ПРОГРАММИРОВАНИЕ, 1996, N° 3, с.1 - ??

Тематический раздел журнала

УДК 21.22

ЭФФЕКТИВНАЯ АДАПТАЦИЯ ПОСЛЕДОВАТЕЛЬНЫХ ПРОГРАММ ДЛЯ СОВРЕМЕННЫХ ВЕКТОРНО-КОНВЕЙЕРНЫХ И МАССИВНО-ПАРАЛЛЕЛЬНЫХ СУПЕР-ЭВМ

© 1996г. А.С. Антонов, Вл.В. Воеводин

Поступила в редакцию 22 января 1996

Данная работа посвящена обсуждению основных этапов и проблем переноса программного обеспечения на параллельные компьютеры. В качестве примера рассматривается адаптация двух программ известного пакета Perfect Club Benchmarks, для которых удалось получить лучшие, на настоящий момент, показатели производительности для векторно-конвейерных супер-ЭВМ CRAY Y–MP и массивно-параллельного компьютера CRAY T3D.

1. ВВЕДЕНИЕ

Несмотря на то, что проблема оптимизации программ для компьютеров с параллельной архитектурой существует уже более двадцати лет, ее решение до сих пор нельзя считать удовлетворительным, даже для ставших уже классическими, векторно-конвейерных супер–ЭВМ. В действительности, было предложено много различных подходов к оптимизации, но не один из них не предполагает комплексного решения всего множества возникающих проблем. В данной работе мы описываем процесс адаптации двух последовательных программ для семейства векторно-конвейерных компьютеров CRAY и массивно-параллельного компьютера CRAY T3D.

Подбирая материал для данной работы, мы ставили перед собой две цели. Во-первых, мы хотели показать основные этапы самого процесса адаптации программ к архитектуре параллельных компьютеров. В той или иной форме через эти этапы придется пройти всем, кто будет вовлечен в подобного рода деятельность, поэтому полученный нами опыт поможет другим исследователям лучше представить характер и объем предстоящей работы.

На многих этапах приходилось решать довольно сложные, даже в плане постановки, задачи. С частью из них нам уже приходилось иметь дело, однако многие возникли лишь в процессе реальной работы. Эта сторона проделанной работы предопределила вторую цель — описать и тем самым привлечь внимание других специалистов к задачам, возникающим на каждом из этапов процесса адаптации. Многие задачи хорошо известны, и для их решения существуют эффективные методы, но другую часть до сих пор приходится "решать вручную", что делает процесс адаптации очень трудоемким.

Описываемые в данной работе анализ и преобразование программ мы выполняли с помощью V-Ray технологии и построенной на ее основе системе V-Ray. Технология имеет серьезное математическое основание, опирающееся на результаты теории графов алгоритмов, теории графов и теории статического анализа программ. Сама технология описана в нескольких работах, например, в [1,2], а здесь мы хотели бы больше сосредоточиться на результатах и опыте реальной работы на супер-ЭВМ.

Для экспериментов мы взяли пакет Perfect Club Benchmarks, который уже в течение многих лет используется для определения реальной производительности параллельных компьютеров на промышленных приложениях. К настоящему времени его состав, структура отдельных программ, ключевые фрагменты отдельных процедур, производительности для большого числа различных компьютеров и многие другие параметры подробно исследованы и неоднократно описаны во многих работах [3,4]. Однако именно широкая известность пакета и показалась нам привлекательной: во-первых, намного интереснее получить нетривиальный результат там, где прежде уже работали другие исследователи, а, во-вторых, только таким образом и можно объективно оценить эффективность новой технологии по сравнению с существующими методами анализа и преобразования программ.

Из самого пакета мы выбрали программы TRFD и FLO52, так как первая имеет крайне низкую производительность на всех параллельных компьютерах фирмы CRAY, а вторая очень сложна для адаптации к массивно-параллельной архитектуре и, насколько нам известно, успешных попыток ее оптимизации не было.

