К новым компилирующим инструментам разработки быстрых программ.

Штейнберг Б.Я. , Южный федеральный университет

Работа поддержана ОАО «Ангстрем»

Аннотация. Эта статья о требованиях к новым компиляторным инструментам разработки быстрых программ.

Ключевые слова. Параллельные вычисления, система памяти, размещение данных, диалоговая компиляция.

To the New Compiling Instrument for the Creating Fast Programs.

B.J. Steinberg, Southern Federal University.

The paper are supported by “Angstrem” Inc.

Abstract. Some demands for new compiling system of fast programs creating are discuses in this paper.

Key words. Parallel computations, memory system, data placement, dialog compilation.

Кризис быстрых вычислений

Прогресс в технологиях быстрых вычислений определяет развитие многих отраслей экономики: связь, авиа- и машиностроение, космос, безопасность, фармакология, робототехника и др. Несмотря на постоянное увеличение производительности вычислительных систем, потребность в быстрых вычислениях возрастает с каждым годом. Речь идет не только об известных приложениях суперкомпьютеров, входящих в список Top500, но и для вычислений на настольных и мобильных компьютерах. Например, быстрые вычисления нужны финансистам для быстрой оценки изменения ситуации на бирже, для системы автоматического предупреждения столкновений автомобилей, в задачах искусственного интеллекта, решаемых роботами (которые не могут быть оснащены суперкомпьютером) – в таких случаях нужны высокопроизводительные процессоры или небольшие ускорители.

В 2006 г. начался выпуск многоядерных процессоров, и, в данный момент, кроме многоядерных процессоров Интел на рынке присутствует широкий спектр параллельных процессоров и ускорителей: 16-ядерный AMD Opteron6000, 512-ядерный графический ускоритель NVidia-Fermi, 9-ядерный IBM Cell BE, 96-ядерный ускоритель ClearSpeed Advance X620, 64-ядерный процессор Tile64, и другие. Значительный вклад в многообразие высокопроизводительных ускорителей вносят программируемые логические интегральные схемы (ПЛИС). Перечисленные архитектуры отличаются: количеством вычислительных ядер, способом их соединений, видами параллельных вычислений (SIMD, MIMD, Pipeline), системой используемой памяти и другими особенностями. Расширение многообразия вычислительных архитектур приводит к трудностям создания программного обеспечения,

адекватно использующего возможности новых вычислительных архитектур. Трудности разработки эффективного программного обеспечения сдерживают развитие новых вычислительных архитектур. Приближение кризиса, связанного с этой проблемой, отмечается в зарубежных и отечественных публикациях [1, 2].

Отставание оптимизирующих компиляторов от развития вычислительных архитектур легко наблюдается на задаче перемножения матриц. На процессоре Intel Core2duo занимающая несколько строк кода студенческая программа перемножения матриц (умножающая, согласно определению строки одной матрицы на столбцы другой) на задачах большой размерности работает в 60 раз медленнее соответствующей программы библиотеки MKL. Очевидно, что разработка быстрой библиотечной программы требует очень высокой квалификации программиста и большого времени, поскольку при оптимизации использования кэш-памяти разных уровней и SSE команд объем кода такой программы должен быть на два-три порядка больше.

Проблему отставания программного обеспечения от вычислительных систем для быстрых вычислений можно преодолевать двумя путями: развитием образования и созданием новых инструментальных средств разработки эффективных программ. В данной статье пойдет речь о новых чертах инструментальных систем разработки эффективных программ. Следует отметить, что инструмент разработки программ может стать основой создания электронного средства обучения, такого как «Тренажер параллельного программиста» [13].

Имеются работы [31], [32] по одновременному проектированию аппаратуры и ее программы. Этот подход пока используется для разработки программно-аппаратных комплексов. Интересен проект настраиваемого компилятора (retargetable compiler [21], [33]), который на входе требует описание вычислительной архитектуры, но формат описания и оптимизация ориентированы только на узкий класс регистровых процессоров.

В данной работе предлагается создание инструмента разработки эффективных программ, как модификации семейства распараллеливающих компиляторов. Описываются проблемы существующих распараллеливающих компиляторов, формулируются требования к новому инструменту, приводятся примеры и численные эксперименты, демонстрирующие его перспективные преимущества. В качестве прототипа такого будущего инструмента рассматривается ДВОР – диалоговый высокоуровневый оптимизирующий распараллеливатель. Описываемые численные эксперименты реализованы на базе ДВОР.

