УДК 004.4'416

На правах рукописи

Ермолицкий Александр Викторович

МЕТОДЫ АВТОМАТИЧЕСКОЙ ВЕКТОРИЗАЦИИ НА ЭТАПЕ КОМПИЛЯЦИИ ДЛЯ АРХИТЕКТУР С ПОДДЕРЖКОЙ КОРОТКИХ ВЕКТОРНЫХ ИНСТРУКЦИЙ

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

АВТОРЕФЕРАТ
диссертации на соискание ученой степени
кандидата технических наук

Москва – 2011

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

Научный руководитель:	кандидат физико-математических наук Нейман-заде Мурад Искендер-оглы
Официальные оппоненты:	доктор физико-математических наук, старший научный сотрудник Галатенко Владимир Антонович кандидат технических наук Чернис Евгений Натанович
Ведущая организация:	ОАО «Институт точной механики и вычислительной техники им. С.А.Лебедева»

Защита состоится «8» июня 2011 г. в 14 часов на заседании диссертационного совета *Д 409.009.01* при **ОАО «Институт электронных управляющих машин им. И.С.Брука»** , расположенном по адресу: 119334, г.Москва, ул.Вавилова д.24

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

Автореферат разослан «26» апреля 2011 г.

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

Общая характеристика работы

Актуальность работы. Широкое распространение мультимедийных приложений привело к тому, что сейчас практически во всех микропроцессорах общего назначения присутствуют наборы коротких векторных инструкций. Это важное архитектурное нововведение последних десятилетий, которое позволяет значительно увеличить производительность процессора на мультимедийных и вычислительных задачах, в которых присутствует параллелизм на уровне данных.

Одним из наиболее распространенных в настоящее время методов введения векторных инструкций в код программы является использование ассемблерных вставок или вызовов специальных библиотечных функций. Оно наиболее эффективно с точки зрения производительности конечного кода, однако, приводит к существенному увеличению сложности разработки и негативно влияет на переносимость программных продуктов. Кроме того, появление новых векторных инструкций в архитектуре современных процессоров зачастую приводит к необходимости дорабатывать давно отлаженный код для повышения его производительности.

Альтернативный подход заключается в автоматической генерации векторных инструкций оптимизирующим компилятором. Это позволяет избежать указанных выше проблем, свойственных низкоуровневому программированию, и значительно упростить создание новых и усовершенствование уже разработанных высокопроизводительных приложений. Становится возможным использование векторных инструкций в программах, полностью написанных на языках высокого уровня.

Оптимизирующие компиляторы, способные автоматически векторизовать программы, написанные на языке высокого уровня, появились в 1970-х годах. Несколько десятилетий исследований в этой области позволили достичь

значительных результатов в повышении производительности приложений за счет векторизации. Однако алгоритмы векторизации, разработанные для традиционных векторных процессоров, не позволяют эффективно ускорять современные мультимедийные приложения по двум причинам. Во-первых, классические векторные инструкции, для которых создавались эти алгоритмы, и современные короткие векторные инструкции имеют ряд существенных различий. Во-вторых, многие современные приложения имеют особенности, несвойственные задачам, для которых создавались эти алгоритмы.

Таким образом, с появлением коротких векторных инструкций в составе современных процессоров возникла необходимость создания новых алгоритмов автоматической векторизации. За последнее десятилетие было предложено множество алгоритмов, адаптированных к коротким векторным инструкциям, они были внедрены в различные оптимизирующие компиляторы. Тем не менее, проблема автоматической векторизации в ряде важных случаев не была решена либо была решена недостаточно эффективно.

Существенным фактором является наличие в циклах *разветвлений управления и выходов не по счетчику.* Методы векторизации подобных циклов, разработанные для традиционных векторных машин, не могут быть непосредственно использованы для современных процессоров с короткими векторными инструкциями, поскольку в их архитектуре отсутствует поддержка предикатного исполнения векторных инструкций. Современные алгоритмы не решают эту проблему в общем виде, а позволяют векторизовать лишь простые шаблонные разветвления управления.