Суть проблемы легко объяснить на примере программы TRFD. На исходном варианте TRFD один процессор компьютера CRAY Y–MP C90 с пиковой производительностью почти 1 Gflop/s достигает только 89.5 Mflop/s. Лучший вариант ручной оптимизации этого кода увеличил производительность лишь до 139.7 Mflop/s (значения производительности после ручной оптимизации предоставлены Д.Бруксом из CRAY Research, Inc., Eagan MN, USA). Ясно, что программа в первоначальном виде не только достаточно плохо векторизуется, но и "ручные" изменения не могли существенно исправить положения дел. Естественно сразу возникают два вопроса:

- В чем причина столь низкой производительности?
- Можно ли увеличить производительность только с помощью эквивалентного преобразования исходного текста данной программы?

Забегая немного вперед, мы дадим ответ сразу — да, это сделать можно! В результате проведенных исследований нам удалось, во-первых, поднять производительность одного процессора CRAY Y–MP C90 на данной программе до 579.7 Mflop/s, во-вторых, добиться устойчивого роста производительности для многопроцессорных конфигураций, и, наконец, получить действительно масштабируемую программу с производительностью 1.7 Gflop/s на 256-ти процессорном компьютере CRAY T3D. Как добиться такого эффекта — об этом и будет рассказано ниже.

Работа состоит из четырех частей. В первой части кратко описаны основные особенности векторно-конвейерных и массивно-параллельных компьютеров. Вторая и третья части посвящены описанию процесса оптимизации программ TRFD и FLO52, и вместе с этим, эти части содержат описание главных этапов, характерных для оптимизации практически всех реальных программ. В заключении приведен краткий список основных задач, с которыми приходится сталкиваться в процессе анализа и преобразования программ, причем для одних задач решение получить не сложно, а для других настолько нетривиально, что вопрос о нахождении приемлемого решения остается открытым до сих пор.

2. НЕМНОГО О КОМПЬЮТЕРАХ CRAY Y–MP И CRAY T3D

Архитектура компьютеров **CRAY Y-MP C90** сохранила все лучшие черты векторно-конвейерных моделей фирмы CRAY Research, Inc.: CRAY–1, CRAY X–MP и CRAY Y–MP M90. Как и прежде, основная ставка делается на эффективное выполнение конвейерными функциональными устройствами операций над векторами. Время такта уменьшено до 4.1 нсек и, вместе с тем, увеличен объем оперативной памяти до 8 Гбайт, причем вся память одинаково доступна для всех процессоров. Архитектура этого компьютера достаточно хорошо известна, и ее описание можно найти, например, в [5].

Появившийся относительно недавно **CRAY T3D** представляет собой массивно-параллельный компьютер с распределенной памятью, объединяющий от 32 до 2048 процессоров DEC ALPHA. Каждый процессор работает на частоте 150 МГц, имеет свою локальную память от 2 до 8 Мслов (каждое слово — 64 разряда), кэш-память команд и кэш-память данных по 1Кслову. Все процессоры разбиты на пары, каждая пара составляет вычислительный узел, а все узлы объединены в трехмерный тор. Связь между соседними узлами может идти одновременно в обоих направлениях по двум однонаправленным каналам. Маршрутизация сообщений между двумя узлами всегда происходит так, что сначала выравниваются их координаты по измерению X, затем по Y, и в конце по Z. Максимальная скорость передачи по каналу составляет 140 Мбайт/с.

Рис. 1.

Несмотря на то, что память физически распределена по процессорам, предусмотрена аппаратная поддержка удаленного доступа в память другого процессора, причем время удаленного доступа всего лишь в 6 раз больше времени обращения к локальной памяти. Специальные аппаратные схемы синхронизации являются хорошей поддержкой для реализации традиционно "неудобных" примитивов параллельной работы процессов типа барьерной синхронизации.

3. ИССЛЕДОВАНИЕ И ОПТИМИЗАЦИЯ ПРОГРАММЫ TRFD

3.1. Общая характеристика и структура TRFD

Итак, прежде, чем начинать оптимизацию программы, необходимо понять ее общую структуру: какие подпрограммы составляют основное вычислительное ядро, где сосредоточен ввод/вывод, какова последовательность вызова отдельных подпрограмм и т. п. Для программы TRFD это сделать легко, так как она относительно невелика — весь ввод/вывод расположен в главной части, а бо́льшая часть времени приходится на подпрограмму OLDA. Структуру данной подпрограммы можно представить с помощью ее циклического профиля, в котором каждая горизонтальная скобка соответствует одному циклу DO (рис. 1).

