УДК 681.3

АВТОМАТИЗАЦИЯ РАЗРАБОТКИ ПРОГРАММ ДЛЯ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ С РАСПРЕДЕЛЕННОЙ ПАМЯТЬЮ

Б.Я. Штейнберг
Южный федеральный университет
Россия, 344006, Ростов-на-Дону, Большая Садовая ул., 105/42
E-mail: borsteinb@mail.ru

В работе рассматриваются проблемы автоматического распараллеливания программ. Описываются проекты, ориентированные на решение некоторых из перечисленных проблем.

PROGRAMS AUTOMATIC CREATING FOR PARALLEL COMPUTING SYSTEMS WITH DISTRIBUTED MEMORY / B.Y. Steinberg (Southern Federal University, Russia, 344006, Rostov-on-Don, B.Sadovaja st. 105/42) Problems of a program automatic parallelization are considered in this paper. Two projects oriented to solving some of these problems are described.

Введение

Актуальность темы связана с бурным развитием параллельной вычислительной техники, за которым не поспевают технологии параллельного программирования.

Многообразие и усложнение архитектур создает проблемы разработки нового и переносимости старого параллельного программного обеспечения. Ручное переписывание больших объемов высокопроизводительного программного обеспечения долго и дорого. Поэтому особенно обостряется потребность в разработке новых инструментов автоматизации распараллеливания программ для различных суперкомпьютеров, которые могли бы повысить эффективность использования параллельных компьютеров, понизить требования к квалификации программистов, уменьшить время разработки, повысить надежность и удешевить параллельное программное обеспечение.

Немного истории автоматического распараллеливания.

Мечта о распараллеливающих компиляторах появилась в начале семидесятых годов прошлого века
[54], [55] практически сразу после появления первого параллельного суперкомпьютера ILLIAK-IV. Распараллеливающие компиляторы ставили должны были решить задачу переноса (на уровне исходного кода) накопленных библиотек прикладных программ с последовательных компьютеров на параллельные. По этой тематике были написаны тысячи статей и книг. Было совершено множество попыток написания распараллеливающих компиляторов. Например, в одной из своих книг Владимир Александрович Вальковский приводит обзор [9] разрабатывавшихся в СССР распараллеливающих компиляторов. В этом обзоре 1989 г. представлено 18 распараллеливающих компиляторов! Один из этих компиляторов имеет объем 10000 строк, (6 человеко-лет), а остальные – значительно меньшего объема.

– Наивность без границ! Современные распараллеливающие системы и компиляторы с функциями автоматического распараллеливания (Intel, Microsoft и GCC) имеют объем порядка ста тысяч или даже миллиона строк – и при этом для эффективного переноса старых последовательных программ на новые параллельные компьютеры иногда программу приходится переделывать.

Следует отметить, что успешными являются системы автоматического распараллеливания для вычислительных машин с общей памятью [61], [17], [19], [83], [84]. Для вычислительных машин с распределенной памятью успешной можно отметить только диалоговую систему Parawise [73].

Класс распараллеливаемых программ

Не все программы распараллеливаются автоматически или даже вручную.

Не любую программу можно эффективно распараллелить.

Известны методы распараллеливания некоторых рекуррентных циклов.

Пример 1.
```
FOR I := 1 TO N DO
X(I) := cos(X(I-1)) ;
```

В этом рекуррентном вычислении можно сделать подстановку

```
X(1) := cos(X(0));
FOR I := 1 TO N STEP 2 DO
BEGIN
X(I) := cos(cos(X(I-1)));
X(I+1) := cos(cos(X(I)));
END
```

после чего цикл можно будет разбить на два цикла, которые можно представить как независимые процессы

```
X(1) := cos(X(0));
FOR I := 1 TO N STEP 2 DO
BEGIN
X(I) := cos(cos(X(I-1)));
END
FOR I := 1 TO N STEP 2 DO
BEGIN
X(I+1) := cos(cos(X(I)));
END
```

Эти два процесса можно выполнять независимо, но каждый из них потребует времени столько же, сколько исходный процесс.

Пример 2. Аналогично, в данном цикле с условным оператором подстановка приведет к появлению вложенных условных операторов (и суперпозиций функций)

```
FOR I := 1 TO N DO
IF PI(X(I-1)) THEN  X(I):= FI(X(I-1))
ELSE X(I) := HI(X(I-1)) ;
```

Границы возможностей распараллеливания.

Предположим, что программу удалось распараллелить. Возникает вопрос, достаточно ли эффективно программа распараллелена? Если можно программу распараллеливать, то до какой степени?

Хорошо известно, что вычисление суммы N чисел на p процессорах можно выполнить за $N/p + Log_2 p$ шагов (одновременных сложений) и НЕВОЗМОЖНО выполнить быстрее. Такую теорему несложно доказать. Для каких еще задач (программ) можно получить оценку возможной эффективности распараллеливания? Можно ли получить такую оценку не только в терминах одновременно выполняемых арифметических операций, но и в терминах более долгих операций обращения к памяти? Как доказать, что количество пересылок данных в алгоритме наименьшее? (Автору здесь не известны результаты, кроме [39], [41]) Какие программы можно распараллеливать, а какие – нельзя?

Здесь много важных нерешенных математических задач. С другой стороны, в нашей стране много сильных математиков, которые решают задачи, актуальность которых была определена 30 лет назад. У некоторых таких задач актуальность утрачена и возникает

большая организационная проблема переориентировать (например, с помощью грантов) деятельность ученых высшей квалификации в направление решения востребованных задач.

Проблемы автоматического распараллеливания

4.1. Сохранение синтаксической и семантической корректности программы при применении преобразований

Преобразования программ в высокоуровневом внутреннем представлении приводят к актуальности проверок нового типа: проверок на сохранение преобразованиями синтаксической и семантической корректности [31].

4.2. Проблемы определения информационных зависимостей для линейного класса программ

Определение информационных зависимостей в программе – основа распараллеливающих и оптимизирующих преобразований программ. Известные методы не всегда могут точно автоматически установить информационные зависимости в программе. Подавляющее большинство таких методов ориентировано на так называемый линейный класс программ [14]. Но даже для этого класса программ иногда в принципе невозможно автоматически точно установить зависимость. Рассмотрим следующий пример

Пример 3.