Другой важной проблемой является *ручная оптимизация кода.* Довольно часто горячие циклы современных приложений оптимизируются разработчиками для повышения производительности. Наиболее сильно распространена практика ручной раскрутки циклов, а также использование массивов для хранения предварительно вычисленных значений. Подобные оптимизации со-

здают сложности для автоматической векторизации, требуя разработки дополнительных анализов и преобразований программы. Известные методы не позволяют эффективно решить эту проблему.

Наконец, одной из основных проблем является *недостаток информации о компилируемой задаче*. В частности, это информация о выровненности адресов обращений к памяти. Аппаратные ограничения, накладываемые в большинстве современных архитектур на выровненность адреса векторных инструкций обращения к памяти, приводят к значительному снижению эффективности генерируемого векторного кода в случае отсутствия информации о выровненности. Современные методы векторизации позволяют решить данную проблему лишь частично, используя различные вспомогательные преобразования скалярного кода и специальные техники генерации векторного кода.

Таким образом, существующие методы введения коротких векторных инструкций в код программы не позволяют в полной мере раскрыть резерв повышения производительности современных микропроцессоров. Это обусловливает **актуальность** развития методов автоматической векторизации как наиболее перспективного способа использования этого резерва.

Цель исследования. Целью исследования явилось увеличение эффективности оптимизирующего компилятора путем разработки новых и усовершенствования известных методов автоматической векторизации для архитектур с поддержкой коротких векторных инструкций. В соответствии с этим были поставлены следующие задачи:

- анализ существующих и разработка новых методов автоматической векторизации кода без разветвлений управления;
- разработка метода векторизации кода, содержащего произвольные разветвления управления и выходы не по счетчику;

- разработка эффективного метода векторизации раскрученных вручную циклов;
- анализ существующих и разработка новых вспомогательных преобразований скалярного кода, повышающих эффективность автоматической векторизации;
- реализация указанных методов в составе оптимизирующего компилятора для микропроцессора «Эльбрус» и Sparc V9.

Методы исследования заимствованы из областей системного программирования, технологии компиляции, теории графов и теории алгоритмов.

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

- усовершенствованный алгоритм векторизации кода без разветвлений управления, позволяющий векторизовать сложные рекуррентные выражения, ациклический код и раскрученные вручную циклы;
- новый алгоритм векторизации циклов, содержащих произвольные разветвления управления и любое количество выходов не по счетчику;
- усовершенствованный алгоритм выравнивания инструкций обращения к памяти;
- новый алгоритм скрутки раскрученных вручную циклов.

Практическая ценность работы состоит в создании новых и усовершенствовании известных методов автоматической векторизации. Все представленные в диссертационной работе алгоритмы и методы реализованы в составе оптимизирующего компилятора языков высокого уровня Си, Си++,

Фортран для микропроцессора «Эльбрус» и Sparc V9. Эффективность предложенных методов подтверждена замерами производительности на задачах из пакетов SPEC CPU92, SPEC CPU95, SPEC CPU2000, а также на функциях высокопроизводительной библиотеки EML.

Апробация

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

- на V международной научно-практической конференции «Современные информационные технологии и ИТ-образование» , Москва, МГУ, 2010 г.;
- on The IASTED International Conference on Informatics «Parallel and Distributed Computing and Systems» , Marina del Rey, USA, 2010;
- на V международной конференции «Параллельные вычисления и задачи управления» , Москва, ИПУ РАН, 2010 г.;
- на XLIX научной конференции «Современные проблемы фундаментальных и прикладных наук» , Москва, МФТИ, 2006 г.;
- на XLVIII научной конференции «Современные проблемы фундаментальных и прикладных наук» , Москва, МФТИ, 2005 г.

Публикации

По теме диссертации опубликовано 8 печатных работ, одна из которых - в журнале списка ВАК.