Конфликт массовой параллельности и памяти.

Время передачи данных с кристалла оперативной памяти на кристалл процессора во много раз больше времени обработки этих данных.

Многие вычислительные алгоритмы имеют следующую схему: часть данных из оперативной памяти перекачивается в память кристалла процессора, обрабатывается, результат перекачивается обратно на кристалл оперативной памяти, затем следующая порция данных закачивается на память кристалла и т.д. Время работы алгоритма определяется двумя характеристиками: 1) временем обработки порции данных; 2) количеством перекачек данных между памятью процессора и оперативной памятью и объемом перекачиваемых порций данных. Такая модель времени вычислений формально описана в [24]. И увеличение объема памяти кристалла процессора и увеличение количества вычислительных ядер могут повышать быстродействие процессора. Чем меньшую площадь кристалла будут занимать вычислительные ядра, тем больше может быть память на этом кристалле и наоборот. Возникает вопрос об оптимальном распределении площади кристалла между памятью и множеством вычислительных устройств (ядер). Оказывается, что для разных алгоритмов

оптимальное распределение площади кристалла разное. Алгоритмы можно разделить на несколько типов [24]:

1. алгоритмы, для которых увеличение объема памяти и количества ядер на кристалле процессора не приводят к увеличению быстродействия (например, скалярное произведение векторов) – время вычисления сводится ко времени перекачки данных с кристалла памяти на кристалл процессора.
2. алгоритмы, для которых существенно увеличение памяти на кристалле процессора (сортировка),
3. алгоритмы, для которых существенно увеличение количества ядер (вычисление степенных рядов),
4. алгоритмы, быстродействие которых зависит от баланса между объемом кэш памяти и количеством вычислительных ядер (перемножение матриц).

Во многих современных оптимизирующих компиляторах имеются функции автоматического распараллеливания, но имеется мало средств, оптимизирующих перекачки данных между кристаллом оперативной памяти и кристаллом процессора.

Оптимизация использования памяти.

Известны некоторые приемы программирования, повышающие быстродействие за счет умелого использования памяти, которые пока не применяются в компиляторах. Эти приемы представляют собой возможный резерв развития компилирующих инструментов автоматизации разработки программ.

Компилятор может взять на себя функции некоторого преобразования кода и такого размещения данных в памяти, при котором минимизируются обмены данными между кристаллами или между элементами внутри кристалла.

Автоматическое выравнивание реализовано во всех компиляторах, поддерживающих автоматическую векторизацию (GCC, Intel, LLVM, MSVC 2012). Некоторые векторные команды (SSE) в процессорах Intel работают только с выровненными данными.

Есть семейство публикаций, направленных на преобразование алгоритмов к блочному виду (Tiling) [5], [26]. Разбиение кода на блоки для широкого класса алгоритмов обработки заполненных матриц может дать ускорение программы в несколько раз.

Матрицы (массивы) в оперативной памяти размещаются компилятором автоматически по столбцам (ФОРТРАН) или по строкам (Си, Паскаль), независимо от исполняемого кода. Для минимизации кэш-промахов в одной и той же программе (например, перемножение матриц) может быть выгодно одну матрицу разместить одним способом, а другую – другим. Более того, для блочных алгоритмов может быть выгодно размещать массивы блоками. Такие возможности реализованы (и показывают ускорение до 40%) в распараллеливающей системе ДВОР [13].

Для некоторых классов прикладных задач можно данные размещать в распределенной памяти с перекрытиями и получать при этом ускорение вычислений [25]. При таком размещении происходит увеличение количества выполняемых арифметических операций, но уменьшается количество пересылок данных, что приводит к увеличению быстродействия. Этот прием может дать ускорение не только при пересылках между процессорами в высокопроизводительном кластере, но и между модулями локальной памяти процессорных элементов, расположенных на одном кристалле (типа Tile64 или Cell BE).