Алгоритм Флойда имеет на входе матрицу весов графа, а на выходе – матрицу кратчайших расстояний. Может быть использован для построения матрицы достижимостей (транзитивного замыкания) графа, для поиска кратчайших путей между всеми парами вершин, для поиска компонент сильной связности ориентированного графа и для построения фактор-графа по компонентам сильной связности.

```
FOR K=1 TO N DO
FOR I=1 TO N DO
FOR J=1 TO N DO
IF A(I,J)>A(I,K)+A(K,J) THEN
A(I,J)=A(I,K)+A(K,J)
```

Граф зависимостей может показать наличие циклически порожденных зависимостей. Следовательно, ни один цикл из данного тройного гнезда не может быть распараллелен.

Символьный анализ может показать, что информационные зависимости реализуются при I=K или J=K. Выполнив символьно любую из этих подстановок, можно увидеть, что логическое выражение условного оператора ложно, если элементы матрицы положительны. Но тогда нет обращения программы к генератору A(I,J)= и зависимостей, на самом деле, нет. Следовательно, распараллеливание циклов со счетчиками I,J возможно.

Но как компилятор для такого анализа может узнать, что все элементы матрицы положительны? Здесь компилятор мог бы уточнить диапазоны данных в диалоге с пользователем, после чего доказать возможность распараллеливания.
Конец примера.

Сложные проблемы определения информационных зависимостей возникают для программ, не принадлежащих линейному классу. Проблемы остаются для тех случаев, в которых:

1) на наличие зависимости влияют внешние переменные,
2) в случае нелинейной зависимости индексных выражений переменных от счетчиков циклов,
3) при наличии внешних переменных в индексных выражениях массивов,
4) наличие псевдонимов

4.3. Проблемы автоматизации распараллеливания на вычислительные системы с распределенной памятью

Размещение массивов в распределенной памяти с оптимизацией затрат на межпроцессорные пересылки. Минимальные затраты на пересылки реализуются при вычислениях на одном процессоре. Минимизировать следует суммарное время пересылок и вычислений. Следует отметить, что время пересылок зависит от коммуникационного устройства [57], [7], [26] и размещения данных в параллельной памяти. Здесь же возникает задача оптимальных переразмещений данных при различных способах организации распределенной памяти.

4.4. Проблема разработки новых эффективно распараллеливаемых алгоритмов.

Для решения одной и той же задачи на различных вычислительных архитектурах могут быть эффективны различные алгоритмы. Рассмотрим, например, задачу перемножения матриц размерности порядка 100-1000. Для процессора, у которого время выполнения умножения чисел дольше времени обращения к памяти (70-ые годы) эффективен алгоритм Штрассена. Для нынешних процессоров стандартный алгоритм перемножения матриц значительно эффективней [63].

Таким образом, возникает проблема для одной и той же задачи разрабатывать различные алгоритмы для различных вычислительных архитектур: MIMD, SIMD, Reconfigurable pipeline…; и с различными способами организации памяти: общая, распределенная, многоуровневая…; с различными коммуникационными системами: кольцо, решетка, гиперкуб, дерево…

4.5. Проблема распознавания алгоритма.

Предположим, что следует распараллелить программу перемножения матриц, в основе которой лежит алгоритм Штрассена. Такая программа плохо распараллеливается. Было бы правильно заменить ее программой, в основе которой стандартный алгоритм перемножения матриц, который затем с успехом распараллеливать. Проблема в том, чтобы по тексту программы компилятор мог понять, что эта программа перемножает матрицы.

4.6. Максимальная общность преобразований и формализация эквивалентности

Теория (эквивалентных) преобразований программ.

Блок оптимизации и распараллеливания программ – большая по объему и используемому интеллекту часть компилятора. Возникает естественное желание использовать одну и ту же библиотеку преобразований для программ разных языков. Это приводит к необходимости формально описать класс языков, к программам которых конкретное преобразование можно применить, и сформулировать условия, при которых такое применение приводит к программе, эквивалентной исходной. Здесь придется определять понятие языка программирования, исполнения программы и пр.

Пример 4. Понятие языка – четко определено. А программа – что это такое? Что такое «исполнение программы»? Например, тексты на языке TeX после исполнения создают

изображение на экране. Какие два текста на языке TeX эквивалентны? Язык TeX является языком программирования? Можно ли говорить об эквивалентных преобразованиях текстов языка TeX?

Утопичность идеи автоматического эффективного переноса всех старых последовательных программ на параллельные компьютеры

Итак, упомянутые выше проблемы автоматического распараллеливания, границы возможностей распараллеливания и примеры вообще нераспараллеливаемых программ приводят к выводу о несостоятельности мечты программистов семидесятых об автоматическом переносе последовательных программ на различные параллельные компьютеры.

Кстати, существуют наборы программ (бенчмарки) для тестирования производительности компиляторов и компьютеров. Здесь тоже не все в порядке – тестировать, видимо, следует на спектре программ для каждой задачи… [78].

Автоматическое распараллеливание – основа инструментальных средств разработки параллельных программ.

Ясно, что чем больше автоматизации при разработке параллельных программ – тем лучше. Здесь возможны два пути развития: от средств автоматического распараллеливания к инструментальным системам (системы диалогового распараллеливания) и внедрение в средства ручной разработки параллельных программ средств автоматического распараллеливания.

В Южном федеральном университете рассматриваются оба подхода. Первый подход воплощен в проекте «Диалоговый высокоуровневый оптимизирующий распараллеливатель программ и его приложения», а второй – в проекте DistribF – расширение языка ФОРТРАН операторами параллельного программирования, основанными на автоматическом распараллеливании.

Проект «ДВОР и его приложения».

ДВОР – это диалоговый оптимизирующий распараллеливатель программ, разрабатываемый на мехмате Южного федерального университета. Эта распараллеливающая система вобрала в себя опыт исследовательской Открытой распараллеливающей системы [62] и предназначена для распараллеливания программ с процедурных языков (в данный момент реализованы пасеры языков Си и ФОРТРАН). В основе ДВОР лежит высокоуровневое внутренне представление, позволяющее генерировать код на различные вычислительные архитектуры. Высокий уровень внутреннего представления также удобен для диалоговых преобразований программ (в преобразованной программе можно узначать исходную). Одновременно с распараллеливателем разрабатываются «приложения»: библиотеки программ по финансовой математике, криптографии и ACELAN (по механике и пьезокерамике). Библиотеки разрабатываются последовательными, но с учетом эффективного автоматического анализа и распараллеливания с помощью ДВОР.