Структура и объем работы

Диссертация состоит из введения, трех глав и заключения. Список литературы составляет 63 наименования. Объем диссертации составляет 138 страниц текста. Диссертация содержит 57 рисунков.

Содержание работы

Во **Введении** обоснована актуальность темы диссертационной работы, сформулирована ее цель и научная новизна исследований, показана практическая ценность.

Глава 1 посвящена обзору и анализу существующих методов автоматической векторизации и вспомогательных преобразований, адаптированных к коротким векторным инструкциям.

Во вступительной части описываются короткие векторные инструкции, типичные для современных микропроцессоров общего назначения. Приводятся их отличия от векторных инструкций традиционных векторных машин. Анализируются формальные условия применимости классических алгоритмов векторизации и их адаптация к современным архитектурам. Обосновывается необходимость разработки методов векторизации для современных процессоров общего назначения.

Далее рассматриваются известные методы векторизации, рассчитанные на короткие векторные инструкции. Их суть заключается в преобразовании цикла, содержащего скалярные инструкции, в цикл с векторными инструкциями. Это позволяет уменьшить время работы цикла за счет параллельного исполнения его итераций. По способу преобразования скалярных вычислений в векторные вычисления методы векторизации можно разделить на три класса:

1. *Методы векторизации, основанные на алгоритмах, которые используются для традиционных векторных машин.* В них скалярный цикл трансформируется в ациклический код для абстрактной векторной машины, способной работать с векторами произвольной длины, после чего этот код преобразуется в цикл с короткими векторными инструкциями.

2. *Методы векторизации на уровне цикла.* Используется раскрутка цикла (дублирование итераций цикла) с последующей заменой в цикле групп изоморфных скалярных инструкций соответствующими векторными инструкциями. При этом фазы раскрутки и генерации инструкций тесно взаимосвязаны.

3. *Векторизация на уровне линейного участка.* Раскрутка циклов используется в качестве независимого базового преобразования. Далее алгоритм векторизации применяется ко всем (в том числе ациклическим) линейным участкам программы, итеративно заменяя группы скалярных инструкций соответствующими векторными там, где это возможно.

Далее рассматриваются известные методы повышения эффективности автоматической векторизации за счет выравнивания адресов обращений к памяти. Инструкция обращения к памяти является выровненной, если адрес ячейки памяти, к которой она обращается, кратен формату инструкции. В результате векторизации формат инструкций обращения к памяти увеличивается, из-за чего эти инструкции могут стать невыровненными. В зависимости от архитектуры, обращение по невыровненному адресу приводит к возникновению блокировки либо исключительной ситуации. Для решения данной проблемы существуют известные вспомогательные преобразования, такие как *открутка итераций цикла* (Loop Peeling), *динамические проверки выровненности с созданием версий цикла* (Align Versioning) и *дополнение массивов* (Array Padding). Эти преобразования, выполняемые перед векторизацией, выравнивают либо динамически проверяют выровненность инструкций обращения к памяти, что позволяет значительно повысить эффективность автоматической векторизации во многих случаях.

Несмотря на свое многообразие, исследованные методы векторизации и вспомогательные преобразования имеют ряд недостатков.

Во-первых, все рассмотренные методы имеют *ограничения по применимости*. Так, ни один из них не позволяет векторизовать циклы с более чем одним выходом и циклы без выхода по счетчику. Они также не позволяют векторизовать код с произвольными разветвлениями управления на уровне цикла. Такие циклы встречаются на практике, особенно часто в целочисленных задачах.

Во-вторых, рассмотренные методы позволяют векторизовать только простейшие *рекуррентные выражения*, в которых данные, выработанные на одной итерации, потребляются на следующей итерации.

В-третьих, ни один из рассмотренных методов не позволяет эффективно векторизовать *циклы, раскрученные программистом*. Так, метод векторизации на уровне линейного участка позволяет векторизовать раскрученные циклы, однако при этом к циклу не могут применяться вспомогательные преобразования.

