УДК 004.4'416

На правах рукописи

Галазин Александр Борисович

МЕТОДЫ ОПТИМИЗАЦИИ ДОСТУПА К ПОДСИСТЕМЕ ПАМЯТИ НА ЭТАПЕ КОМПИЛЯЦИИ ДЛЯ МИКРОПРОЦЕССОРНЫХ СИСТЕМ С АРХИТЕКТУРОЙ ШИРОКОГО КОМАНДНОГО СЛОВА

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

АВТОРЕФЕРАТ

диссертации на соискание ученой степени кандидата технических наук

Москва – 2008

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

Научный руководитель:	кандидат физико-математических наук Нейман-заде Мурад Искендер-оглы
Официальные оппоненты:	доктор технических наук, профессор Сухомлин Владимир Александрович кандидат технических наук Добров Андрей Дмитриевич
Ведущая организация:	Институт проблем информатики РАН

Защита состоится «____»_____ 2008 г. в ____ 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 «____»_____ 2008 г.

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

Общая характеристика работы

Актуальность темы. Одним из основных факторов увеличения быстродействия современных микропроцессорных вычислительных систем является сокращение времени выборки данных и кода из подсистемы памяти, позволяющее в полной мере использовать производительность процессоров системы. В связи с постоянным увеличением тактовой частоты, количества исполнительных устройств микропроцессора и развитием аппаратных и программных технологий распараллеливания вычислительного процесса этот фактор определяющим образом влияет на общую производительность системы. Известным методом решения проблемы является введение в состав системы иерархии кэш-памяти. В то же время, надо отметить, что используемые алгоритмы кэширования в ряде случаев не отличаются высоким уровнем эффективности в приложениях, связанных с интенсивной выборкой данных, значительную часть которых составляют научные и инженерные задачи.

Следующей немаловажной причиной, снижающей производительность, являются блокировки конвейера микропроцессора, возникающие при отсутствии необходимого кода. Алгоритмы кэширования часто не решают задачу сокращения времени выборки в силу незначительной пространственной локальности и редкого повторного использования кода. Проблема отсутствия кода особенно актуальна для системных приложений, таких как, операционные системы, оконные менеджеры и базы данных.

Для преодоления недостатков методов кэширования используются различные аппаратные, программные и комбинированные методы предварительной подкачки данных и кода. Они позволяют прогнозировать обращения в память и производить выборку в кэш или другое специальное устройство за некоторое время до использования. Эффективность таких методов во многом определяется тем, с помощью каких адресных

структур организовано в программе хранение и обращение к данным (в дальнейшем они характеризуются как «шаблоны хранения и обращения»).

Помимо этих проблем на производительность микропроцессоров непосредственно влияет физическая организация и размеры кэш-памяти. Наличие конфликтов по банкам памяти при интенсивной обработке данных может привести к блокировке устройств кэша и, как следствие, к блокировке конвейера микропроцессора, либо к вытеснению необходимых данных из кэша. Для сокращения числа конфликтов используют оптимизационные преобразования, повышающие локальность обращения к данным, либо преобразования размещения данных.

Обозначенные выше проблемы особенно актуальны для архитектур, обеспечивающих высокую скорость вычислений с использованием парадигмы широкого командного слова. Ее эффективная реализация предполагает введение большого количества исполнительных устройств, для которых следует обеспечить своевременный доступа к данным и коду. В то же время необходимо отметить, что известные методы, предварительной подкачки и эффективного использования кэш-памяти не в полной мере учитывают специфику архитектур с широким командным словом, а в некоторых случаях применительно к ним вообще не сформулированы.

Принципиально то обстоятельство, что в данном контексте усовершенствованные или разработанные методы должны быть реализованы в составе компилятора, ответственного за учет и оптимальное статическое распределение аппаратных ресурсов. Значение этого фактора в полной мере проявилось в процессе разработки архитектуры и программного обеспечения микропроцессоров серии «Эльбрус», на базе которых созданы и планируются вычислительные комплексы для ряда систем стратегического назначения. Это обусловливает **актуальность** исследования

и разработки эффективных методов доступа к подсистеме памяти, реализуемых на этапе компиляции.

Цель диссертационной работы. Конечной целью исследования являлась оптимизация выборки данных и кода из памяти в процессе компиляции на основе механизмов предварительной подкачки и эффективного использования аппаратных ресурсов. В соответствии с этой целью были поставлены следующие задачи:

- провести анализ особенностей микропроцессорной архитектуры широкого командного слова, влияющих на время выборки данных и кода;
- разработать и реализовать в составе компилятора эффективные методы предварительной подкачки, соответствующие основным шаблонам хранения и обращения к данным, подлежащих обработке;
- разработать методы оптимизации размещения данных в памяти для эффективного использования многобанкового кэша данных;
- разработать методы предварительной подкачки кода для архитектур с широким командным словом;
- реализовать указанные методы в составе оптимизирующего компилятора для микропроцессора «Эльбрус».