Распараллеливание циклов на различные вычислительные архитектуры.

Под автоматическим распараллеливанием программ понимают преобразование последовательной программы к параллельной. Распараллеливание программ может проводиться на разных уровнях: на уровне функций, циклов, линейных участков, микрокоманд. Способ распараллеливания зависит от архитектуры вычислительной системы. Более всего при автоматическом распараллеливании рассматривается распараллеливание циклов: параллельное выполнение цикла – это одновременное выполнение его итераций. Это определение не корректно, поскольку не учитывает параллельную вычислительную архитектуру. На разные архитектуры могут распараллеливаться разные циклы.

Распараллеливанию циклов могут мешать программные зависимости. Условия распараллеливаемости циклов формулируются в терминах программных зависимостей.

Асинхронная архитектура позволяет распараллеливать циклы любой глубины вложенности. Эти циклы могут содержать условные операторы. Но в этих циклах допускаются только те потоковые, выходные и анти- зависимости, которые являются циклически независимыми. Циклически порожденные зависимости требуют дополнительных операторов синхронизации.

Синхронная архитектура SIMD позволяет распараллеливать циклы с фиксированным количеством итераций, содержащие другие такие циклы и операторы присваивания. Допускаются потоковые, анти- и выходные зависимости, которые на графе информационных связей направлены сверху-вниз (от вхождений, которые раньше обращаются к памяти, к вхождениям, которые позже обращаются к памяти). Недопустима выходная самозависимость (генератор, не зависящий от счетчика цикла). Такая архитектура может непосредственно вычислять циклы лишь с некоторыми простыми условными операторами, с использованием маски и с потерей производительности.

Общая память допускает входные зависимости (и для SIMD и для MIMD). Входная зависимость означает, что разные итерации цикла используют одни и те же данные. При распределенной памяти входная зависимость мешает, но может быть устранена предварительной рассылкой данных.

Специальные преобразования программ могут приводить нераспараллеливаемые циклы к распараллеливаемому виду, этим самым значительно расширяя класс распараллеливаемых программ.

В системе ДВОР ведутся работы по автоматической генерации текстов на языках Си и ФОРТРАН с вызовами функции библиотеки MPI и прагмами OpenMP. Рассматривается вопрос об автоматической генерации параллельного кода архитектур SIMD.

Использование решетчатых графов – перспектива скачка в развитии автоматического распараллеливания.

Решетчатые графы – модель информационной зависимости между точками пространства итераций гнезда циклов.

1. Построение решетчатых графов для циклов с внешними переменными.
2. Перспективы оптимизаций на основе решетчатых графов:
3. SSA-форма для массивов,
4. Символьный анализ
5. Подстановка и переименование индексных переменных
6. Протягивание константных значений массивов

7. Удаление мертвого и полумертвого кода
8. РАСПАРАЛЛЕЛИВАНИЕ ГНЕЗД ЦИКЛОВ

В проекте ДВОР реализована программа построения решетчатых графов для гнезд циклов, причем эти гнезда могут содержать внешние переменные.

Библиотека преобразований в ДВОР.

Библиотека оптимизирующих и распараллеливающих преобразований программ – это необязательная часть в компилирующей системе, как и всякая оптимизация. Но, поскольку распараллеливание преследует единственную цель – повышение быстродействия, такая библиотека становится необходимой.

Библиотека оптимизирующих/распараллеливающих преобразований характеризуется с одной стороны, набором преобразований, а. с другой стороны, широтой класса программ, к которым эти преобразования применимы. И то и другое расширяется во время жизненного цикла системы.

Можно выделить следующие отличия проекта библиотеки преобразований ДВОР от других систем автоматического распараллеливания:

1. Наличие преобразований, распараллеливающих рекуррентные циклы.
2. Наличие преобразований, использующих решетчатые графы
3. Использование диалоговой формы преобразований
4. Использование SSA-формы массивов
5. Использование специальных проверок применимости преобразований
6. Использование специальной системы тестирования преобразований

Система автоматизации тестирования преобразований программ.

Эта система предназначена для проверки корректности преобразований программ в системе ДВОР. Тестирование проводится в два этапа.

1) По конфигурационным файлам преобразований генерируются тестовые программы, к которым применяются преобразования.

2) Если преобразование выполнилось, то появляется задача проверить, эквивалентна ли результирующая программа исходной (тестовой).

На первом этапе работает генератор тестовых программ, использующий грамматику входного языка, конфигурационные файлы тестируемого преобразования и генератор случайных чисел [69].

На втором этапе используется специальная система автоматической проверки эквивалентности программ [74], [73]. Эта система может использоваться и для проверки корректности и масштабируемости параллельных программ.

Преимущества диалогового распараллеливания по сравнению с автоматическим..

Диалоговое распараллеливание по сравнению с автоматическим, с одной стороны, дольше генерирует параллельный код и требует более высокой квалификации пользователя. Но, с другой стороны, диалоговое распараллеливание значительно дешевле и в диалоге можно получить более эффективный код, чем при автоматической компиляции.

Поясним эти преимущества.

Диалоговый распараллеливатель дешевле автоматического, поскольку в нем могут отсутствовать дорогие автоматические оценки целесообразности преобразований (применять преобразование или нет, решает пользователь). Эти оценки очень громоздки, поскольку очень сильно зависят от контекста преобразуемого фрагмента программы. Зависимость от архитектуры не позволяет такие оценки переносить их с компилятора для одного компьютера на другой. В частности, отсюда вытекает что, диалоговая система может быстрее адаптироваться к новым архитектурам.

Более высокое качество кода реализуется за счет возможности пересмотреть несколько вариантов оптимизации и выбрать лучший.

В диалоге можно уточнять информационные зависимости на основе знаний пользователя о диапазонах значений данных. В частности, для приведенного выше примера не распараллеливаемого автоматически алгоритма Флойда возможна диалоговая распараллеливающая компиляция.

При диалоговой компиляции возможно изменение порядка выполнения ассоциативных арифметических операций, что может привести к повышению точности вычислений.