В-четвертых, известные вспомогательные преобразования применимы лишь к отдельным инструкциям, выполняющим в цикле обращение к смежным фрагментам памяти (*одношаговые инструкции*), в то время как в приложениях встречаются группы взаимосвязанных инструкций, для которых это условие не выполняется.

Наконец, все рассмотренные методы основаны на использовании *профильной информации* для определения целесообразности применения векторизации. Данный подход работает эффективно только в случае хорошей корреляции поведения программы при получении профиля и при реальном исполнении, однако, для некоторых приложений это условие не выполняется.

Глава 2 посвящена описанию нового алгоритма векторизации, который позволяет снять описанные выше ограничения известных методов. В качестве базового используется алгоритм векторизации на уровне цикла, поскольку он

является наиболее универсальным и простым в реализации, не уступая при этом по эффективности другим методам.

Расширение применимости алгоритма автоматической векторизации к циклам с произвольными разветвлениями управления основывается на понятии *векторного предиката* (**ВП**) - вектора, элементы которого хранят предикатные значения. При векторизации каждой дуге/узлу управляющего графа исходного цикла ставится в соответствие ВП, j-ый элемент которого хранит значение *true* на i-ой итерации векторизованного цикла, если по соответствующей дуге или в соответствующий узел передается управление на k-ой итерации исходного цикла ($k = i * L_v + j$, L_v - количество элементов ВП), и *false* в противном случае.

В отличие от традиционных векторных машин, современные процессоры общего назначения не поддерживают предикатное исполнение векторных инструкций. Вместо этого они предоставляют другой механизм поддержки вычислений с разветвлениями управления. В большинстве современных векторных расширений ВП представляет собой *битовую маску* - обычный векторный регистр, элемент которого заполняется единичными битами в случае значения *true* и нулевыми в случае значения *false*. Векторные предикаты вырабатываются инструкциями векторного сравнения, над ними определены основные булевы операторы (AND, OR, NOT и т.п.).

Предложенный в работе **алгоритм битового маскирования** позволяет векторизовать выражения с произвольными разветвлениями управления. Алгоритм вычисляет ВП всех дуг и узлов цикла и использует их для сведения потоков данных, вырабатываемых векторными образами инструкций с разных веток управления. При этом в векторизованном цикле все инструкции исполняются безусловно.

Код, полученный в результате векторизации цикла со сложными разветвлениями управления, зачастую содержит множество избыточностей. Для их

удаления в работе предложен ряд локальных оптимизаций векторных предикатных вычислений. Наряду с более чем 30 шаблонными преобразованиями, такими как $X - ((X\&C)|(Y\&!C)) \rightarrow (X - Y)\&!C$ (где X и Y - произвольные вектора, а C - векторный предикат), выполняется оптимизация предикатных вычислений на основе построения дизъюнктивных нормальных форм (ДНФ) и последующего их упрощения с помощью правил булевой логики.

Существующие алгоритмы автоматической векторизации применимы лишь к *исчислимым* циклам, то есть к таким, количество итераций которых можно вычислить, не исполняя цикл. Применение этих алгоритмов к неисчислимым циклам может привести к исполнению лишних итераций и, как следствие, некорректному поведению программы. На практике исчислимость удобно определять по отсутствию *боковых выходов* из цикла. Выход из цикла, осуществляемый по достижению индуктивной переменной некоторого инвариантного значения, называется *выходом по счетчику. Боковым выходом* является выход из цикла, не являющийся выходом по счетчику. Таким образом, цикл является исчислимым при отсутствии боковых выходов.

