыточности, *общие для всех типов архитектур* (архитектурно-независимые избыточности). Оптимизации, направленные на удаление архитектурно-независимых избыточностей, представлены в литературе. Практически все они базируются на использовании результатов предварительного анализа, называемого нумерацией значений (*Value Numbering*). Стоит отметить, что известные методы удаления архитектурно-независимых избыточностей в ряде случаев недостаточно эффективны, и причиной тому являются недостаточно полные результаты анализа. Поэтому *развитие методов нумерации значений является одним из главных резервов, позволяющих выявлять архитектурно-независимые избыточности и впоследствии удалять их.* Кроме того, совместно с развитием самой техники анализа, представляется актуальным *развитие методов оптимизации программ с использованием результатов нумерации значений*.

Проблема наличия избыточностей приобретает особенную актуальность применительно к архитектурам, рассчитанным на достижение высших показателей производительности. Их аппаратные особенности приводят к появлению специфических видов избыточностей, которые не могут быть удалены общими методами. В этом ряду следует выделить архитектуры, использующие статический подход к распараллеливанию на уровне

инструкций (EPIC - Explicitly Parallel Instruction Computing), к которым относится архитектура отечественных микропроцессоров серии «Эльбрус», предназначенных для разработки крупномасштабных иформационно-вычислительных систем стратегического значения. Ключевой задачей компилятора в данном случае является нахождение максимального параллелизма программы на уровне операций, поэтому основным видом избыточностей по праву считаются вычисления, *препятствующие эффективному статическому распараллеливанию*. Существенно также, что в этих архитектурах используется аппаратная поддержка *предикатных вычислений* и *конвейеризации циклов*, которая вызывает появление дополнительных избыточностей. Эти факторы не позволяют полноценно использовать преимущества EPIC-архитектуры, поэтому развитие методов их устранения представляется актуальным.

В качестве еще одного класса, выделяют межпроцедурные избыточности, которые возникают вследствие наличия в программах вызовов функций. Основным средством *устранения межпроцедурных избыточностей* традиционно считается подстановка тела вызываемой функции в точку вызова (инлайн-подстановка). После нее компилятор получает возможность избавиться от большинства межпроцедурных избыточностей, а в случае, если целевая архитектура имеет тип EPIC, еще и планировать вызывающую и вызываемую функции вместе, повышая при этом общий параллелизм исполнения программы. Для принятия решения о возможности и целесообразности конкретной подстановки компиляторы производят предварительные оценки, которые зачастую являются слишком грубыми, что приводит к отказу от применения преобразования и влечет потерю производительности. Исходя из этого, представляется актуальным *расширение области применимости и повышение эффективности инлайн-подстановок путем развития сопутствующих методов статического и динамического анализа программы.*

Цель исследования. Конечной целью исследования является увеличение эффективности работы оптимизирующего компилятора путем разработки новых и развития имеющихся методов анализа и оптимизации программ. В соответствии с этой целью, были поставлены следующие задачи:

- исследовать существующие и разработать новые методы удаления архитектурно-независимых избыточностей;
- повысить эффективность исследованных и предложенных оптимизаций путем разработки усиленного алгоритма нумерации значений; при этом необходимо использовать положительные результаты уже существующих подходов, в то же время добившись углубления анализа за счет снятия основных присущих им ограничений;
- разработать методы удаления избыточностей для архитектур с явно выраженным параллелизмом на уровне инструкций;
- разработать методы анализа и оптимизации, направленные на удаление межпроцедурных избыточностей;
- реализовать предложенные алгоритмы и методы в составе оптимизирующего компилятора для архитектуры «Эльбрус».

Научная новизна Решение поставленных в диссертационной работе задач определяет научную новизну исследования, которую составляют:

- усовершенствованный метод нумерации значений, позволяющий выявлять эквивалентности между операциями обращения к памяти, операциями с разными именами, а также включающий в себя сбор информации о константных свойствах переменных, и его использование в оптимизациях;
- методы удаления избыточных предикатных вычислений и упрощения предикатных выражений;
- метод многоцелевой балансировки деревьев ассоциативных выражений, позволяющий приводить длинные последовательные цепочки операций к параллельному виду;

- методы повышения эффективности аппаратной конвейеризации циклов путем применения балансировки и комбинирования операций;
- методы расширения области применимости inline-подстановок на основе результатов статического анализа и профильной информации.