Концепция переносимости параллельного ПО.

Эффективному автоматическому распараллеливанию поддается небольшое множество последовательных программ. Автоматически распараллелить накопленные человечеством последовательные программы – неосуществимая мечта, поскольку большинство последовательных программ написаны в стиле, не поддающемуся автоматическому анализу и распараллеливанию. Другое дело – распараллеливать специально для этого написанные программы.

В данном проекте реализуется следующая идея: разработать систему автоматизированного распараллеливания программ ДВОР и разработать библиотеки прикладных программ так, чтобы они допускали эффективный автоматический анализ и распараллеливание этой распараллеливающей системой. Такой подход позволит при появлении новой параллельной вычислительной архитектуры дописать к ДВОР соответствующий генератор кода и перекомпилировать все библиотеки прикладных программ. Это создает новый подход к решению проблемы переносимости параллельных программ.

ДВОР разрабатывается одновременно с библиотеками программ криптографии, финансовой математики и с модификацией библиотеки АСЕЛАН для моделирования задач механики и пьезокерамики.

Проект DistribF – расширение языка ФОРТРАН для распределенных вычислительных систем.

Расширение должно состоять из следующих элементов:

- Блочно-аффинные размещения данных в распределенной памяти.
- Библиотека переразмещения данных.
- Операторы параллельного выполнения циклов и гнезд циклов.

Блочно-аффинные размещения данных.

Пусть p – количество модулей памяти, натуральные числа $d_1, d_2, \ldots, d_m$ и целые константы $s_0, s_1, s_2, \ldots, s_m$ зависят только от m-мерного массива X. Блочно-аффинное по модулю p размещение m-мерного массива X – это такое размещение, при котором элемент $X(i_1,i_2,\ldots,i_m)$ находится в модуле памяти с номером

$$u = (]i_1/d_1[* s_1 +]i_2/d_2[* s_2 + \ldots +]i_m/d_m[* s_m + s_0) \bmod p.$$

Здесь]x[- наименьшее целое, которое не меньше x.

Блочно-аффинное размещение данных задается набором целочисленных параметров, варьируя которыми, можно подобрать оптимальное размещение данных для выполнения разрабатываемой программы. Блочно-аффинные размещения данных допускают, с одной стороны, простое описание, а, с другой стороны, обобщают большинство используемых способов размещения одномерных и двумерных массивов на многомерный случай. Рассматриваемый подход к размещению данных восходит к работе [40] и описан в работах [64], [72]. Блочно-аффинные размещения данных содержат в себе блочные размещения, размещения матриц «по строкам», «по столбцам» или в «скошенной форме» [33], размещения, допускаемые HPF [85].

Библиотека переразмещения данных.

Библиотека переразмещений данных должна содержать функции, преобразующие массив, размещенный одним блочно-аффинным способом в массив, размещенный другим способом. В частности, эта библиотека должна содержать транспонирование и скашивание матриц, размещенных в параллельной памяти, и обобщение этих операций для многомерных массивов. Данный подход к размещению данных восходит к работам [39], [41].

Оператор параллельного выполнения гнезда циклов

Операторы параллельного выполнения гнезд циклов предполагают параллельное выполнение процессов, соответствующих точкам пространства итераций гнезда циклов. Например, в гнезде циклов

```
DO 99 I = 1, N
DO 99 J = 1, N
X[I,J] = X[I-1,J]+X[I,J-1]
```

не может быть параллельно выполнен ни один из двух циклов, в то время, как после выполнения (I,J) = (1,1) возможно одновременное выполнение (1,2) и (2,1) и т.д. [55]. Возможна и другая параллельная реализация этого гнезда циклов, в зависимости от размещения данных [47]. Параллельное выполнение циклов – частный случай параллельного выполнения гнезда циклов. Оператор параллельного выполнения гнезда циклов предполагает автоматический анализ программных зависимостей, аналогичных приведенным в примере, и сам генерирует параллельно выполняемые процессы. В основе такого автоматического анализа лежит теория решетчатых графов, развитая в работах В.В. Воеводина, P. Feautrier и др. [5], [14], [13], [52], [53], [51], [50], [8], [30], [46], [43], [45], [44], [71]. Следует отметить предшествующие работы по распараллеливанию гнезд циклов, неявно (в качестве иллюстраций) использующие решетчатые графы [55], [54], [11], [10], [6], и др.

Реализация параллельного выполнения гнезд циклов для векторных компьютеров или, более общо, для вычислительных архитектур SIMD [33], [22], [77], возможна на основе метода гиперплоскостей [55]. Метод, изложенный в работах [65], [66], [47] может быть использован для распараллеливания гнезд циклов на многоконвейерные или MIMD архитектуры (см., например, [29], [34] и др.).

Перспективной для эффективного отображения циклов и гнезд циклов представляются программируемые (перестраиваемые) архитектуры [48], [49], [28], [27], [82].

ДОПОЛНЕНИЕ. Особенности распараллеливания программ на вычислительные системы с распределенной памятью.

Классическая теория алгоритмической сложности рассматривает сложность алгоритма: количество арифметических операций, как функцию от количества входных данных. Такая модель адекватно отражала вычисления на компьютере в те времена, когда эта теория зарождалась [56]. Но, с некоторого момента развития вычислительной техники, самой длительной операцией стало обращение к памяти (запись, чтение). Оптимизации обращений к памяти посвящена работа [23]. Если скорость обработки данных возрастает ежегодно на 40%, то скорость подготовки данных (обращений к памяти) возрастает на 9% [76]. Эта проблема приводит к необходимости использования распределенной памяти, которая позволяет параллельно выполнять операции чтения и записи. При этом, «распределенная память» может означать не только наличие у каждого процессора своей локальной памяти, но и наличие у каждого процессорного ядра на кристалле тоже своей памяти.