Научная новизна

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

- определение принципиальных особенностей аппаратуры микропроцессорных систем с архитектурой широкого командного слова, влияющих на взаимодействие процессоров с подсистемой памяти;

- реализованные в составе функций компилятора эффективные методы предварительной подкачки данных, основанные на расширенной классификации шаблонов хранения и обращения к данным, включающей 0-линейно регулярные, n-(не)линейно (псевдо)регулярные и n-кольцевые рекуррентные шаблоны;

- программный метод предварительной подкачки кода, ориентированный на архитектуру с широким командным словом;

- комплексный метод решения задачи эффективного использования многобанкового кэша данных, включающий в себя, преобразование сложных агрегатных структур данных к автономным массивам данных.

Практическая ценность результатов работы состоит в том, что на основе предложенных методов были разработаны реализованные на этапе компиляции методы эффективного взаимодействия с подсистемой памяти с учетом особенностей целевой архитектуры. Предложенные методы оптимизации взаимодействия процессоров и подсистемы памяти могут эффективно использоваться для микропроцессоров с архитектурой широкого командного слова. Все представленные в диссертационной работе алгоритмы и методы реализованы в составе оптимизирующего компилятора с языков высокого уровня Си, Си++, Фортран для микропроцессора «Эльбрус», и их эффективность подтверждена на пакетах задач SPEC CPU2000, SPEC CPU95.

Результаты, выносимые на защиту

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

1. Комплексный метод программной поддержки комбинированного метода предварительной подкачки 0-линейно регулярных данных,

который позволил в среднем повысить производительность на 24% на задачах пакета SPEC CPU2000.

2. Методы программной предварительной подкачки n-(не)линейно (псевдо)регулярных и n-кольцевых рекуррентных данных, учитывающие особенности архитектуры, что дополнительно в среднем улучшило на 18,5% и 9,1% показатели производительности задач с соответствующим контекстом.

3. Метод программной предварительной подкачки кода для архитектур с широким командным словом, который позволил в среднем на 3,2% ускорить задачи пакета SPEC CINT2000.

4. Метод определения оптимального взаимного расположения глобальных массивов, что дополнительно в среднем улучшило показатели производительности задач пакета SPEC CFP95 на 10% и пакета SPEC CFP2000 на 5,9%.

Апробация

Результаты, полученные в работе, изложены в ряде печатных публикаций, докладывались на научных конференциях и семинарах, в частности на XXXIV Международной молодежной научной конференции «Гагаринские чтения»(Москва, МАТИ, 2008 г.), XXIII научно-технической конференции «Направления развития и применения перспективных вычислительных средств и новых информационных технологий в ВВТ РКО»(Москва, в/ч 03425, 2007 г.), XIV Международной конференции студентов, аспирантов и молодых ученых «Ломоносов»(Москва, МГУ им. М. В. Ломоносова, 2007 г.), XXXIII Международной молодежной научной конференции «Гагаринские чтения»(Москва, МАТИ, 2007 г.), семинарах секции программного обеспечения ЗАО «МЦСТ».

Публикации

По теме диссертации опубликовано 10 печатных работ.

Структура и объем диссертации. Диссертация состоит из введения, четырех глав, заключения и приложения. Список литературы составляет 77 наименований. Объем диссертации составляет 149 страниц текста. Диссертация содержит 55 рисунков.

Содержание работы

Во Введении обоснована актуальность диссертационной работы, сформулирована цель и аргументирована научная новизна исследований, показана практическая значимость полученных результатов. Дана краткая характеристика содержания работы.

В первой главе изучаются особенности архитектуры, влияющие на производительность вычислительных систем с точки зрения обеспечения необходимого соответствия между быстродействием процессора и временными параметрами доступа к подсистеме памяти. В этом качестве особое значение имеет организация кэш-памяти данных и буфера инструкций, позволяющая обеспечить непрерывность работы конвейера микропроцессора.

В современных микропроцессорах кэш данных формируется из нескольких блоков (банков) памяти с независимым доступом, что позволяет одновременно исполнить несколько инструкций чтения/записи. В качестве примера можно привести системы Sun OpenSPARC T1, Intel Itanium 2. Многобанковая кэш-память является важным элементом, обеспечивающим максимальную загрузку исполнительных устройств архитектуры с широким командным словом. Она реализована в качестве кэш-памяти второго уровня (**L2$**) в микропроцессоре «Эльбрус».