Практическая ценность работы состоит в создании новых и усовершенствовании известных методов оптимизации программ, а также в существенном повышении информативности аналитических компонентов компилятора. Все представленные в диссертационной работе методы реализованы в составе оптимизирующего компилятора с языков высокого уровня Си, Си++, Фортран для микропроцессора «Эльбрус». Кроме того, часть из них, не зависящая от целевой платформы, была реализована в составе оптимизирующего компилятора для микропроцессора Sparc. Эффективность предложенных методов подтверждена замерами производительности на пакетах задач SPEC CPU95 и SPEC CPU2000 и на функциях, входящих в состав высокопроизводительной библиотеки векторных вычислений для архитектуры «Эльбрус».

Апробация.

Результаты, полученные в работе, изложены в ряде печатных публикаций, докладывались на научных конференциях и семинарах, в частности :

- на XXXII Международной молодежной научной конференции «Гагаринские чтения» (Москва, МАТИ, 2006 г.);
- на XXIII научно-технической конференции «Направления развития и применения перспективных вычислительных средств и новых информационных технологий в ВВТ РКО» (Москва, в/ч 03425, 2007г.);
- на XXXIII Международной молодежной научной конференции «Гагаринские чтения» (Москва, МАТИ, 2007 г.);
- на 51-й научной конференции МФТИ (2008г.);

- на XXXV Международной молодежной научной конференции «Гагаринские чтения» (Москва, МАТИ, 2009 г.);
- на семинарах секции программного обеспечения ЗАО «МЦСТ» и ОАО «ИНЭУМ».

Публикации.

По теме диссертации опубликовано 6 печатных работ

Структура и объем работы.

Диссертация состоит из введения, четырех глав и заключения. Список литературы составляет 67 наименований. Объем диссертации составляет 161 страницу текста. Диссертация содержит 57 рисунков и 3 таблицы.

Содержание работы.

Введение обосновывает актуальность диссертационной работы, формулирует ее цель, аргументирует факторы научной новизны исследований и практической значимости полученных результатов.

Во вступительной части **Главы 1** дана классификация избыточностей, базовые элементы которой приведены на рис.1

Далее, исследуется процесс выявления архитектурно-независимых избыточностей при статическом анализе программы. Наиболее распространенным из используемых при этом методов является *нумерация значений*, суть которой заключается в разбиении множества операций промежуточного представления на классы эквивалентности (конгруэнтности). Две операции признаются эквивалентными только тогда, когда компилятору удалось доказать, что они всегда вырабатывают одинаковый результат. В главе проводится сравнение описанных в литературе методов нумерации значений, характеризуются их достоинства и недостатки, определяются направления развития.

Рис. 1. Классификация избыточностей

Подробное исследование трех известных методов нумерации значений - пессимистического (Hash Based Value Numbering), оптимистического (Value Partitioning) и универсального (SCC Based Value Numbering) – в части устранения внутрипроцедурных избыточностей приводит к выводу, что, несмотря на многообразие возможностей, они имеют ряд недостатков.

Во-первых, применение любого из методов к **операциям обращения к памяти** приводит к некорректным результатам.

Во-вторых, во всех методах необходимым условием эквивалентности операций полагается **совпадение их имен**. В реальных же программах возникает достаточно случаев, когда операции, имеющие разные имена, вырабатывают одинаковый результат.

В-третьих, известные методы недостаточно полно отражают **константные свойства операций**. В них константные классы конгруэнтности присваиваются только операциям, имеющим константные аргументы. На самом же деле, существует множество примеров, в которых операция, полу-

чающая на вход переменные аргументы, гарантированно вырабатывает константный результат. Кроме того, встречаются случаи, когда результат операции, не являясь полностью константным, может принимать лишь определенный набор значений и обладание этим фактом может позволить упростить исходную программу.

В **Главе 2** рассматриваются способы устранения описанных выше ограничений. Помимо этого, аналитически и экспериментально исследуется проблема использования результатов нумерации значений в платформонезависимых оптимизациях.

Расширение применимости алгоритмов нумерации значений к операциям работы с памятью основывается на понятии *объекта* – непрерывной последовательности N байт в памяти. На фазе построения промежуточного представления каждой операции чтения/записи сопоставляется объект, с которым она работает. Тип объекта (скаляр, массив, структура и т.п.) однозначно устанавливается по типу исходной переменной при конвертации программы в промежуточное представление. В работе сформулированы и доказаны способы вычисления классов конгруэнтности для операций доступа к памяти. Применение того или иного способа зависит от типа объекта, с которым работает конкретная операция.