О соотношении времени обработки данных ко времени подготовки данных

Чем больше процессорных элементов (ПЭ) - тем, с одной стороны, меньше тактов с арифметическими операциями, но, с другой стороны, больше пересылок данных. Это подтверждает и эксперимент, проведенный на ВЦ Ростовского госуниверситета в 1989 году на ЭВМ ПС-2000 [22]: с использованием различных количеств ПЭ решалась задача вычисления суммы 128 целых чисел, и при этом измерялось время счета. В результате эксперимента выяснилось, что на 16 ПЭ программа работает вдвое быстрее, чем на 64 (на 32 или 8 ПЭ чуть дольше, чем на одном). Эти соображения приводят к задаче об оптимальном количестве процессоров. Тот факт, что из-за пересылок данных распараллеливание может привести не к ускорению, а к замедлению времени счета в задаче вычисления суммы N чисел отмечался и ранее, например, в [**81**]. Однако вопрос об оптимальном количестве ПЭ не ставился. Задача оптимального использования ресурсов при параллельных вычислениях рассматривалась в [**79**]. Но в этой работе программы представляются в виде графовой модели [5]. В частности, рекуррентные циклы по такой методике должны выполняться в одном ПЭ.

В работах [3], [**80**], получены формулы оптимального количества процессорных элементов для вычисления суммы элементов массива с учетом пересылок данных на МВС. Эти формулы обобщаются на некоторый класс программ, допускающий циклы с линейной рекуррентной зависимостью и с операторами присваивания. Оптимальное количество

процессорных элементов зависит от соотношения времени выполнения арифметических операций и времени подготовки данных.

Блочная организация данных.

Использование распределенной памяти создает известную проблему межпроцессорных пересылок – операций, которые по длительности значительно превосходят арифметические операции и операции обращения к памяти. Возникает задача такого размещения данных в параллельной памяти, при которой соотношение пересылок данных и одновременно выполняемых арифметических операций будет сбалансировано и будет минимизировать общее время выполнения программы.

Архитектуры современных вычислительных систем, как правило, представляют собой иерархию блоков: 1) на одном кристалле располагается множество ядер, 2) на одной плате располагается множество кристаллов, 3) в одном корпусе располагается несколько плат, 4) в одном шкафу располагается корпусов... При этом внутри кристалла пересылки данных требуют одно время, внутри платы – второе, внутри корпуса – третье… Таким образом, задачу естественно разбивать на относительно слабо связанные части, каждая из которых считается в своем кристалле (плате, корпусе,…). Еще более это существенно для данных, пересылки которых особенно дорого стоят. Эти рассуждения приводят к блочной организации данных [18]. Приведем примеры, иллюстрирующие преимущества блочного размещения данных.

Распознавание двумерных образов по шаблону.

Основная (самая долгая) часть алгоритма распознавания двумерных образов по шаблону выглядит примерно так же, как двойной цикл в итерационном алгоритме решения задачи Дирихле в прямоугольнике.

```
      do 99 i = 1, n
      do 99 j = 1, n
      S = 0
      do 88 i1 = 1, m1
      do 88 j1 = 1, m2
88    S = S + A(i – m1/2 + i1, j – m2/2 + j1)*V(i1,j1)
99    A(i,j) = S
```

Размещение матрицы A в параллельной памяти для этой задачи так же, как и для задачи Дирихле, выгоднее выполнять поблочно, чем по столбцам или по строкам. Параллельно выполняется внешнее гнездо из двух циклов.

Решение данной задачи распознавания на вычислительной модели с кэш-памятью имеет такой же характер, как и задача Дирихле и блочное размещение здесь также предпочтительно.

Сеточные методы решения трехмерных задач.

Схема решения сформулированных задач при семиточечном шаблоне выглядит примерно следующим образом:

```
      do 99 k = 1, n
      do 99 i = 1, n
      do 99 j = 1, n
99    A(i,j,k) = A(i-1,j,k) + A(i+1,j,k) + A(i,j-1,k)+ A(i,j+1,k) + A(i,j,k-1)+ A(i,j,k+1)
```

Если в схеме участвуют диагональные элементы трехмерной сетки, то они не внесут изменений в вопрос об оптимальном считывании данных.

Пусть количество процессоров $p = n$.

Разместим трехмерный массив A в параллельной памяти поблочно с размерами блоков $p^{1/3}$. Количество пересылочных тактов равно $6*p^{1/3}$.

Если массив разместить так, что элемент $A(i,j,k)$ находится в модуле памяти с номером $i \bmod p$, то количество пересылочных тактов $2*p$. Таким образом, блочное размещение имеет очевидные преимущества.

Рассмотрим решение данной трехмерной задачи на вычислительной модели с кэш-памятью.

Предположим, что массив A считывается параллелепипедами со сторонами $M1$, $M2$, $M3$ ($M1*M2*M3 = M$). Тогда на считывание M данных приходится вычислений тела цикла

$$\text{ValumeCulc} = (M1 - 2)*(M2 - 2)*(M3 - 2)$$
$$= M - 2*(M1*M2 + M2*M3 + M1*M3) + 4*(M1 + M2 + M3) - 8$$

Несложно установить, что данная функция достигает оптимального значения $(M^{1/3} - 2)^3$ при равенстве всех сторон параллелепипеда $M1 = M2 = M3 = M^{1/3}$.

Количество чтений памяти равно $n^3/(M^{1/3} - 2)^3$.

Если выполнять программу непосредственно, как она записана тройным циклом, то будут считываться строки (i,j), $(i-1,j)$, $(i+1,j)$, $(i,j-1)$, $(i,j+1)$ по $M/5$ чисел. То есть

$$\text{ValumeCulc} = M/5 - 2.$$

Количество чтений памяти равно $R = n^3/(M/5 - 2) \approx 5*n^3/M$.

Выигрыш при блочном размещении данных асимптотически (при больших M) в 5 раз.

Блочные размещения данных для решения СЛАУ с ленточной матрицей.

Рассмотрим СЛАУ с ленточными матрицами, решение которых методами редукции и окаймления приводит к СЛАУ с блочными трехдиагональными матрицами.

Любая ленточная матрица может быть разбита на блоки так, что она станет блочной трехдиагональной матрицей.

$$\left(\begin{array}{cc|cc|cc|cc|c}
1 & d_1 & & & & & & & h_1 \\
b_2 & 1 & d_2 & & & & & & h_2 \\
\hline
c_3 & b_3 & 1 & d_3 & & & & & h_3 \\
 & c_4 & b_4 & 1 & d_4 & & & & h_4 \\
\hline
 & & c_5 & b_5 & 1 & d_5 & & & h_5 \\
 & & & c_6 & b_6 & 1 & & & h_6 \\
\hline
. & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . \\
\hline
 & & & & & & c_{n-1} & b_{n-1} & h_{n-1} \\
 & & & & & & & c_n & h_n
\end{array}\right)$$

Имеет место следующий очевидный факт.