Стандартная схема многобанковой кэш-памяти предполагает наличие арбитра, который осуществляет управление запросами от микропроцессора, коммутируя их и через схему приоритетов выдавая в банки кэша. При невозможности немедленного обслуживания, запросы буферизуют-

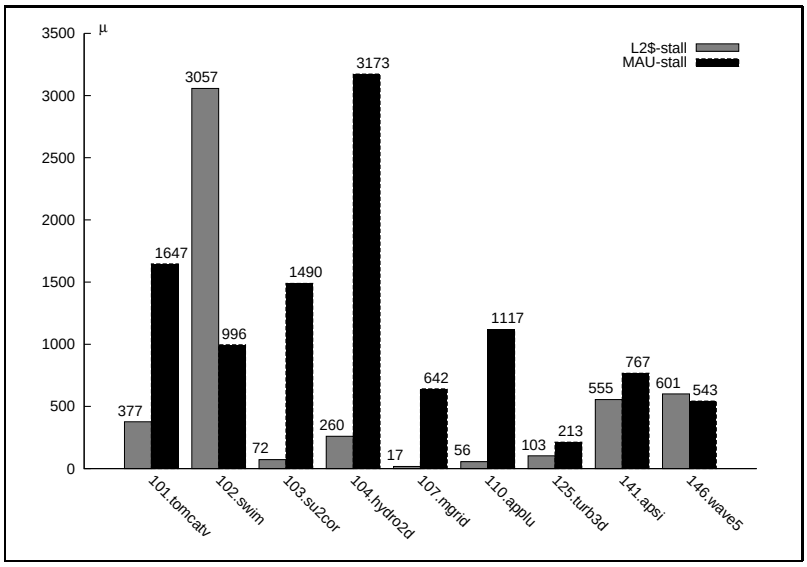

Рис. 1. Внутренние блокировки кэш-памяти данных на задачах пакета SPEC CFP95

ся и формируются очереди запросов по чтению и записи к банкам. Размер каждой очереди ограничен.

Обработка запросов в каждом банке происходит независимо, а результаты выдаются в общий выходной мультиплексор, соединенный с устройством доступа в память. Полная пропускная способность кэша складывается из пропускной способности каждого банка.

Причинами, по которым реальная пропускная способность не достигает теоретического предела, являются вырабатываемые кэшем блокировки, которые могут быть условно разделены на два класса:

1) блокировки работы отдельных банков, вызванные переполнением их очередей, на фоне неполной загруженности других банков;

2) блокировки, причиной которых является занятость устройства доступа в память.

Блокировки первого класса в дальнейшем будем называть внутренними блокировками кэша данных(**L2$-stall**), а блокировки второго класса — блокировками устройства доступа в память (**MAU-stall**).

Изучение причин возникновения блокировок проводилось на вы-

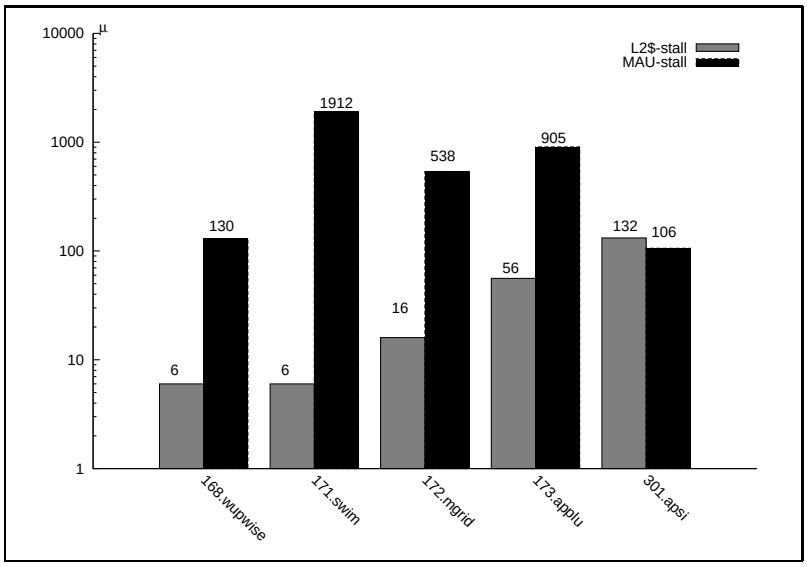

Рис. 2. Внутренние блокировки кэш-памяти данных на задачах пакета SPEC CFP2000

числительном комплексе «Эльбрус-3М1» с помощью инструментальных средств подсчета событий микропроцессора. В качестве единицы измерения было взято количество тактов блокировки на 1000 исполненных широких команд (в дальнейшем μ).