Расширение применимости алгоритма векторизации к циклам с боковыми выходами основано на предварительных преобразованиях, выполняемых непосредственно перед векторизацией: реорганизации кода в цикле и построении *компенсирующего кода*. Реорганизация кода заключается в переносе в нижнюю часть цикла инструкций с побочными эффектами (записи в память и инструкции, результат которых используется за пределами цикла), так, чтобы исполнение этих инструкций и выход из цикла по боковому выходу не могли произойти на одной итерации. Это позволяет избежать нежелательных последствий при исполнении лишних итераций цикла. Компенсирующий код (КК), исполняемый при выходе из цикла по боковому выходу, представляет собой копию исходного цикла. КК пересчитывает значения, вычисленные на последней итерации векторизованного цикла, что позволяет компенсировать

изменения, вызванные реорганизацией кода.

Реализация алгоритма векторизации циклов с разветвлениями управления и боковыми выходами позволила повысить производительность в 1.83 раза на задаче 023.eqntott (пакет тестов SPEC CINT92) и в 2.17 раз на задаче 124.m88ksim (SPEC CINT95). Эти целочисленные задачи содержат циклы с боковыми выходами, на которые приходится значительная часть времени исполнения.

Выражение, содержащее *рекуррентную зависимость*, в общем случае не векторизуется. Тем не менее, векторизация возможна в важном частном случае, когда выражение представляется в виде:

$$x = f(x, g(i)) \tag{1}$$

где x - скалярная переменная, i - индуктивная переменная цикла[1], $f()$ - ассоциативная арифметическая функция (*оператор редукции*), например, сложение, $g()$ - нерекуррентное векторизуемое подвыражение. Все известные методы векторизации накладывают дополнительное ограничение - оператор редукции должен быть реализован посредством одной арифметической инструкции.

В работе предложен метод ослабления последнего ограничения. Он заключается в предварительной (до раскрутки цикла) замене шаблонных семантических конструкций фиктивными скалярными инструкциями, такими как вычисление максимума двух величин или сложение с насыщением результата. Это позволяет в ряде случаев упростить оператор редукции до одной инструкции, так что ограничение оказывается выполненным. На этапе векторизации фиктивные скалярные инструкции заменяются соответствующими фиктивными векторными инструкциями. Далее, все фиктивные инструкции

[1]Переменная, изменяющаяся на инвариантную величину на каждой итерации цикла

(как скалярные, так и векторные) заменяются реальными, доступными в целевой архитектуре.

Кроме того, в работе показано, что данный подход может быть использован для векторизации некоторых рекуррентных выражений, не представимых в виде (1). В частности, был предложен метод векторизации оператора сдвига массива в памяти, основанный на использовании фиктивной инструкции сдвига. Предложенный метод векторизации сложных рекуррентностей позволил на 11% увеличить производительность задачи 256.bzip2 из пакета SPEC CINT2000.

В отличие от векторизации на уровне линейного участка описанные в литературе методы векторизации на уровне цикла неприменимы к ациклическим участкам кода. В диссертационной работе показано, каким образом применимость предлагаемого метода может быть расширена на ациклические участки. Для этого используется *анализ изоморфизма*, позволяющий для любой инструкции линейного участка найти множество изоморфных ей инструкций. Используя данную информацию, компилятор заменяет группы изоморфных скалярных инструкций линейного участка соответствующими векторными инструкциями. Эта доработка позволила получить прирост производительности в 2% на задаче 252.eon из пакета SPEC CINT2000.

Практическая применимость предложенных в главе алгоритмов и методов исследовалась также на 373 функциях библиотеки EML(Elbrus Math Library), реализующих наиболее распространенные операции над векторами и матрицами. Предложенный алгоритм векторизации позволил дополнительно ускорить функции EMLв среднем[2] на 6.3% по сравнению с известными алгоритмами; на отдельных функциях показатель роста производительности составил до 9.76 раз.

[2]для усреднения использовалось среднее геометрическое

Глава 3 посвящена методам повышения эффективности векторизации за счет вспомогательных преобразований.

Основным фактором, влияющим на эффективность автоматической векторизации, является выровненность инструкций обращения к памяти. Векторизация невыровненных инструкций требует значительных накладных расходов, для уменьшения которых необходимо минимизировать количество невыровненных инструкций в цикле. Для этого в диссертационной работе предложен **алгоритм выравнивания инструкций**, позволяющий увеличить эффективность автоматической векторизации.