Остается открытым вопрос проведения нумерации значений для операций, которым не удалось сопоставить объект. В диссертационной работе сформулирован и доказан признак конгруэнтности для таких операций, базирующийся на понятии *доминирующего фронта записей* (D_a). Причем, если компилятору доступны результаты анализа конфликтов операций доступа в память, то данный признак может быть усилен. Усиленный признак конгруэнтности основан на понятии *доминирующего фронта конфликтующих записей* (Dc_a). В работе показано, что он действительно является более точным, причем оба признака будут эквивалентны при наличии консервативных результатов анализа конфликтов (когда каждая операция

конфликтует с каждой). Кроме того, в работе предложен ряд алгоритмов построения множеств D_a и Dc_a. Алгоритмическая сложность каждого из них определяется, исходя из свойств графа управления процедуры, а также общего количества операций чтения, которым не удалось сопоставить объект. В каждом конкретном случае для построения множеств D_a и Dc_a компилятор применяет алгоритм с наименьшей сложностью.

Эффективность предложенного метода нумерации значений для операций обращения в память экспериментально проверена на задачах пакета SPEC CPU2000. В ходе экспериментов для каждого теста подсчитывалась процентная доля «потенциально избыточных»[1] операций чтения из памяти. Она в среднем составила 33% от общего количества чтений, а на некоторых тестах достигала 58%.

Следующим результатом в части сокращения ограничений, свойственных методам устранения внутрипроцедурных избыточностей, является алгоритм выявления эквивалентностей между арифметическими операциями, имеющими разные имена. Он базируется на использовании техники *PS-форм* (форма сумм произведений, Product Sums Form). PS-форма представляет собой полином вида $c_0 + c_1x_{11}x_{12}...x_{1k} + ... + c_nx_{n1}x_{n2}...x_{nk}$, где $c_0, ..., c_n$ - некоторые константы, а x_{ij} – переменные (в нашем случае, классы конгруэнтности). В качестве дополнительного признака эквивалентности операций предложенный алгоритм использует совпадение соответствующих им PS-форм. Причиной выбора PS-форм в качестве символьных выражений для анализа является обилие арифметических операций, используемых, главным образом, для вычисления индексов многомерных массивов, встречающееся в кодах реальных программ.

И наконец, еще одна возможность сокращения ограничений связана с тем фактом, что в реальных программах результат операции, не являясь константой, тем не менее, может иметь определенный константные свойства, например, принимать определенный набор значений или находиться

[1] Операция, результат которой также вырабатывается некоторой другой операцией программы

в заданном диапазоне. В работе предложен интегрированный с Value Numbering способ описания константных свойств выражений, при котором значение выражения рассматривается не как число, а как *вектор битов*. Для этого каждому классу конгруэнтности сопоставляется два битовых вектора: вектор известных битов (*known_bits*) и вектор значений известных битов (*bits_val*). Равенство единице бита с номером n вектора *known_bits* означает, что значение этого бита (в результатах всех операций данного класса конгруэнтности) известно. Величина этого известного значения отражена в бите с номером n вектора *bits_val*.

Вычисление векторов *known_bits* и *bits_val* для каждой операции промежуточного представления проводится по-своему, исходя из соответствующих векторов ее аргументов. Алгоритмы вычисления указанных векторов для конкретных арифметических и логических операций представлены в приложении А диссертации.

В ряде случаев с помощью анализа векторов известных битов можно доказать, что некоторые классы конгруэнтности являются константными. Указанное свойство позволяет проводить более эффективную свертку (вычисление) константных выражений в процессе нумерации значений.

Результаты нумерации значений традиционно используются компилятором в оптимизациях, направленных на удаление архитектурно-независимых избыточностей. Наиболее известными из них являются:

- глобальный сбор общих подвыражений (GCSE, Global Common Subexpressions Elimination),
- глобальное распространение копий (GCP, Global Copy Propagation)
- удаление частичных избыточностей (PRE, Partial Redundancies Elimination или VDCM, Value Driven Code Motion)

Эти оптимизации являются внутрипроцедурными, архитектурно-независимыми и входят в состав большинства современных оптимизирующих компиляторов. Их эффективность напрямую зависит от того, на-

сколько полными будут результаты нумерации значений. В работе проанализированы наиболее удобные для реализации алгоритмы указанных преобразований. Кроме того, предложен отдельный способ оптимизации программ, основанный на анализе значащих битов (bitopt).