На рис. 1 и 2 представлены результаты исследования внутренних блокировок[1]. Как следует из графиков, блокировки в основном свойственны исполнению приложений, в которых осуществляется работа с большим числом глобальных статических массивов. Во время компиляции программы на этапе редактирования связей адреса глобальных массивов определяются произвольно. Поскольку для данного запроса аппаратура определяет номер банка кэш-памяти, исходя из адреса, то для многобанковых кэшей запросы могут распределяться между банками неравномерно. Именно это вызывает переполнение очередей запросов в одном или нескольких банках на фоне других малоиспользуемых банков.

Еще один фактор, влияющий на производительность, связан с эффективностью использования буфера инструкций. Он предназначен для загрузки инструкций, как по основной ветви выполняемой программы,

[1]Для наглядности данные по пакету SPEC CFP2000 представлены в логарифмической шкале

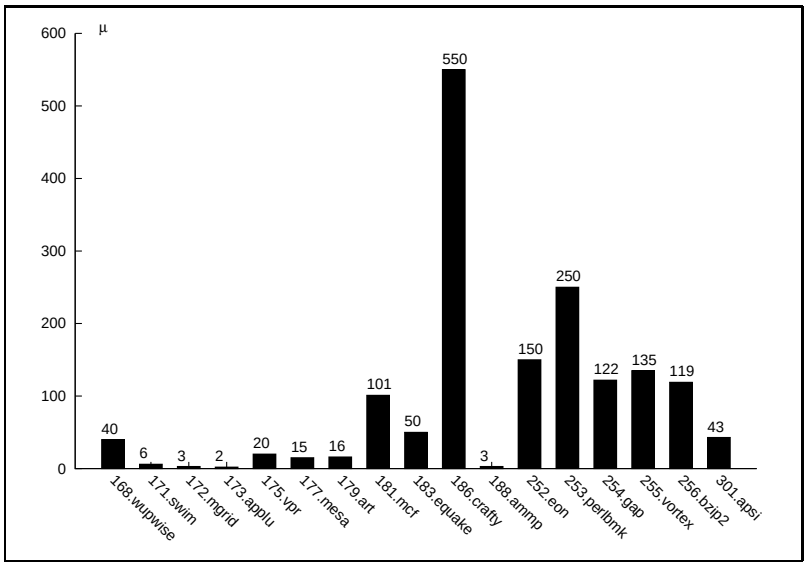

Рис. 3. Количество блокировок по отсутствию кода

так и в направлении переходов, а также — для хранения наиболее часто используемых участков кода. Наличие буфера инструкций позволяет значительно уменьшить время простоя устройства управления и исполнительных конвейеров процессора, а также существенно снижает нагрузку на устройство обращения в память.

Несмотря на наличие этого механизма, блокировки конвейера микропроцессора по отсутствию кода могут возникать на достаточно больших ациклических участках программы, поскольку в таком случае код обладает низкой временно́й локальностью. Эта проблема особенно существенна для ряда системных приложений.

На рис. 3 представлены результаты замеров количества блокировок по отсутствию кода на задачах пакета SPEC CPU2000.

Из графиков следует, что блокировки большей частью возникают при исполнении задач с целочисленными вычислениями, то есть на приложениях системного характера.

Исследование трасс исполнения задач с наибольшим количеством блокировок по ожиданию кода показало, что основная их часть возникает на непрерывных участках, размер которых превосходит размер строки буфера инструкций. Такое их свойство объясняется тем, что аппарат-

ные механизмы загрузки кода не успевают распознать непрерывность исполняемого участка и вовремя послать запрос на подкачку необходимого кода. Соответственно, возникает необходимость в дополнительных методах минимизации таких блокировок, основанных на программной предварительной подкачке.

На основании проведенного исследования в диссертационной работе формулируются задачи, решение которых позволяет решить проблемы повышения производительности.

Вторая глава посвящена разработке методов предварительной подкачки данных. Их эффективность существенно зависит от того насколько точно они соответствуют шаблонам хранения и обращения к данным. Общепринятое разделение шаблонов на два класса (регулярные и нерегулярные), является недостаточным для анализа причин возникновения блокировок и разработки методов их минимизации. Ориентация на эти классы позволяет создать методы, применимые только для небольшого набора шаблонов. В большинстве случаев они ограничиваются подкачкой непосредственно элементов массива или элементов массива, доступных с помощью вторичной индексации. В то же время количество способов доступа гораздо шире. Поэтому в рамках данной работы была разработана более детальная классификация.