Пусть для матрицы $A = \{a_{ij}\}_{i,j=1,\ldots,n}$ m – такое наименьшее число, что $a_{ij} = 0$, если $|i-j| > m$. Тогда для того, чтобы матрица A после разбиения на блоки стала блочной трехдиагональной, необходимо и достаточно, чтобы размер блоков был не меньше m.

Теперь к СЛАУ с блочной трехдиагональной матрицей может быть применен метод редукции, который, в частности, допускает параллельное выполнение [1], [2]. В случае треугольной ленточной СЛАУ получается блочная двухдиагональная матрица. Решение СЛАУ с такой матрицей равносильно вычислению цикла с линейной рекуррентной зависимостью. Параллельные алгоритмы вычисления таких циклов приводятся в [3], [4]. При размещении блочной матрицы в параллельной памяти, очевидно, для нормального выполнения указанных алгоритмов все элементы каждого блока должны находиться в одном и том же модуле памяти. Следовательно, для параллельных алгоритмов решения СЛАУ с такими матрицами естественным является блочное размещение данных.

Параллельное вычисление дробно-линейных рекуррентных последовательностей.

Сначала следует вспомнить, что суперпозиция двух дробнолинейных отображений опять является дробнолинейным отображением.

$$g_1(x) = \frac{a_1 * x + b_1}{c_1 * x + d_1}; \quad g_2(x) = \frac{a_2 * x + b_2}{c_2 * x + d_2}$$

$$f(x) = g_2(g_1(x)) = \frac{(a_2 * a_1 + b_2 * c_1) * x + (a_2 * b_1 + b_2 * d_1)}{(c_2 * a_1 + d_2 * c_1) * x + (c_2 * b_1 + d_2 * d_1)}$$

Если коэффициенты всякого дробнолинейного отображения записывать в виде матрицы 2x2, то коэффициенты суперпозиции отображений представляют собой произведение матриц коэффициентов исходных отображений.

Пусть задана рекуррентная последовательность

$$x_n = f_{n-1}(x_{n-1}) = \frac{a_{n-1} * x_{n-1} + b_{n-1}}{c_{n-1} * x_{n-1} + d_{n-1}}$$

Обозначим матрицу коэффициентов отображения f_n

$$A_{n-1} = \begin{pmatrix} a_{n-1} & b_{n-1} \\ c_{n-1} & d_{n-1} \end{pmatrix}$$

Тогда для вычисления членов последовательности x_n достаточно вычислить все произведения матриц

$B_1 = A_1$
$B_2 = A_1 * A_2$
$B_3 = A_1 * A_2 * A_3$
……
$B_{n-1} = A_1 * A_2 * A_3 * \ldots * A_{n-1}$

(Параллельный алгоритм вычисления всех таких произведений можно найти в [5]). Теперь все члены последовательности x_k могут быть одновременно вычислены по элементам матриц

$$B_k = \begin{pmatrix} a'_k & b'_k \\ c'_k & d'_k \end{pmatrix}$$

по формулам

$$x_k = \frac{a'_k * x_0 + b'_k}{c'_k * x_0 + d'_k}$$

Ясно, что при параллельном вычислении дробнолинейной рекуррентной последовательности на компьютере с параллельной памятью каждая матрица коэффициентов каждой дробнолинейной формулы должна быть целиком в одном и том же модуле памяти. Поэтому для данного алгоритма также естественным является блочное размещение данных.

ЛИТЕРАТУРА:

1. Фаддеева В.Н., Фаддеев Д.К. Параллельные вычисления в линейной алгебре. 1.// Кибернетика.- 1977.- N 6, с. 28-40.
2. Фаддеев Д.К., Фаддеева В.Н. Параллельные вычисления в линейной алгебре. 2. – Кибернетика, 1982, № 3, с. 18-31, 44.
3. Штейнберг Б.Я. Математические методы распараллеливания рекуррентных программных циклов на суперкомпьютеры с параллельной памятью.// Ростов-на-Дону, Издательство Ростовского университета, 2004 г., 192 с.
4. Штейнберг О.Б. Автоматическое распараллеливание рекуррентных циклов с нерегулярным вычислением суперпозиций. PACO'2008, Труды четвертой международной конференции «Параллельные вычисления и задачи управления», Москва.: 27-29 октября 2008 г. с.
5. Воеводин В.В. Математические модели и методы в параллельных процессах// М.: Наука, гл. ред. физ.-мат. лит., 1986, 296 с.
6. Бабичев А.В., Лебедев В.Г. Распараллеливание программных циклов./ Программирование// 1983, N 5, с. 52-63.
7. Бенеш В.Э. Математические основы теории телефонных сообщений.- М.: «Связь», 1968, 292 с.
8. Гуда С.А. Оценки длины критического пути в решетчатом графе. PACO'2008, Труды четвертой международной конференции «Параллельные вычисления и задачи управления», Москва.: 27-29 октября 2008 г. с.
9. Вальковский В.А., Распараллеливание алгоритмов и программ. Структурный подход/ М., «Радио и связь», 1989 г., 176 с.
10. Вальковский В.А. Параллельное выполнение циклов. Метод пирамид. - Кибернетика. 1983, N 5. с. 51-55.
11. Вальковский В.А. Параллельное выполнение циклов. Метод параллелепипедов. - Кибернетика. 1982, N 2. с. 51-62.
12. Векторизация программ. // Векторизация программ: теория, методы, реализация. / Сборник переводов статей М.: Мир, 1991. С. 246 - 267.
13. Воеводин В.В. Математические основы параллельных вычислений, М., МГУ, 1991, 345 с.
14. Воеводин В.В. Воеводин Вл.В. Параллельные вычисления, С-Петербург «БХВ-Петербург», 2002, 599 с.
15. Воеводин Вл. В. Статистические оценки возможности выявления параллельной структуры последовательных программ. // Программирование, № 4, 1990, с. 44-54.
16. Воеводин Вл. В. Теория и практика исследования параллелизма последовательных программ. // Программирование, № 3, 1992, с. 38-54.