Усиленный метод нумерации значений, включающий в себя нумерацию значений для операций обращения в память, использование техники PS-форм, а также применение битового подхода к описанию константных свойств операций, был реализован в составе оптимизирующего компилятора для архитектуры «Эльбрус». В составе компилятора были также реализованы оптимизации GCP, GCSE, VDCM и bitopt.

С практической точки зрения, использование результатов усиленного метода нумерации значений позволило повысить эффективность оптимизаций GCP, GCSE и VDCM и достичь дополнительного прироста производительности, по сравнению с базовым методом. Для пакета SPEC CPU2000 среднее ускорение составило 8,5%; на отдельных тестах ускорение достигло 28,8%. Что касается пакета SPEC CPU95, то для него среднее ускорение составило 3,15%, а на отдельных тестах достигло 6%.

Исследовалось применение оптимизации bitopt, использующей исключительно результаты усиленной нумерации значений, после проведения классических оптимизаций. Для тестов пакета SPEC CINT2000 оно дало дополнительное ускорение, которое в среднем составило 1,3%, а на отдельных тестах достигло 2,7%. На тестах пакета SPEC CPU95 среднее ускорение составило 3%, а максимальное – 6,9%.

Методы, предложенные в данной главе, не зависят от конкретной архитектуры и могут применяться в оптимизирующих компиляторах для различных микропроцессоров. В частности, усиленный метод нумерации значений и использующие его преобразования были реализованы в составе компилятора для суперскалярных микропроцессоров Sparc. На тестах пакета SPEC CPU95 использование усиленного метода нумерации значений позволило достичь ускорения до 4,8%, а использование оптимизации bitopt

– до 2,5%. Проведенные замеры производительность доказывают универсальность предложенных методов, обеспечивающих более эффективное выявление и удаление архитектурно-независимых избыточностей.

В **Главе 3** рассматривается задача удаления избыточностей, специфичных для архитектур класса EPIC.

С целью получения эффективного кода в их аппаратуре реализуется поддержка *предикатных вычислений,* что, как правило, отражено в свойствах низкоуровневого промежуточного представления, используемого компилятором. Компилятор вначале использует непредикатную (безусловную) форму представления, и лишь после ряда оптимизаций преобразует представление в предикатную форму. Причем, на безусловном коде обычно выполняются платформо-независимые оптимизации, в то время как на предикатном коде - оптимизации, специфичные для EPIC архитектур.

Процесс перевода безусловного кода в предикатный традиционно называется *if-conversion*. Для вычисления итоговых предикатов операций в EPIC архитектурах поддерживаются логические операции над предикатами, которые отражены и в соответствующем промежуточном представлении. В случае сложного управления внутри программы, вычисления предикатов могут быть достаточно громоздкими и, как следствие, негативно влиять на итоговое время исполнения.

Решение этой проблемы потребовало разработки специализированных методов *упрощения предикатных выражений* и *удаления избыточных предикатов*. Метод упрощения сложных предикатных выражений основывается на построении совершенных дизъюнктивно нормальных форм (СДНФ) для предикатов и последующем их упрощении с помощью правил булевой и укороченной (short circuit) логики, а также на основе анализа потока данных программы. Кроме того, совпадение СДНФ является критерием для установления эквивалентности предикатов и последующего удаления избыточных (дублирующих) предикатных операций. Использование

предложенных методов в оптимизирующем компиляторе позволило достичь ускорения до 7% на тестах пакета SPEC CPU2000 и до 7,3% на тестах пакета SPEC CPU95.

Важнейшей задачей компилятора для EPIC-архитектур является нахождение максимального параллелизма программ на уровне операций. Одной из причин, негативно влияющих на реализацию параллельности, является наличие в программах *длинных цепочек операций, последовательно связанных друг с другом зависимостями*. В этом случае все операции цепочки должны исполняться последовательно, и преимущества параллельной архитектуры не используются. Естественно, подобная ситуация приводит к замедлению работы программы.