Различные способы размещения массивов в распределенной памяти описаны в [28]. Более общие блочно-аффинные размещения описаны в [29] и реализованы в [13]. Частным случаем блочно-аффинного размещения данных в распределенной памяти является скошенная форма хранения матрицы [4]. Для описания размещений массивов в иерархической распределенной памяти требуется более обобщая модель. Сегодняшние модели размещения данных в

распределенной памяти не учитывают возможности коммуникационной сети.

Внутреннее представление компилирующих систем.

Оптимизация использования памяти не исключает традиционной оптимизации вычислений. Например, если за счет оптимизации последовательных вычислений на одном ядре можно получить ускорение, это означает, что можно использовать меньшее количество процессорных ядер и, следовательно, при проектировании увеличить на кристалле процессора объем памяти. Основа оптимизирующих преобразований программ – внутреннее представление компилирующей системы.

Большинство известных систем автоматической оптимизации и распараллеливания программ имеют внутреннее представление низкого уровня (GCC, LLVM, Intel C++/Fortran Compiler, Microsoft Visual C++, …), близкое к ассемблеру, ориентированное на генерацию команд X86 или близких к ним по уровню. Низкий уровень внутреннего представления удобен для создания оптимизирующих компиляторов с широкого класса языков, даже далеких от Си и ФОРТРАН, таких как Java и Lisp. Но эти внутренние представления не удобны для генерации кода на вычислительные архитектуры, далекие от системы команд X86. Эти системы не используются для генерации кода для высокопроизводительных архитектур нового типа CUDA, Tile64, Мультиклет и др. Также неудобны ассемблерного уровня внутренние представления для генерации HDL-кода (Hardware Description Language) [18] - [20] (C-to-Verilog, Catapult C, Impulse CoDeveloper и др.) и архитектур программируемого конвейера [15] - [17].

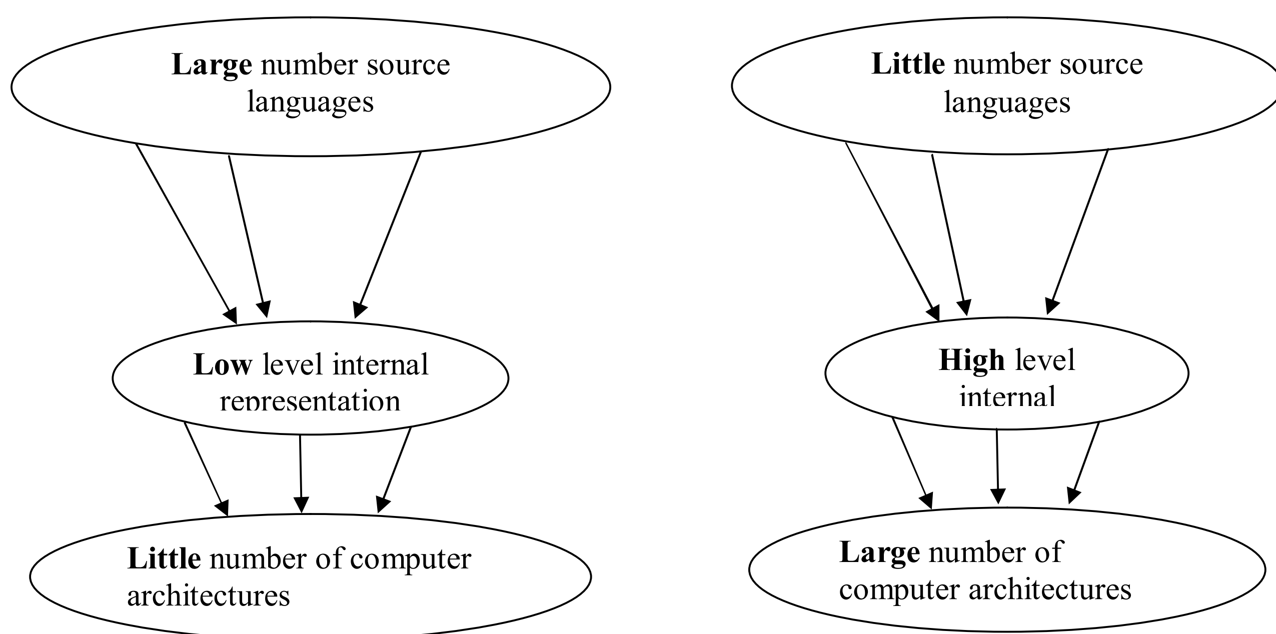

Рис. 1. Сравнение возможностей низкоуровневого и высокоуровневого внутренних представлений.