Формально задача выравнивания может быть сформулирована следующим образом. Имеется цикл, содержащий M инструкций обращения к памяти, часть из которых (возможно, все) могут быть невыровненными. Каждая инструкция имеет некоторый вес w_i, характеризующий величину накладных расходов на ее векторизацию в случае, когда она является невыровненной. Требуется выполнить вспомогательные (выравнивающие) преобразования, минимизирующие сумму:

$$\sum_{i=1}^{M} x_i w_i \to min, \tag{2}$$

где x_i равен 0, если инструкция выровнена, и 1 в противном случае. Известные алгоритмы выравнивания позволяют гарантированно выровнять только одну инструкцию в цикле, в то время как в реальных задачах встречаются циклы, содержащие множество инструкций обращения к памяти по коррелирующим адресам.

Предложенный в работе алгоритм выравнивания позволяет минимизировать функционал (2) с помощью наименьшего количества вспомогательных преобразований, таких как откртка итераций и создание версий цикла. Данный алгоритм основан на поиске в цикле групп инструкций с эквивалентной

выровненностью. Применение выравнивающего преобразования к одной инструкции группы автоматически делает выровненными все инструкции этой группы, что позволяет выровнять большее количество инструкций, используя при этом меньшее количество преобразований. Группа инструкций с максимальным весом[3] выравнивается с помощью откртки итераций, выровненность остальных групп проверяется с помощью динамических проверок с созданием версий цикла.

Алгоритм нахождения групп инструкций с одинаковой выровненностью базируется на использовании техники *PS-форм* (форма сумм произведений, Product Sums Form). PS-формой называется полином вида: $c_0 + c_1 x_{1,1} x_{1,2} ... x_{1,k_1} + ... + c_n x_{n,1} x_{n,2} ... x_{n,k_n}$, где c_0, c_1, ..., c_n - некоторые константы, а $x_{i,j}$ - результаты некоторых инструкций. В работе доказывается возможность представления адреса любой инструкции обращения к памяти в виде PS-формы в случае представления программы в форме SSA (форма статического единственного присваивания, Static Single Assignment). Показывается, каким образом с помощью представления адреса в виде PS-формы можно определить выровненность адреса и эквивалентность выровненности адресов двух инструкций. Использование PS-форм позволяет значительно увеличить эффективность алгоритма поиска инструкций с эквивалентной выровненностью.

В работе предложен **метод частичной откртки итераций цикла**, позволяющий гарантированно выровнять группы инструкций обращения к памяти в важном частном случае. Известные выравнивающие преобразования применимы лишь к *одношаговым инструкциям* обращения к памяти[4]. Однако на практике нередко встречаются группы неодношаговых инструкций, обращающихся к смежным ячейкам памяти. Например, такая ситуация

[3]Вес группы равен сумме весов входящих в нее инструкций

[4]Изменение (шаг) адреса которых за итерацию цикла равно формату инструкции

возникает при работе с комплексными числами, мнимая и действительная часть которых хранится в одном массиве. Для формального описания подобной ситуации необходимо ввести понятие *кортежа* - множества инструкций чтения/записи одинакового формата, обращающихся к смежным ячейкам памяти: {a[i], a[i+1], a[i+2], ..., a[i+k-1]}, где **a** - некоторый массив, а **k** - длина кортежа. Классическая откртука итераций цикла позволяет выровнять кортеж, только если следующее уравнение имеет целочисленное решение x для любых m ($0 \leq m < L_v$):

$$m + Sx = 0 \bmod L_v \tag{3}$$

где m - величина невыровненности первой инструкции кортежа, S - шаг адреса инструкций кортежа за итерацию цикла, x - количество откручиваемых итераций цикла, L_v - длина векторных регистров (все параметры измеряются в числе элементов векторных регистров). Уравнение (3) имеет решение для произвольного m, если S и L_v являются взаимно простыми. Поскольку для всех современных архитектур $L_v = 2^n$ ($n \geq 1$), то выравнивание инструкций путем откртуки итераций цикла возможно только для кортежа с нечетным шагом S. Это условие выполняется не всегда, например, при работе с комплексными числами S является четным.