17. Волконский В. Ю., Ким А.К. Развитие идей параллелизма в архитектуре вычислительных комплексов серии «Эльбрус» PACO'2008, Труды четвертой международной конференции «Параллельные вычисления и задачи управления», Москва.: 27-29 октября 2008 г. с. 42-66.
18. Гергель В.П., Стронгин Р.Г. Основы параллельных вычислений для многопроцессорных вычислительных систем (2003, 2 изд.). Н.-Новгород, ННГУ.
19. Евстигнеев В.А. Некоторые особенности программного обеспечения ЭВМ с длинным командным словом (Обзор) //Программирование, 1991, N 2, с. 69-80.
20. Евстигнеев В.А., Спрогис С.В. Векторизация программ. - В сборнике: "Векторизация программ: теория, методы, реализация." - М.: Мир, 1991, с. 246 - 267.
21. Евстигнеев В.А., Касьянов В.Н. Оптимизирующие преобразования в распараллеливающих компиляторах// Программирование, 1996, № 6, с. 12-26.
22. Затуливетер Ю., Фищенко Е. Многопроцессорный компьютер ПС-2000//Открытые системы, из-во «Открытые системы», 2007 г., № 9, с. 74-79.
23. Касперский К. Техника оптимизации программ. Эффективное использование памяти. - СПб.: БХВ-Петербург, 2003 г. – 456 с.
24. Касьянов В.Н., Оптимизирующие преобразования программ/ М., «Наука», 1988 г., 336 с.
25. Касьянов В.Н., Евстигнеев В.А. Графы в программировании: обработка, визуализация и применение. - СПб.: БХВ-Петербург, 2003 г. – 1104 с.
26. Кодачигов В.И. Электронная коммутация информационных каналов. – Ростов-на-Дону, изд-во Ростовского университета, 1983, 208 с.
27. Корнеев В.В. Программная настраиваемость аппаратной архитектуры//Открытые системы, из-во «Открытые системы», 2007 г., № 10.
28. Корнеев В.В. Архитектура вычислительных систем с программируемой структурой. Новосибирск, Наука, 1985 г., 166 с.
29. Лацис А.О. Как собрать и использовать суперкомпьютер.// М. «Бестселлер», 2003, 238 с.
30. Лиходед Н.А. Распределение операций и массивов данных между процессорами.// Программирование, 2003, №3, с. 73-80.
31. Нис З.Я. Обеспечение статической семантической корректности программ при проведении распараллеливающих и оптимизирующих преобразований. PACO'2008, Труды четвертой международной конференции «Параллельные вычисления и задачи управления», Москва.: 27-29 октября 2008 г. с.
32. 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. с.
33. Прангишвили И.В., Виленкин С.Я., Медведев И.Л., Параллельные вычислительные системы с общим управлением// М.: Энергоатомиздат, 1983. – 312 с.
34. Слуцкин А., Эйсымонт Л. Российский суперкомпьютер с глобально адресуемой памятью//Открытые системы, из-во «Открытые системы», 2007 г., № 9, с. 41-51.
35. Сухинов А.И. Двумерные схемы расщепления и некоторые их приложения. М. МАКС пресс, 2005. – 408 с.
36. Фролов А., Семенов А., Корж А., Эйсымонт Л. Программа создания перспективных суперкомпьютеров//Открытые системы, из-во «Открытые системы», 2007 г., № 9, с. 21-29.
37. Штейнберг Б.Я. Распараллеливание рекуррентных циклов с условными операторами// Автоматика и телемеханика/ 1995, N6, с. 176-184.
38. Штейнберг Б.Я. Распараллеливание с анализом области изменения параметров циклов и с анализом внешних переменных в индексных выражениях. - Всесоюзный научно-технический семинар (г. Калинин) "Разработка системного и прикладного программного обеспечения МВК ПС-2000/2100, ПС-3000/3100", тезисы докладов, Москва, 1990, с. 5-6.
39. Штейнберг Б.Я. Оптимальные параллельные переразмещения двумерных массивов.//Программирование, N 6, 1993 г. с.81-88.
40. Штейнберг Б.Я. Бесконфликтные размещения массивов при параллельных вычислениях// Кибернетика и системный анализ/ 1999, N 1, с. 166-178.

41. Штейнберг Б.Я. Оптимальные параллельные переразмещения многомерных массивов при параллельных вычислениях// Международная научно-техническая конференция <Интеллектуальные многопроцессорные системы>/ 1-5 сентября, 1999, Таганрог, Россия, Сборник трудов, с.151-155.
42. Штейнберг Б.Я. Распараллеливание рекуррентных программных циклов.// Информационные технологии, №4, 2004 г., с. 16 – 23.
43. Штейнберг Б.Я. Открытая распараллеливающая система//Открытые системы, из-во «Открытые системы», 2007 г., № 9, с. 36-41.
44. Штейнберг Б.Я., Морылев Р.И. Распараллеливание программ с помощью Открытой Автоматическое распараллеливающей системы// Научный сервис в сети Интернет. Труды Всероссийской научной конференции, г. Новороссийск, 18-22 сентября 2007. – М.: изд-во МГУ.
45. Штейнберг Б.Я. Подстановка и переименование индексных переменных в многомерных циклах.// Известия вузов. Северокавказский регион. Юбилейный выпуск. 2002 г., с. 94-99.
46. Шульженко А.М. определение циклов ParDo в программе // Известия вузов. Северокавказский регион. Естественные науки. Приложение 11'05. с. 77-88.
47. Штейнберг Р.Б., Вычисление задержки в стартах конвейеров для суперкомпьютеров со структурно процедурной организацией вычислений// Искусственный интеллект. Научно-теоретический журнал. Институт проблем искусственного интеллекта НАНУ. Украина, Донецк, ДонДИШИ, "Наука и Освита", № 4, 2003, с. 105-112.
48. Bondalapati K. Modeling and Mapping for Dynamically Reconfigurable Hybrid Architecture. Ph.D. Thesis, University of Southern California, August, 2001.
49. Bondalapati K., Prasanna V. Reconfigurable computing systems, // http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.11.4918, 2002, 40 p.
50. Feautrier Paul Parametric Integer Programming// Laboratoire MASI, Institut Blaise Pascal, Universite de Versailles St-Quentin, 1988, p. 25.
51. Feautrier Paul Data Flow Analysis for Array and Scalar References// International Journal of Parallel Programming/V. 20, N 1, Feb. 1991, p. 23-53.
52. Feautrier Paul. Some efficient solutions no the affine scheduling problem. Part 1. One-dimentional Time. //"International journal of Parallel Programming", V. 21, № 5, Octouber, 1992.
53. Feautrier Paul. Some efficient solutions no the affine scheduling problem. Part 2. Multidimentional Time. // Technical Report, IBP/MASI, Number 92.78, Octouber, 1992, 28 p.
54. Lamport L. The Coordinate Method for the parallel execution of DO loops// Sagamore Computer Conference on Parallel Processing.- 1973, p. 1-12.
55. Lamport L. The parallel execution of DO loops// Commun.
ACM.- 1974.- v.17, N 2, p. 83-93.
56. Strassen V. Gaussian Elimination is not Optimal. – Numer. Math. 1969, vol. 13, № 4, p. 354-356.
57. Treleaven P.C. Parallel architecture overview.// Parallel Computing, 8 (1988), p. 59-70. North-Holland.
58. www.top500.org
59. www.parallel.ru
60. www.parallel.ru/v-ray
61. www-suif.stanford.edu/research/
62. www.ops.rsu.ru
63. Штейнберг Б. Я. Блочно рекурсивное параллельное перемножение матриц. «Известия ВУЗов. Приборостроение», т. 52, №10, 2009 г., с. 33-41.
64. Штейнберг Б.Я. Блочно-аффинные размещения данных в параллельной памяти. «Информационные технологии». №6, 2010, стр. 36-41
65. Штейнберг Р. Б. Отображение гнезд циклов на многоконвейерную архитектуру, Программирование, 2010, № 3.