В предлагаемой классификации выделяются следующие шаблоны хранения и обращения к данным в циклах:

1. а) 0-*линейно регулярный* — чтение объекта, адрес которого вычисляется с помощью линейной функции от безусловной индуктивной переменной цикла[2] (например, `a[i+1]`);

   б) 0-*нелинейно регулярный* — чтение объекта, адрес которого вычисляется с помощью нелинейной функции от базовой индуктивной переменной цикла (например, `a[i&K]`);

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

в) *n-линейно регулярный* — чтение объекта, адрес которого линейно зависит от результата $(n-1)$-линейно регулярного чтения (например, доступ по вторичным индексам является 1-линейно регулярным `b[a[i+1]+2]`);

г) *n-нелинейно регулярный* — чтение объекта, адрес которого зависит от результата $(n-1)$-нелинейно регулярного чтения, либо нелинейно зависит от результата $(n-1)$-линейно регулярного чтения (например, `b[a[i&K]+1]`);

2. а) 0-*линейно псевдорегулярный* — чтение объекта, адрес которого вычисляется с помощью линейной функции от условной индуктивной переменной[3];

б) 0-*нелинейно псевдорегулярный* — чтение объекта, адрес которого вычисляется с помощью нелинейной функции от условной индуктивной переменной;

в) *n-линейно псевдорегулярный* — чтение объекта, адрес которого линейно зависит от результата $(n-1)$-линейно псевдорегулярного чтения;

г) *n-нелинейно псевдорегулярный* — чтение объекта, адрес которого зависит от результата $(n-1)$-нелинейно псевдорегулярного чтения;

3. 0-*кольцевой рекуррентный* — чтение объекта, адрес которого определяется рекуррентной цепочкой, состоящей только из арифметических инструкций, при этом каждая составляющая адреса полностью определяется своим значением с предыдущей итерации цикла (например, `a+=b; b+=a; c[b];`);

[3]Переменная, изменяющаяся на инвариантную величину при выполнении некоторого условия

4. *n-кольцевой рекуррентный* — чтение объекта, адрес которого определяется рекуррентной цепочкой, состоящей из арифметических инструкций и $n \geq 0$ инструкций чтения, при этом каждая составляющая адреса полностью определяется своим значением с предыдущей итерации цикла (например, `a = c[a+1];`);

Тип 1а является наиболее распространенным способом обращения к данным. Большинство существующих методов подкачки ориентировано именно на обработку такого типа данных.

Типы 1б, 1в, 1г встречаются в реальных приложениях реже, однако достаточно для того, чтобы принять их во внимание при разработке методов предварительной подкачки. В этом качестве особенно интересен тип 1-линейно регулярных данных.

В типе 2 достаточно существенны n-линейно псевдорегулярные данные при $n \geq 0$. Случаи нелинейного доступа ничем не отличается от линейных в плане анализа возможности проведения предварительной подкачки.

Тип 3 также может быть объектом предварительной подкачки, поскольку при задании расстояния подкачки адрес упреждающего запроса вычислим на этапе компиляции. Гораздо сложнее случай n-кольцевого рекуррентного доступа при $n > 0$ (тип 4), так как в таком случае в цепочке инструкций, вычисляющих адрес присутствуют инструкции чтения. Это делает адрес упреждающего запроса невычислимым, а значит нельзя однозначно утверждать, что произойдет подкачка необходимых данных. Тем не менее, предварительная подкачка для данных 4-го типа возможна, однако требует специфических методов подкачки, которые были разработаны в рамках данной работы.

На основании этой классификации предложен базовый алгоритм генерации программ предварительной подкачки, который основывается на методах распознавания данных, классифицируемых 0-линейно регуляр-

ным шаблоном, и реализует программную поддержку комбинированного метода подкачки в рамках оптимизирующего компилятора для микропроцессора «Эльбрус». Его использование привело к повышению производительности в среднем на 21% по пакету SPEC CPU2000.

Несмотря на эффективность базового алгоритма, существуют дополнительные резервы повышения производительности за счет максимального использования аппаратных возможностей. Чтобы увеличить количество обращений, предваряемых подкачкой, в память был разработан алгоритм оптимизации генерируемой программы. Он позволяет подкачивать данные широкими блоками и осуществить замену максимально возможного количества инструкций чтения. Разработанная оптимизация уменьшает общий размер составляющей подкачки коде, в то же время увеличивая количество обращений в память, предваряемых подкачкой, что в итоге приводит к снижению числа блокировок.