Высокоуровневое внутреннее представление есть в распараллеливающих системах SUIF, ДВОР [6] и новом развивающемся проекте Rose Compiler [8]. Распараллеливающие компиляторы фирмы PGI (Portland Group Incorporation) [3] основаны на распараллеливающей системе SUIF, и, таким образом, использует высокоуровневое внутреннее представление.

Чтобы сообщения системы пользователю были легко понимаемы, эти сообщения должны использовать обозначения пользователя. Это возможно, если внутреннее представление

близко к исходному тексту программы – еще одно преимущество высокоуровневого внутреннего представления.
Высокоуровневая оптимизация может разрабатываться до создания ассемблера, что позволяет сократить сроки разработки распараллеливающих компиляторов.

Контроль корректности преобразований программ

Анализ информационных зависимостей в программе – основа контроля корректности автоматических преобразований программ. Несмотря на обилие методов определения информационных зависимостей, эти зависимости во многих случаях определяются неточно или долго.

Для ускорения некоторых функций, которые используются при определении информационных зависимостей и требуют больших объемов вычислений, можно использовать их параллельную реализацию. Больших объемов вычислений могут потребовать анализ псевдонимов и анализ зависимостей вхождений многомерных массивов в гнездах циклов большой размерности.

Следует отметить, что анализ информационных зависимостей не гарантирует корректность преобразований программ. Для контроля корректности преобразований следует учитывать и возможные нарушения правил семантики языка [7], [30].

Перспективным выглядит использование в инструментах автоматизированной оптимизации программ технологии решетчатых графов [22], [23], [9], [10], [11], [12]. Решетчатые графы для построения графа информационных связей использует распараллеливающая система ДВОР, а в распараллеливающей системе Pluto на основе решетчатых графов (функции библиотеки PipLib [22]) реализованы преобразования программ. Решетчатые графы удобны для описания информационных зависимостей между точками пространства итераций в гнездах циклов.

Рис. 2. Визуализация элементарного решетчатого графа с (см. окно справа) разбиением пространства итераций на области и линейными функциями, описывающими этот граф в данных областях.

Для анализа зависимостей между вхождениями переменных, лежащих на различных ветках условных операторов, может использоваться символьный анализ.

Есть программы, в которых автоматически невозможно определить факт информационной зависимости между вхождениями переменных. В некоторых таких случаях может помочь диалоговый режим компиляции.

Диалоговый режим компиляции

Диалоговый режим компиляции, в первую очередь, может использоваться для уточнения информационных зависимостей. Рассмотрим пример, в котором информационную зависимость трудно определить из-за внешней переменной (переменная, которая не определяет своего значения внутри цикла) в индексном выражении.

```
    DO 99 I=1,N
99  X(2*I)=X(2*I+k)
```

В этом примере, если число k нечетное, то между обоими вхождениями переменной X нет информационной зависимости; если k четное и отрицательное, то есть истинная информационная зависимость, делающая цикл рекуррентным; если k четное и неотрицательное, то имеет место антизависимость. Если до выполнения программы неизвестно, какое значение примет k к началу выполнения цикла, то и характер зависимости тоже нельзя определить. В этом случае в графе информационных зависимостей должны быть проведены обе дуги между двумя вхождениями переменной X.

Определение диапазонов внешних переменных невозможно выполнить автоматически, поскольку в тексте программы эти диапазоны могут не указываться. Расширять язык программирования возможностью описания таких диапазонов имеет свои недостатки: с одной стороны, для некоторых переменных, особенно, для промежуточных, это выполнить сложно, а, с другой стороны, это нужно не для всех переменных. В таких случаях может быть очень полезен диалоговый режим компиляции. Для уточнения значений переменных диалоговый режим компиляции используется у распараллеливающей системы Parawise [14] и в системе ДВОР [13]. В распараллеливающей системе ДВОР формируется предикат, от истинности которого зависит возможность распараллеливания. Истинность этого предиката определяется по возможности автоматически, или пользователем в противном случае. Предикат создается на основе внутреннего представления программы с помощью символьного анализа.