66. Штейнберг Р. Б. Использование решетчатых графов для исследования многоконвейерной модели вычислений. «Известия ВУЗов. Северокавказский регион. Естественные науки», №2, 2009 г., с. 16-18.
67. Штейнберг О. Б. Распараллеливание рекуррентных циклов с нерегулярным вычислением суперпозиций. «Известия ВУЗов. Северокавказский регион. Естественные науки», №2, 2009 г., с. 18-21.
68. Суховерхов С.Е., Штейнберг О. Б. Автоматическое распараллеливание рекуррентных циклов с проверкой устойчивости. «Информационные технологии», №1, 2010 г.
69. Алымова Е.В. Автоматическая генерация тестов на основе конфигурационных файлов для оптимизирующих преобразований компилятора «Известия ВУЗов. Северокавказский регион. Естественные науки», 2010, №3
70. Абрамян М.Э. Реализация универсального электронного задачника по программированию // Информатика и образование, 2009, № 6. — С. 118–120.
71. Савельев В.А., Штейнберг Б. Я. Распараллеливание программ. Ростов-на-Дону, изд-во Южного федерального университета, 2008 г. 192 с.
72. Штейнберг Б.Я. Оптимизация размещения данных в параллельной памяти. Ростов-на-Дону, изд-во Южного федерального университета, 2010, 255 с.
73. Б.Я. Штейнберг, Е.В. Алымова, А.П. Баглий, Р.И. Морылев, З.Я. Нис , В.В. Петренко, Р.Б. Штейнберг. Автоматизация тестирования элементов высокопроизводительного программного комплекса. Научный сервис в сети Интернет: масштабируемость, параллельность, эффективность: Труды Всероссийской суперкомпьютерной конференции (21-26 сентября 2009 г., г. Новороссийск). М.: Изд-во МГУ, 2009. – (с. 287-291)
74. Steinberg B., Alimova E., Baglij A., Morilev R., Nis Z., Petrenko V., Steinberg R. The System for Automated Program Testing. Proceedings of IEEE East-West Design & Test Symposium (EWDTS'09). Moscow, Russia, September 18-21, 2009 (pp. 218 - 220)
75. Нис З. Я. Преобразования программ: контроль семантической корректности // Изв. вузов. Сев.-Кавк. регион. Естественные науки. 2010 г., №1. С. 18-21.
76. Корнеев В.В. Проблемы программирования суперкомпьютеров на базе многоядерных мультредовых кристаллов. Научный сервис в сети Интернет: масштабируемость, параллельность, эффективность: Труды Всероссийской суперкомпьютерной конференции (21-26 сентября 2009 г., г. Новороссийск). М.: Изд-во МГУ, 2009.
77. Полетаев С.А. Параллельные вычисления на графических процессорах. Конструирование и оптимизация параллельных программ. В сб. Новосибирск: Институт систем информатики им. А.П. Ершова СО РАН, 2008, 332 с.
78. Корж А.А. Результаты масштабирования бенчмарка NPB UA на тысячи ядер суперкомпьютера Blue Gene/P с помощью PGAS-расширения OpenMP. Вычислительные методы и программирование. 2010, т. 11, с. 31-41.
79. Ершов Н.М., Попов А.М. Оптимизация параллельных вычислений с учетом архитектуры и быстродействия связей в вычислительной системе.// Вестн. Моск. ун-та. Сер.15, Вычислительная математика и кибернетика. 1993, N 1. С. 24-30.
80. Левин И.И., Штейнберг Б.Я. Сравнительный анализ эффективности параллельных программ для различных архитектур параллельных ЭВМ. // Искусственный интеллект. Научно-теоретический журнал. Институт проблем искусственного интеллекта НАНУ. Украина, Донецк, ДонДИШИ, "Наука и Освита", 2001, № 3, с. 234-242.
81. Самарский А.А. Введение в численные методы.-М.: Наука, 1987, 280 с.
82. Каляев А.В., Левин И.И. Модульно-наращиваемые многопроцессорные системы со структурно-процедурной организацией вычислений //М., «Янус-К», 2003, 380 с.
83. http://www.csrd.uiuc.edu/promis/
73. http://www.parallelsp.com/
84. Дроздов А. Ю. Компонентный подход к построению оптимизирующих компиляторов // Программирование. 2009. № 5.

85. R. Allen, K. Kennedy Optimizing compilers for Mordern Architetures// Morgan Kaufmann Publisher, Academic Press, USA, 2002, 790 p.