Принятие решения о генерации программы предварительной подкачки производится после оценки эффективности оптимизации, поскольку в ряде случаев возникающие накладные расходы на запуск программы могут превысить ее положительный эффект. Оценка эффективности производится в предположении, что данные, к которым обращаются в цикле, находятся в оперативной памяти. Вывод об эффективности применения предварительной подкачки делается на основе следующих характеристик:

1) оценивается время, требуемое для чтения данных в случае применения предварительной подкачки; оно складывается из времени на запуск программы и времени на физическую пересылку данных из оперативной памяти в кэш;

2) оценивается время, требуемое для чтения данных без использования предварительной подкачки; оно вычисляется как произведение

среднего времени выборки строки из оперативной памяти в кэш на количество строк кэша, прочитанных в цикле;

3) дополнительно принимается во внимание тот факт, что использование предварительной подкачки позволяет уменьшить длину аппаратно конвейеризуемого цикла, так как освобождает арифметические устройства от инструкций чтения.

В оценках учитывается среднее количество итераций цикла. В случае, если оценка времени исполнения цикла с использованием программы подкачки оказывается больше времени исполнения цикла без подкачки, на этапе компиляции данный метод не используется.

Важно отметить, что программа предварительной подкачки работает асинхронно, что позволяет лучше адаптироваться ко времени доступа в оперативную память и избежать останова предварительной подкачки в момент обработки полученных данных основным кодом циклического участка.

Использование комплексного метода генерации программы предварительной подкачки позволило в результате в среднем повысить производительность на 24% на задачах пакета SPEC CPU2000.

Построение программы предварительной подкачки возможно для чтений, классифицируемых 0-линейно регулярным шаблоном. При использовании остальных шаблонов типов 1 и 2 это затруднительно, поскольку требует расширенной аппаратной поддержки, однако, в данном случае возможно применение явных инструкций предварительной подкачки. С этой целью был разработан программный метод предварительной подкачки, который основывается на прогнозировании адресов данных, необходимых для последующих вычислений. Он ориентирован на n-(не)линейно псевдорегулярные обращения при $n \geq 0$ и n-(не)линейно регулярные обращения при $n > 0$. Это потребовало создания и реализа-

ции алгоритмов распознавания шаблонов доступа. Кроме того, в методе учитываются архитектурные особенности подсистемы памяти, что позволяет на раннем этапе избежать создания излишних инструкций предварительной подкачки.

Применительно к n-кольцевому рекуррентному шаблону, методы, основанные на учете специфики подсистемы памяти, являются неэффективными, поскольку в этом случае адреса чтений не поддаются статическому предсказанию на этапе компиляции — соответственно, невозможно рассчитать параметры подкачиваемых данных. Поэтому был разработан метод расстановки инструкций предварительной подкачки, базирующийся на профильной информации. Он хорошо применяется к обработке списочных структур данных с нерегулярным расположением элементов в памяти. В этом случае предлагается использовать предварительный запуск программы со сбором профильной информации о расположении адресов чтений с соседних итераций.

Разработанные в данной работе методы программной предварительной подкачки n-(не)линейно (псевдо)регулярных и n-кольцевых рекуррентных данных, дополнительно в среднем повысили на 18,5% и 9,1% показатели производительности задач с соответствующим контекстом.

Предложенная в данной работе классификация шаблонов хранения и обращения к данным и методы предварительной подкачки на ее основе могут быть использованы в оптимизирующем компиляторе для различных универсальных микропроцессоров. Они обеспечивают более точное и эффективное определение данных, подлежащих подкачке.

В третьей главе исследуется проблема блокировок конвейера микропроцессора, возникающих при отсутствии своевременной подачи кода для исполнения. Микропроцессорам с архитектурой широкого командного слова свойственна аппаратная поддержка предикатных вычислений, которая позволяет увеличивать длину непрерывных участков исполне-

ния. В то же время следует отметить, что в результате передачи управления в буфер инструкций загружается код, размер которого не превосходит размер строки буфера инструкций. Кроме того, аппаратным механизмам подкачки кода необходимо время, чтобы распознать непрерывность исполняемого участка. Эти факторы при исполнении непрерывных участков кода могут привести к значительному количеству блокировок.