Преобразование матричных алгоритмов к блочному виду может давать ускорение в несколько раз. Выполнять замену стандартного матричного алгоритма блочным технически возможно, но, строго говоря, компилятор этого делать не может, поскольку при этом меняется порядок выполнения операций, основанный на законе ассоциативности сложения, который не всегда корректен при компьютерных вычислениях. При изменении порядка выполнения сложений в сумме может меняться погрешность, вызванная округлением вещественных чисел в компьютере, и может возникать переполнение. Если программист сам пишет блочный матричный алгоритм, то он берет на себя ответственность за такие возможные явления. Но разработка блочного алгоритма – хлопотное дело – она связана со значительным увеличением объема кода и со сложной индексацией массивов. Здесь может

быть уместен диалоговый режим компиляции, который бы автоматически преобразовал код, но предупредил бы пользователя о возможных проблемах.

Соотношение ручной, диалоговой (полуавтоматической) и автоматической оптимизации (распараллеливания) выглядит следующим образом:
Время выполнения генерируемого кода:
Ручная компиляция < диалоговая компиляция < автоматическая компиляция
Уровень квалификации программиста:
Ручная компиляция > диалоговая компиляция > автоматическая компиляция
Время разработки программы:
Ручная компиляция > диалоговая компиляция > автоматическая компиляция
Множество оптимизируемых и/или распараллеливаемых программ:
Ручная компиляция > диалоговая компиляция > автоматическая компиляция

Кроме уточнения информационных зависимостей, представленные в системах ДВОР и Parawise, и описанного выше изменения порядка выполнения операций (при переходе к блочным алгоритмам), диалог системы с пользователем может дать следующие преимущества по сравнению с автоматическим распараллеливанием:

- Принятие решений о целесообразности преобразований
- Быстрая адаптируемость к новым архитектурам
- Перебор вариантов распараллеливания за счет хранения истории преобразований.

Пакеты переносимых параллельных прикладных программ ($П^5$), ассоциированные с оптимизирующей распараллеливающей системой.

Хорошо известна острая проблема переносимости эффективного программного обеспечения. Так называемые масштабируемые параллельные программы позволяют решать проблему эффективной переносимости в рамках ограниченного класса вычислительных систем. Наличие инструментальной системы разработки оптимизированных программ позволяет некоторым образом решать эту проблему. Опишем этот подход.

Пакеты переносимых параллельных прикладных программ должны быть написаны, как последовательные, но решатели (функции, требующие больших объемов вычислений) этих пакетов должны допускать автоматический анализ и распараллеливание (возможно несколько эквивалентных решателей).

Семейство таких пакетов должно быть ассоциировано с распараллеливающей системой (семейство распараллеливающих компиляторов с общим внутренним представлением и множеством генераторов кода). При переносе пакетов на новые вычислительные архитектуры к распараллеливающей системе дописывается генератор кода, и пакеты перекомпилируются распараллеливающей системой. Этим самым удешевляется адаптация каждого пакета к новой вычислительной архитектуре.

Внутреннее представление такой распараллеливающей системы может быть открытым кодом. Это может расширить группы разработчиков и создать конкуренцию для разработчиков генераторов кода. Библиотеки преобразования программ в этом коде могут быть, как открытыми, так и закрытыми. Закрытость кода может привлечь коммерческих производителей качественного кода, что, в конечном счете, повышает качество пакетов. Кроме того, закрытость генератора кода скрывает архитектуру вычислительной системы, в чем могут быть заинтересованы ее разработчики.
К пакетам могут прилагаться аппаратные (например, на ПЛИС) ускорители решателей.

Существует семейство компиляторов с языка программирования высокого уровня в язык описания электронных схем [18]-[20]. Такого типа компилятор может стать частью компилятора для архитектур программируемых кристаллов. Например, конвертор C2HDL [27], использует внутреннее представление системы ДВОР. Оптимизация использования ПЛИС-ускорителя может выглядеть следующим образом: 1) в программе выделяется гнездо циклов, требующее наибольшего времени выполнения; 2) по наиболее глубоко вложенному циклу гнезда с помощью конвертора C2HDL генерируется VHDL-описание схемы ускорителя; 3) полученная схема «прожигается» на ПЛИС; 4) в гнезде самый вложенный цикл заменяется обращением к ПЛИС-ускорителю. Как и при использовании любого ускорителя в компиляторе должны быть проверки эффективности: компенсирует ли получаемое ускорение вычислений время перекачки данных из управляющего процессора на ускоритель и обратно.