Значительное число циклов, большая глубина их вложенности (5) в совокупности и определяют вычислительную сложность данной подпрограммы. Однако для анализа информационной структуры эти факторы сами по себе не являются серьезными препятствием. Важной особенностью подпрограммы OLDA является то, что, во-первых, число итераций в каждом цикле относительно невелико, и, во-вторых, основной многомерный массив в целях экономии памяти хранится компактно в виде одномерного, поэтому индексные выражения вычисляются по сложным нелинейным (относительно параметров объемлющих циклов) формулам.

После первого анализа подпрограммы OLDA были обнаружены три интересных факта:

- векторизуемость всех самых внутренних циклов OLDA можно определить даже самыми простыми средствами анализа;
- возможность распараллеливания циклов, находящихся на внешнем уровне, определить формальными методами сложно, но легко глядя на текст, иными словами, визуально. Именно по этой причине не увеличивались значения базовой производительности для исходного варианта с ростом числа процессоров, и именно этот факт позволил увеличить значения производительности для многопроцессорных конфигураций в случае ручной оптимизации;
- комбинация сложной циклической структуры, большой вложенности циклов и компактной формы хранения основного массива является очень серьезным препятствием для выполнения сколько-нибудь сложного преобразования программы.

Безусловно, третий факт сильно портит общее впечатление о программе, но если подпрограмма OLDA уже обладает "хорошей" структурой, то, быть может, нам ничего с ней делать и не придется? Особенности архитектуры компьютеров CRAY Y–MP — несколько векторно-конвейерных процессоров над общей памятью, диктуют общепринятую "оптимальную" форму программ для машин данного класса: векторизуемость самых внутренних циклов и возможность распараллеливания на внешнем уровне.

Для начала посмотрим, что дает реализация исходного варианта TRFD, которая учитывает лишь структуру внешних и внутренних циклов. С одной стороны, в исходной программе удалось найти структуры, хорошо согласующиеся с архитектурой компьютеров типа CRAY Y–MP: практически все операции программы исполняются в векторном режиме и практически все время все процессоры могут быть загружены полезной работой. С другой стороны, даже в случае ручной оптимизации, производительность каждого процессора не превышает 25% от

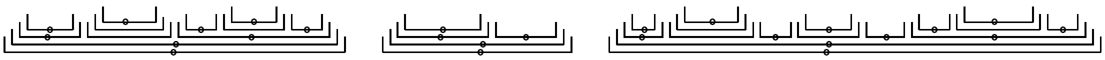

Рис. 2.

пиковой!

Ясно, что надо попытаться что-то изменить, но выполнить какое-либо серьезное преобразование программы, так же как и понять, на что именно оно должно быть направлено, нельзя — структура программы во многом осталась нераскрытой. В самом деле мы определили только структуру самых внешних и самых внутренних циклов исходной программы. Имея столь бедную информацию, мы не сможем найти решение ни одной серьезной проблемы:

- нельзя оценить потенциальных свойств программы, в частности, потенциал параллелизма;
- многие свойства программы скрыты, а значит и не понятно, на устранение каких факторов должно быть направлено преобразование;
- если, исходя из каких-то предположений, мы и выберем сколько-нибудь нетривиальное преобразование, то в рамках существующей формы записи программы выполнить его будет технически крайне сложно;
- в принципе, возможен и такой вариант, при котором уже полученные для данной программы 139.7 Mflop/s мы превысить и не сможем, но в этом надо быть точно уверенным прежде, чем отказываться от каких-либо дальнейших попыток улучшить производительность.

Весь указанный комплекс проблем был успешно решен, и принципиально важные этапы исследования структуры программы описаны в следующем разделе.