В качестве решения данной проблемы для ассоциативных операций предложен метод *классической балансировки*, основной задачей которого является преобразование последовательных цепочек выражений в параллельный вид. Помимо решения основной задачи, этот метод позволяет компилятору сворачивать некоторые константные подвыражения, а также уменьшать количество используемых регистров.

Еще одной важной особенностью EPIC-архитектур является аппаратная поддержка конвейеризации циклов, благодаря которой становится возможным применение *технологии планирования с перекрытием итераций*. Она была отработана в компиляторных проектах для архитектур «Эльбрус» и Itanium и зарекомендовала себя, как одно из наиболее эффективных программно-аппаратных средств повышения производительности. В этом контексте в число избыточных операций включаются и операции, ухудшающие эффективность данной технологии.

Время исполнения аппаратно-конвейеризованного цикла монотонно зависит от величины $\max(n_r, n_a)$, где n_r – длина максимальной рекуррентности, а n_a - минимальное число инструкций, необходимое для упаковки опе-

раций цикла. Соответственно, суть удаления избыточностей для таких циклов сводится к минимизации этих двух характеристик.

В качестве способа минимизации n_r в диссертационной работе предложен *метод специализированной балансировки,* предполагающий вынос «лишних» операций из рекуррентных цепочек. В рамках оптимизирующего компилятора для архитектуры «Эльбрус» классическая и специализированная балансировка реализованы в виде единой *многоцелевой балансировки*. Она включает в себя различные эвристические условия, исходя из которых принимается решение о применении классического или специализированного алгоритма преобразования. Использование многоцелевой балансировки в компиляторе позволило достичь ускорения до 15,5% и 19,5% на тестах пакетов SPEC CPU2000 и SPEC CPU95 соответственно.

В качестве способа минимизации n_a, в диссертационной работе предложен метод построения *комбинированных операций*. Комбинированная операция сочетает в себе последовательное исполнение двух обычных операций, причем результат первой из них используется в качестве аргумента второй. Сокращение числа инструкций цикла достигается за счет того, что комбинированная операция исполняется на одном арифметико-логическом устройстве, в то время как составляющие ее операции заняли бы два. В ряде случаев построение комбинированных операций может повлечь увеличение длины максимальной рекуррентности и, как следствие, общего время исполнения цикла. Поэтому перед комбинированием каждой пары операций проводится эвристическая оценка эффективности преобразования, основанная на анализе длин максимальных рекуррентностей.

На задачах пакета SPEC CPU95 комбинирование операций в контексте аппаратно-конвейеризованных циклов привело к ускорению до 6%. Кроме того, данная оптимизация позволила достичь очень хорошего эффекта - ускорение до 98% - на функциях, входящих в состав высокопроизводительной библиотеки векторных вычислений для архитектуры “Эльбрус”.

Глава 4 посвящена методам удаления межпроцедурных избыточностей. Наличие вызовов функций в программах приводит к появлению неявных избыточностей в результирующем коде, основные виды которых перечислены на рис.1.

В качестве одного из основных средств выявления и устранения межпроцедурных избыточностей оптимизирующими компиляторами применяется подстановка тела вызываемой функции в точку вызова (инлайн-подстановка). Это позволяет устранять упомянутые выше типы избыточностей с помощью внутрицпроцедурных оптимизаций, таких как сбор общих подвыражений, удаление мертвого кода и т.п. Применительно к EPIC-архитектурам, подстановка тела функции в точку вызова обладает еще одним важным, с точки зрения производительности, эффектом. А именно, компилятор дополнительно получает возможность совместного планирования вызывающей и вызываемой процедур.

На практике, подстановка тела функции в точку вызова не всегда возможна и не всегда эффективна. Для принятия решения о возможности и целесообразности инлайн-подстановки в каждом конкретном случае компиляторами используется различная информация, как полученная с помощью статического анализа, так и собранная в процессе пробного исполнения программы. Количество произведенных инлайн-подстановок и устраненных межпроцедурных избыточностей напрямую зависит от того, насколько полную и достоверную информацию удастся собрать с помощью предварительного анализа.