Заключение.

Для разработки высокопроизводительных программ развитие оптимизирующей компиляции может решать следующие задачи:

- уменьшение сроков разработки
- понижение требований к квалификации программистов
- понижение стоимости (в виду предыдущих пунктов)
- повышение надежности (в виду высокоуровневости кода)
- упрощение переносимости на уровне исходного высокоуровневого кода

Высокоуровневые эффективные программы должны быть написаны в стиле, допускающем автоматический анализ, распараллеливание и локализацию кода и данных.

Современные компиляторы недостаточно эффективно отображают высокоуровневые программы на современные архитектуры вычислительных систем. Ручное программирование долго и дорого. Требуется новое поколение компилирующих инструментов для разработки эффективных программ.

В современных компилирующих системах наблюдаются некоторые черты будущих инструментов разработки эффективных параллельных программ: высокоуровневое внутреннее представление, оптимизация использования памяти, диалог с пользователем.

Приведем основные черты инструментальной оптимизирующей компилирующей системы, отличные от современных распараллеливающих компиляторов:

1. диалоговый режим компиляции
2. оптимизация работы с памятью
3. автоматическое (полуавтоматическое) распараллеливание
4. использование решетчатых графов

Распараллеливающая система ДВОР может стать основой для создания нового инструмента разработки программ.

Компьютер без программ – деньги на ветер ☺ .

Литература

1. Mycrost A. Programming Language Design and Analysis motivated by Hardware Evolution (invited Presentation) http://www.cl.cam.ac.uk/~am21/papers/sas07final.pdf дата обращения 11.05.12.
2. Галушкин А.И. Стратегия развития современных суперкомпьютеров на пути к экзафлопным вычислениям. Приложение к журналу «Информационные технологии», №2, 2012 г.
3. http://www.pgroup.com/index.htm — сайт разработчика компиляторов PGI

4. Прангишвили И.В., Виленкин С.Я., Медведев И.Л., Параллельные вычислительные системы с общим управлением// М.: Энергоатомиздат, 1983. – 312 с.
5. Michael. E. Wolf, *Monica*. S. *Lam*. A Data Locality Optimizing. Algorithm. Computer Systems Labo Stanford University, CA 94305
6. Петренко В.В. Внутреннее представление REPRISE распаралелливающей системы. PACO’2008, Труды четвертой международной конференции «Параллельные вычисления и задачи управления», Москва.: 27-29 октября 2008 г.
7. Нис З. Я. Преобразования программ: контроль семантической корректности // Изв. вузов. Сев.-Кавк. регион. Естественные науки. 2010 г., №1. С. 18-21.
8. http://www.cse.ohio-state.edu/~pouchet/software/polyopt/ - PolyOpt/C, a Polyhedral Optimizer for the ROSE compiler
9. Штейнберг Б.Я. Подстановка и переименование индексных переменных в многомерных циклах.// Известия вузов. Северокавказский регион. Юбилейный выпуск. 2002 г., с. 94-99.
10. Штейнберг Б.Я. Открытая распараллеливающая система//Открытые системы, из-во «Открытые системы», 2007 г., № 9, с. 36-41.
11. Штейнберг Б. Я. О взаимосвязи между решетчатым графом программы и графом информационных связей. «Известия ВУЗов. Северокавказский регион. Естественные науки», №5, 2011 г.
12. Савельев В.А., Штейнберг Б. Я. Распараллеливание программ. Ростов-на-Дону, изд-во Южного федерального университета, 2008 г. 192 с.
13. www.ops.rsu.ru
14. http://www.parallelsp.com/parawise.htm
15. Bondalapati K.K. Modeling and Mapping for Dynamically Reconfigurable Hybrid Architectures, Thesis of Philosophy Doctor Dissertation, University of Southern Caroline, 2001.
16. Bondalapati K., Prasanna V. Reconfigurable computing systems, // http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.11.4918 , 2002, 40 p.
17. Каляев А.В., Левин И.И. Модульно-наращиваемые многопроцессорные системы со структурно-процедурной организацией вычислений //М., «Янус-К», 2003, 380 с.
18. http://www.c-to-verilog.com/
19. http://www.mentor.com/esl/catapult/overview/
20. http://www.impulseaccelerated.com/products.htm
21. https://sites.google.com/site/lccretargetablecompiler/
22. http://www.piplib.org/
23. Воеводин В.В. Воеводин Вл.В. Параллельные вычисления, С-Петербург «БХВ-Петербург», 2002, 599 с.
24. Штейнберг Б.Я. Зависимость оптимального распределения площади кристалла процессора между памятью и вычислительными ядрами от алгоритма. PACO’2012/ Труды международной конференции «Параллельные вычисления и задачи управления». М., 24-26 октября 2012 г., ИПУ РАН.
25. Гервич Л.Р., Штейнберг Б.Я. Параллельное итерационное умножение ленточной матрицы на вектор. Труды научной школы И.Б. Симоненко. Сборник статей. Ростов-на-Дону, изд.-во ЮФУ, 2010. с. 58-66.
26. Лиходед Н.А., Соболевский П.И. Разбиение на тайлы итерационных областей алгоритмов, заданных пересечением параллелепипедов. Известия национальной академии наук Блоруссии, серия физико-математических наук 2012 № 1, с. 110.