Проведенный анализ показал, что в задачах пакета SPEC CINT2000 на начальном этапе компиляции имеется в среднем 1,5% непрерывно исполняемых участков кода, ограниченных инструкциями передачи управления, с длиной, превосходящей размер строки буфера инструкции. После применения методов слияния ветвей графа управления в рамках глобального планирования и линеаризация графа управления, которые используются в оптимизирующих компиляторах для поддержки предикатных вычислений, доля таких участков возрастает в среднем до 15,5%, причем, в основном, блокировки по отсутствию кода фиксируются на них.

Решение этой проблемы потребовало разработки специального метода минимизации блокировок, основанного на понятии θ-непрерывно исполняемого участка кода, где θ — минимальная вероятность исполнения инструкции, заканчивающей непрерывный участок. Суть метода состоит в предварительной подкачке кода, исполняемого по прямой ветви. Важно отметить, что подкачке подлежат только ациклические участки кода с достаточно высокой вероятностью исполнения. Кроме того, следует задать эффективное значение расстояния до подкачиваемого кода с тем, чтобы обеспечить его своевременную загрузку. При этом надо учитывать невозможность вычисления этого значения на этапе компиляции. При реализации метода применительно к архитектуре «Эльбрус» была исследована зависимость количества блокировок от параметра θ и других параметров и выбраны их оптимальные значения, соответствующие

основным классам задач. Использование предложенного метода привело к в среднем к повышению производительности на 3,2% на задачах пакета SPEC CINT2000.

В четвертой главе предлагается программный метод, направленный на снижение количества конфликтов, возникающих из-за несоответствия физической организации кэш-памяти специфике обращения к массивам данных в программе. В данном случае общий подход состоит в оптимизации расположения начальных адресов массивов. Однако он не исследован применительно к используемым на данный момент схемам организации кэш-памяти, в частности к многобанковой памяти.

Основной проблемой в многобанковых кэшах является неравномерность запросов в память, обрабатываемых различными банками. Вследствие этого возможно переполнение очередей наиболее интенсивно используемых банков при слабой загруженности других. В итоге аппаратура кэш-памяти посылает сигнал о наличии конфликта, что приводит к блокировке конвейера микропроцессора. Обычно такая проблема характерна для приложений с интенсивной выборкой элементов нескольких различных массивов в циклах.

С учетом того, что блокировки кэша большей частью связаны с исполнением циклов, загрузка банков требует найти такое взаимное расположения массивов данных, чтобы во всех циклах распределение запросов к разным банкам в любой момент времени было близко к равномерному. При этом необходимо учитывать, что изначально установленное во избежание блокировок распределение адресов по банкам будет сохраняться только в случае одинакового темпа изменения адреса. Кроме того, важно обеспечить равномерную загрузку банков в наиболее важных циклах задачи, то есть необходимо дополнительно учитывать профильную информацию. Эти соображения можно представить в аналитической форме. Пусть l – один из множества циклов задачи L, s – определенный шаг из-

менения адреса из множества S шагов и α – набор смещений начальных адресов, определяющий взаимное расположение массивов в памяти. Следует найти минимум по α максимального среднего количества запросов в память использующих j-ый банк кэша на одной итерации цикла $\psi_j^{s,l}(\alpha)$. Тогда, обозначив значение счетчика цикла через $\omega(l)$, количество банков с одинаковым максимальным количеством запросов – $\phi^{s,l}(\alpha)$, получаем задачу минимизации функционала:

$$\sum_{\substack{l\in L\\ s\in S}} \omega(l) \cdot \left(\max_{j=0..n-1} \left(\psi_j^{s,l}(\alpha) + \frac{\phi^{s,l}(\alpha) - 1}{n} \right) \right) \underset{\alpha}{\rightarrow} \min .$$

Точное решение задачи предполагает исследование всех возможных смещений для всех возможных вариантов упорядоченного расположения массивов и аналитически невыразимо. Поэтому для задач пакетов SPEC CFP95 и SPEC CFP2000 были проведены оценки применимости двух методов определения минимума функционала, которые позволяют избежать полного перебора — метода Монте-Карло и метода покоординатного спуска. Они показали, что второй алгоритм при значительно меньшем времени исполнения позволяет в такой же мере сократить количество блокировок.

Использование предложенного программного метода позволило дополнительно в среднем улучшить показатели производительности задач пакета SPEC CFP95 на 10% и пакета SPEC CFP2000 на 5,9%.

Предлагаемая формализация позволяет решить проблему снижения количества конфликтов по банкам для любого микропроцессора с многобанковой кэш-памятью, а алгоритм минимизации функционала позволяет быстро находить эффективное решение для широкого класса задач.