Одним из негативных эффектов инлайн-подстановок является *увеличение размера исполняемого кода*. Поэтому оптимизирующий компилятор обычно старается найти компромисс, при котором достигается выигрыш производительности, но увеличение размера исполняемого кода остается в разумных пределах. Традиционно считается, что после инлайн-подстановки размер вызывающей процедуры будет равен сумме ее размера до подстановки и размера вызываемой процедуры. Однако при этом не

учитывается тот факт, что после подстановки появляется возможность упрощения вызывающей процедуры за счет удаления упомянутых ранее избыточностей, то есть ее реальный размер может быть уменьшен. В диссертационной работе предложен метод анализа, позволяющий более точно прогнозировать размер вызывающей процедуры после инлайн-подстановки и удаления избыточностей. Этот метод, названный *межпроцедурной нумерацией значений*, позволяет выявлять конгруэнтность операций, принадлежащих разным процедурам. Имея на руках результаты анализа, можно точнее оценить размер вызывающей процедуры после подстановки. Так, если в вызывающей и вызываемой процедурах есть операции с одинаковыми классами конгруэнтности, то после инлайн-подстановки избыточные операции можно будет удалить. Поэтому правильной оценкой будет

res_size = before_size1 + before_size2 – common_opers_num

где *res_size* – итоговый размер вызывающей процедуры после подстановки, *before_size1* и *before_size2* - размеры вызывающий и вызываемой процедур до подстановки, а *common_opers_num* – число эквивалентных операций в вызывающей и вызываемой процедурах.

Справедливости ради, нужно сказать, что в реальных программах контекст для такой специализированной подстановки встречается довольно редко. Реализованная в составе оптимизирующего компилятора, инлайн-подстановка на основе результатов межпроцедурной нумерации значений применилась лишь на двух тестах из пакета SPEC CPU2000 и одном тесте из пакета SPEC CPU95. Максимальное ускорение составило 3%. При этом размер исполняемой программы увеличился незначительно.

В реальных программах зачастую встречаются *вызовы процедуры* не по имени, а по *адресу* (косвенные вызовы, неявные вызовы). В этом случае компилятор не имеет возможности выполнять инлайн-подстановку, так как неизвестно тело какой процедуры следует подставлять. Известный подход

к решению данной проблемы состоит в *статическом межпроцедурном анализе указателей*, с помощью которого иногда удается узнать, какая именно процедура вызывается в данной точке. Однако в большинстве случаев возможностей статического анализа хватает лишь на то, чтобы немного сократить множество процедур, которые могут быть потенциально вызваны в данной точке.

В диссертационной работе предложен динамический метод сбора статистической информации о количестве и частоте вызовов той или иной процедуры в данной точке. Указанная информация собирается в процессе пробного запуска программы, называемого *профилированием*. Предложенный *метод профилирования адресов косвенных вызовов* позволяет узнать, какие процедуры и как часто вызываются в данной точке программы, а также вычислить вероятность вызова конкретной процедуры.

На основании полученной статистической информации, в работе предложен метод *условной инлайн-подстановки*. Он сочетает в себе последовательную специализацию кода и, собственно, подстановку тела функции в точку вызова. Суть первой состоит в том, что если вероятность вызова в данной точке программы конкретной процедуры `func` превосходит некоторый порог, то непосредственно перед этой точкой строится ветвление управления. Условием ветвления полагается совпадение адреса вызова и адреса процедуры `func`. В случае если эти адреса совпадают, управление передается на участок кода, содержащий явный вызов `func`. В противном случае, управление передается на исходный участок кода, содержащий косвенный вызов. Теперь, после того, как в программе появился явный вызов, компилятор в состоянии выполнить подстановку тела процедуры `func` в точку вызова.

Методы профилирования адресов косвенных вызовов и последующего использования полученой информации в условных инлайн-подстановках были реализованы в составе оптимизирующего компилятора. Очевидно, они применимы лишь к программам, содержащим неявные

вызовы. В пакете SPEC CPU2000 содержится 5 таких программ, на которых удалось достичь ускорения до 14,5%.

Заключение

В диссертационной работе рассматривается проблема устранения избыточностей в процессе оптимизирующей компиляции. Произведенный анализ известных методов выявления и удаления избыточностей, основным из которых является метод нумерации значений, показал, что в ряде случаев их эффективность недостаточна. Исходя из этого, были предложены способы усиления известных алгоритмов и представлены оригинальные алгоритмы, эффективность которых доказана экспериментально.