27. Steinberg B., Abramov A., Alymova E., Baglij A., Guda S., Demin S., Dubrov D., Ivchenko A., Kravchenko E., Makoshenko D., Molotnikov Z., Morilev R., Nis Z., Petrenko V., Povazhnij A., Poluyan S., Skiba I., Suhoverkhov S., Shapovalov V., Steinberg O., Steinberg R. Dialogue-based Optimizing Parallelizing Tool and C2HDL Converter. Proceedings of IEEE East-West Design & Test Symposium (EWDTS'09). Moscow, Russia, September 18-21, 2009 (pp. 216 - 218)
28. Wolfe M. High performance compilers for parallel computing/ Addison-Wesley Publishing Company, 1996. 570 p.
29. Штейнберг Б.Я. Блочно-аффинные размещения данных в параллельной памяти. «Информационные технологии», М.: из-во «Новые технологии», №6, 2010 г., с. 36-41.
30. Штейнберг Б.Я., Алымова Е.В., Баглий А.П., Гуда С.А., Кравченко Е.Н., Морылев Р.И., Нис З.Я., Петренко В.В., Скиба И.С., Шаповалов В.Н., Штейнберг О.Б. Особенности реализации распараллеливающих преобразований программ в ДВОР. ПАСО'2010/ Труды международной конференции «Параллельные вычисления и задачи управления». М., 26-28 октября 2010 г., ИПУ РАН, с.787-854
31. A.Barabanov, M.Bombana, N.Fominykh, A.Terekhov, G.Gorla. Reusable Object for Optimized DSP Design // Embedded Microprocessor Systems, C.M_uller-Schloer et al. (Eds.). _ IOS Press, 1996.
32. D.Boulytchev, O.Medvedev. Hardware Description Language Based on Message Passing and Implicit Pipelining // 7th IEEE East-West Design & Test Symposium (EWDTS). _ 2009.
33. D.Boulytchev, D.Lomov. An Empirical Study of Retargetable Compilers // 4th International Andrei Ershov Memorial Conference on Perspectives of System Informatics. _ 2001.

Штейнберг Борис Яковлевич
Должность: зав. кафедрой Алгебры и дискретной математики
Название организации: Южный федеральный университет
Ученая степень: д.т.н.
Звание: с.н.с.
Печатных работ 90, в т.ч. 2 монографии
Область научных интересов: автоматическое распараллеливание программ.
Рабочий телефон: 8(863)2975114-204
E-mail: borsteinb@mail.ru

Last name: Steinberg
First name: Boris
Patronymic name: Jakovltvich
Position: head of chair "Algebra and discrete mathematics"
The organization name: South federal university
Scientific degree: doctor of science
Rank: senior researcher
90 publishing works, including 2 pornographies
Fields of scientific research: automatic program parallelization
Work phone number: 8(863)2975114-204
E-mail: borsteinb@mail.ru