Заключение

В диссертационном исследовании рассматривается проблема эффективного доступа микропроцессора, реализующего парадигму широкого командного слова, к подсистеме памяти. В связи с постоянным увеличением тактовой частоты, количества исполнительных устройств микропроцессора и развитием аппаратных и программных технологий распараллеливания вычислительного процесса этот фактор определяющим образом влияет на общую производительность системы.

Проведенные эксперименты позволили выделить три причины, снижающие производительность. К ним относятся блокировки конвейера микропроцессора по отсутствию данных, кода, и конфликты в структурах многобанковых кэш-памятей.

Для снижения количества блокировок по отсутствию данных были разработаны и реализованы в составе компилятора новые эффективные методы предварительной подкачки данных, основанные на предложенной в работе расширенной классификации шаблонов хранения и обращения к данным. Их использование позволило в среднем повысить производительность на задачах пакета SPEC CPU2000 на 24% для 0-линейно регулярных данных и на 18,5% и 9,1% для n-(не)линейно (псевдо)регулярных и n-кольцевых рекуррентных данных соответственно.

В ходе исследования было установлено, что основная часть блокировок по отсутствию кода возникает при исполнении непрерывных участков, доля которых в среднем на пакете задач SPEC CINT2000 составляет 15,5%. Для решения этой проблемы был разработан новый метод минимизации блокировок, основанный на понятии θ-непрерывно исполняемого участка кода, что позволило в среднем на 3,2% повысить производительность.

Для снижения количества конфликтов кэш-памяти был разработан метод определения оптимального взаимного расположения массивов

данных, базирующийся на изменении начальных адресов. Его использование метода дополнительно улучшило показатели производительности задач пакета SPEC CFP95 в среднем на 10% и пакета SPEC CFP2000 в среднем на 5,9%.

Предложенные методы были реализованы в составе оптимизирующего компилятора для микропроцессора «Эльбрус» и могут быть адаптированы и использоваться для различных микропроцессоров, ориентированных на высокопроизводительные вычисления.

Список работ, опубликованных по теме диссертации

1. Галазин А. Б., Грабежной А. В., Нейман-заде М. И. Оптимизация размещения данных для эффективного исполнения программ на архитектуре с многобанковой кэш-памятью данных // Информационные технологии, № 3, 2008, С. 35-39.

2. Галазин А. Б., Ступаченко Е. В., Шлыков С. Л. Программный метод предварительной подкачки кода в архитектурах со статическим планированием // Программирование № 1, 2008, С. 67-74.

3. Галазин А. Б. Методы оптимизации размещения данных для архитектур с многобанковой кэш-памятью данных // Научные труды XXXIV Международной молодежной научной конференции «Гагаринские чтения», т. 6. М.: МАТИ, 2008, С. 167-168.

4. Галазин А. Б., Степаненков А. М., Ступаченко Е. В. Программная предварительная подкачка кода для микропроцессора Эльбрус-3М // Информационные технологии, № 11, 2007, С. 35-42.

5. Галазин А. Б., Грабежной А. В. Эффективное взаимодействие микропроцессора и подсистемы памяти с использованием асинхронной

предварительной подкачки данных // Информационные технологии, № 5, 2007, С. 26-31.

6. Галазин А. Б. Методы предварительной подкачки в микропроцессоре Эльбрус-3М // Научные труды XXXIII Международной молодежной научной конференции «Гагаринские чтения», т. 6. М.: МАТИ, 2007, С. 213-214.

7. Галазин А. Б. Предварительная подкачка кода для микропроцессора Эльбрус-3М // Материалы докладов XIV Международной научной конференции студентов, аспирантов и молодых ученых «Ломоносов». / Отв. ред. И. А. Алешковский, П. Н. Костылев. [Электронный ресурс] – М.: Издательский центр Факультета журналистики МГУ им. М.В. Ломоносова, 2007. – 1 электрон. опт. диск (CD-ROM).

8. Галазин А. Б. Оптимизация участков кода с малым количеством исполнений // Высокопроизводительные вычислительные системы и микропроцессоры: сборник трудов ИМВС РАН, Выпуск № 9, 2006, С. 58-63.

9. Дроздов А. Ю., Новиков С. В., Боханко А. С., Галазин А. Б. Def-Use граф и методы его использования в современных оптимизирующих компиляторах // Компьютеры в учебном процессе, № 12. 2005. С. 3-14.

10. Дроздов А. Ю., Новиков С. В., Боханко А. С., Галазин А. Б. Глобальный граф потока данных и его роль в проведении оптимизирующих преобразований программ // Высокопроизводительные вычислительные системы и микропроцессоры: сборник трудов ИМВС РАН, Выпуск № 8, 2005, С. 78-87.