В процессе решения этих задач автором были получены следующие **результаты, выносимые на защиту**:

1. Исследованы методы нумерации значений, встречающиеся в литературе, проведен их сравнительный анализ, выявлены их достоинства и недостатки. На этом основании *предложены направления развития методов нумерации значений.*

2. Разработан *усиленный метод нумерации значений*, включающий в себя анализ обращений к памяти, использование PS-форм и сбор информации о константных свойствах операций; за счет этого был получен прирост производительности до 28,5% и 6% на тестах пакетов SPEC CPU2000 и SPEC CPU95, исполненных на микропроцессоре с архитектурой «Эльбрус»; кроме того, установлена применимость метода к другим архитектурам - при исполнении тестов пакета SPEC CPU95 микропроцессором архитектуры Sparc достигнуто ускорение до 6%.

3. Предложен метод оптимизации программ на основе *вычисления векторов значащих битов*, который позволил достичь ускорения до 2,7% на задачах пакета SPEC CINT2000 и до 6,9% на задачах пакета SPEC CPU95 (архитектура «Эльбрус»); метод также продемонстрировал эффективность для микропроцессора Sparc (ускорение до 2,5% на пакете SPEC CPU95).

Перечисленные далее методы разработаны применительно к EPIC-архитектурам. Их эффективность экспериментально установлена в процессе замеров производительности для микропроцессора «Эльбрус».

4. Предложен метод *удаления избыточностей и упрощения предикатных выражений* на предикатной форме промежуточного представления, который позволил получить прирост производительности до 7% на тестах пакета SPEC CPU2000 и до 7,3% на тестах пакета SPEC CPU95.

5. Предложен метод *многоцелевой балансировки деревьев ассоциативных выражений*, который позволил получить прирост производительности до 15,5% на тестах пакета SPEC CPU2000 и до 19,5% на тестах пакета SPEC CPU95.

6. Предложен метод построения *комбинированных операций* в контексте аппаратно-конвейеризованных циклов, который позволил достичь ускорения до 6% на тестах пакета SPEC CPU95 и до 98% на функциях, входящих в состав высокопроизводительной библиотеки векторных вычислений для архитектуры «Эльбрус».

7. Предложен метод проведения *межпроцедурной нумерации значений* и последующего использования результатов анализа в инлайн-

подстановках, который позволил достичь ускорения до 3% на ряде задач.

8. Предложен метод *специализированной подстановки тела функции в точку вызова на основе профиля значений*, который позволил получить прирост производительности до 14,5% на задачах, содержащих неявные вызовы.

Все представленные в диссертационной работе методы и алгоритмы были реализованы в составе оптимизирующего компилятора с языков Си, Си++ и Фортран для микропроцессора «Эльбрус», а те из них, которые не зависят от целевой платформы, реализованы и в составе оптимизирующего компилятора для микропроцессора Sparc. Использование этих методов и алгоритмов вносит ощутимый вклад в задачу повышения производительности упомянутых вычислительных систем. Кроме того, результаты исследования могут быть адаптированы для использования в других вычислительных системах, ориентированных на высокопроизводительные вычисления.

Список работ, опубликованных по теме диссертации

1. Филиппов А.Н., Шлыков С.Л. Удаление частичных избыточностей // Высокопроизводительные вычислительные системы и микропроцессоры: сборник трудов ИМВС РАН, Выпуск №9, 2006, С. 49-57

2. Филиппов А.Н. Метод нумерации значений и использование его результатов в оптимизациях программ// Информационные технологии, №4, 2009, С. 43 - 49

3. Филиппов А.Н. Вычисление диапазонов бит для арифметических и логических выражений" // Научные труды XXXII Международной молодежной научной конференции « Гагаринские чтения», т. 6. М. : МАТИ, 2006, С. 185 – 186

4. Филиппов А.Н. Метод нумерации значений и его использование в оптимизациях программ // Научные труды XXXIII Международной молодежной научной конференции « Гагаринские чтения», т. 6. М. : МАТИ, 2007, С. 262-263

5. Филиппов А.Н. Метод нумерации значений для операций обращения к памяти // Труды 51-й научной конференции МФТИ «Современные проблемы фундаментальных и прикладных наук» Часть I. Радиотехника и кибернетика. — М.: МФТИ, 2008, С. 58-60

6. Филиппов А.Н. Условная подстановка тела процедуры в место вызова на основе профильной информации // Научные труды XXXV Международной молодежной научной конференции « Гагаринские чтения», т. 4. М. : МАТИ, 2009, С. 166-167