Предложенный в диссертационной работе *метод частичной откртуки итераций цикла* позволяет выровнять кортеж в случае, когда уравнение (3) не имеет решений. Данное преобразование заключается в создании динамических проверок выровненности и копий цикла, и дальнейшем выносе из копий цикла части нескольких первых и последних итераций. В результате преобразования выравниваемый кортеж инструкций оказывается выровненным во всех копиях цикла.

Реализация предложенного алгоритма выравнивания, способного вырав-

нивать кортежи инструкций обращения к памяти, позволила дополнительно повысить эффективность векторизации. Величина прироста производительности составила 39% на задаче 102.swim из пакета SPEC CFP95.

В определенных случаях получению заметного эффекта от вспомогательных преобразований препятствует практикуемая программистами в целях оптимизации «ручная раскрутка» (дублирование тела) цикла. Для конкретной архитектуры она может не дать предполагаемого результата и препятствовать работе других оптимизаций. В частности, для таких циклов невозможно применение большинства вспомогательных преобразований, необходимых для эффективной векторизации. Для решения данной проблемы в работе предложен **метод скрутки цикла** - преобразование, обратное раскрутке цикла. Алгоритм скрутки основан на поиске изоморфных выражений в цикле и удалении созданных программистом копий тела цикла. Он выполняется перед остальными вспомогательными преобразованиями. Реализация предложенного метода в оптимизирующем компиляторе позволило достичь ускорения на 11% задачи 256.bzip2 из пакета тестов SPEC CINT2000.

Эффективность векторизации зависит от количества итераций цикла - если в цикле мало итераций, то накладные расходы могут нивелировать положительный эффект векторизации. Сейчас общепринятым является подход, когда при помощи профилирования или каких-либо эвристик оценивается среднее число итераций цикла, на основе которого принимается решение о целесообразности применения той или иной оптимизации. Однако такой подход оказывается неэффективным в случае, когда количество итераций цикла значительно меняется от запуска к запуску, а также в случае неправильного предсказания компилятором вероятностей переходов. Например, при компиляции библиотечных функций невозможно предсказать с какими параметра-

ми будут вызваны эти функции, и, как следствие, невозможно предсказать количество итераций большинства циклов в этих функциях.

Предложенный в работе **метод динамического арбитра** позволяет решить указанную проблему следующим образом. В случае если компилятор не может статически определить среднее количество итераций цикла, создается копия цикла и вставляется динамическая проверка, передающая управление на многоитерационную версию цикла, если вычисленное число итераций цикла больше определенного порога, или на малоитерационную версию цикла, если число итераций ниже этого порога. Далее многоитерационная версия цикла может быть векторизована, а другая версия останется скалярной. Таким образом, при исполнении программы выбирается наиболее эффективная версия цикла.

Реализация динамического арбитра позволила существенно увеличить эффективность автоматической векторизации на функциях библиотеки EML. Средний прирост производительности составил 43%.

Заключение

В диссертационной работе рассматривается проблема автоматической генерации коротких векторных инструкций в процессе оптимизирующей компиляции. Проведенные эксперименты показывают, что такие инструкции позволяют значительно повысить производительность вычислительных комплексов на задачах, содержащих большое количество параллелизма на уровне данных. Проведенный анализ описанных в литературе методов автоматической генерации векторных инструкций показал недостаточную их эффективность в ряде случаев. Исходя из результатов анализа, были предложены направления развития известных алгоритмов, а также представлены оригинальные алгоритмы и методы, эффективность которых подтверждена экспе-

риментально.

В процессе исследований и в ходе решения поставленных задач автором были получены следующие **результаты, выносимые на защиту**:

1. Исследованы методы автоматической векторизации и вспомогательных преобразований, описанные в литературе, проведен их сравнительный анализ, выявлены их достоинства и недостатки. На основании проведенного анализа предложены направления развития этих преобразований.

2. Разработан усовершенствованный алгоритм векторизации циклов на основе раскрутки, позволяющий векторизовать сложные рекуррентные выражения, ациклический код и раскрученные вручную циклы. Данный алгоритм позволил получить прирост производительности до 11% на тестах пакета SPEC CINT2000.

3. Разработан новый алгоритм векторизации циклов с произвольными разветвлениями управления и боковыми выходами, что позволило получить прирост производительности до 83% и 117% на тестах пакетов SPEC CINT92 и SPEC CINT95 соответственно, а также - до 9.76 раз на функциях высокопроизводительной библиотеки векторных вычислений EML.

4. Разработан усовершенствованный алгоритм выравнивания инструкций обращения к памяти, способный выравнивать группы кортежей инструкций, реализация которого позволила повысить эффективность автоматической векторизации до 39% на тестах пакета SPEC CFP95.

5. Разработан новый алгоритм скрутки раскрученных вручную циклов, позволивший улучшить эффективность автоматической векторизации до 11% на тестах пакета SPEC CINT2000.

6. Предложен метод динамического арбитра, позволивший повысить эффективность автоматической векторизации в среднем на 43% на функциях EML.

Все представленные в диссертационной работе алгоритмы и методы реализованы в составе оптимизирующего компилятора с языков Си, Си++ и Фортран для микропроцессоров «Эльбрус» и Sparc V9 и играют важную роль в получении эффективного кода для этих вычислительных систем. Замеры производительности выполнялись на вычислительном комплексе «Эльбрус-3М1» . Разработанные методы и алгоритмы могут быть адаптированы и использоваться для различных микропроцессоров, содержащих короткие векторные инструкции.

Список работ, опубликованных по теме диссертации

1. Ермолицкий А.В., Шлыков С.Л. Автоматическая векторизация циклов со сложным управлением // сборник избранных трудов V международной научно-практической конференции «Современные информационные технологии и ИТ-образование» , М.: ИНТУИТ.РУ, 2010, С. 604-611;

2. Mukhanov L., Ilyin P., Shlykov S., Ermolitsky A., Breger A., Grabezhnoy A. Thread-level Automatic Parallelization in the Elbrus Optimizing Compiler // Proceedings of the The IASTED International Conference on Informatics «Parallel and Distributed Computing and Systems» , ACTA Press, Marina del Rey, USA, 2010;

3. Ермолицкий А.В., Нейман-заде М.И. Методы повышения эффективности автоматической векторизации вычислений // труды V международ-

ной конференции «Параллельные вычисления и задачи управления» , ИПУ РАН, 2010;

4. Ермолицкий А.В. Методы повышения эффективности векторизации в оптимизирующем компиляторе // Вопросы радиоэлектроники, №3, 2010, С. 41-50;

5. Ермолицкий А.В., Шлыков С.Л. Автоматическая векторизация выражений оптимизирующим компилятором // Приложение к журналу «Информационные технологии» №11, 2008, С. 17-21;

6. Ермолицкий А.В. Развитие метода автоматической векторизации циклов оптимизирующим компилятором // Труды XLIX научной конференции «Современные проблемы фундаментальных и прикладных наук» , Москва, МФТИ, 2006, С. 48;

7. Ермолицкий А.В. Автоматическая векторизация условных выражений оптимизирующим компилятором // Труды XLVIII научной конференции «Современные проблемы фундаментальных и прикладных наук» , Москва, МФТИ, 2005, С. 57;

8. Волконский В.Ю., Ермолицкий А.В., Ровинский Е.В. Развитие метода векторизации циклов при помощи оптимизирующего компилятора // Высокопроизводительные вычислительные системы и микропроцессоры №8, 2005, С. 34-56.