3.2. TRFD и векторно-конвейерная архитектура CRAY Y–MP

Восстановление полноразмерных массивов по их компактной форме. Мы уже говорили о том, что серьезным препятствием для анализа TRFD является компактная форма хранения массивов. Чтобы решить эту проблему, мы использовали технику восстановления полноразмерных структур данных по их компактной форме. Несмотря на то, что данная задача может иметь много решений, мы можем взять любое, при котором новые индексные выражения будут иметь вид линейной функции. В данном случае для нас важно определить структуру программы, а с помощью каких именно структур данных — тех, с которыми работал автор программы, или каких-либо других — это не так существенно. Одновременно заметим, что мы совсем не отказываемся от задачи экономии памяти, а лишь откладываем ее решение на более поздний этап.

Основной результат данной процедуры заключается в том, что после ее выполнения *информационная структура подпрограммы OLDA может быть полностью определена.* Это действительно принципиально важный шаг, который помог выполнить эффективную оптимизацию данной программы как для CRAY Y–MP, так и для CRAY T3D.

Определение точной информационной структуры программы. Приведение подпрограммы к анализируемой форме позволило точно определить все интересующие нас свойства: существование или отсутствие зависимости между отдельными операторами и фрагментами, локальность использования данных, возможность приватизации массивов и многие другие, наиболее важные из которых будут описаны ниже.

Нахождение и описание всего потенциального параллелизма. Данный пункт является прямым следствием предыдущего, так как если мы знаем точную информационную структуру программы, то, естественно, можем выделить и весь ее потенциальный параллелизм. Весь координатный параллелизм циклов подпрограммы OLDA хорошо виден на ее циклическом профиле (рис. 2), где точками помечены циклы с независимыми итерациями (восстановление полноразмерных массивов привело к появлению новых циклов, которых не было на профиле исходной подпрограммы).

Заметим, что из данной картинки следует

Таблица 1.

CRAY M90 CPUs	Пиковая произв. Mflop/s	Базовая произв. Mflop/s	Ручная опт. Mflop/s	V-Ray опт. Mflop/s
1	333	56.19^*	$81.72^!$	$247^\dagger$
4	1332	54.86	261.64	822
8	2664	54.34	481.03	$954^{+\ddagger}$
CRAY C90 CPUs	**Пиковая произв. Mflop/s**	**Базовая произв. Mflop/s**	**Ручная опт. Mflop/s**	**V-Ray опт. Mflop/s**
1	960	89.5^*	$139.71^!$	$579.7^\dagger$
8	7680	89.6	962.68	$2440.5^\ddagger$
CRAY EL CPUs	**Пиковая произв. Mflop/s**	**Базовая произв. Mflop/s**	**Ручная опт. Mflop/s**	**V-Ray опт. Mflop/s**
1	132	11.37	—	54.60
2	264	11.40	—	106.03
4	528	11.38	—	193.32

Результаты оптимизации программы TRFD для семейства векторно-конвейерных суперкомпьютеров CRAY.

очень важный вывод: если в программе использовать только параллелизм самых внешних и самых внутренних циклов, то большая часть потенциала программы не будет никак использована. Одновременно с этим, мы видим и тот резерв, за счет которого можно в дальнейшем получить ускорение, и который определяет общую стратегию преобразования программы:

- объединить по два самых внешних цикла в первом и третьем гнезде в один, увеличив их длину в среднем с $num/2$ до $num * (num+1)/2$, переместить эти длинные циклы внутрь и именно их использовать для векторизации (второе гнездо служит только для пересылки данных и не содержит арифметических операций);
- самые внутренние циклы исходного варианта подпрограммы переместить наружу и использовать для параллельного исполнения всеми процессорами компьютера CRAY Y–MP;
- последовательные циклы, определяющие перевычисление одних и тех же элементов массивов, использовать для повышения локальности обращения к данным.

Оптимизация работы с памятью. Оптимизация работы с памятью имеет большое значение для эффективной работы программ на компьютерах типа CRAY Y–MP, хотя этот аспект очень часто упускается из рассмотрения. Чаще всего данный вид оптимизации сводится к уменьшению числа пересылок данных между памятью и векторными регистрами. В терминах V-Ray технологии такая оптимизация означает исследование локальности использования данных и целесообразности использования временных массивов. В частности, для программы TRFD именно увеличение локальности использования векторов привело к заметному увеличению производительности.

Приватизация (privatization) массивов. Данная задача к настоящему моменту известна достаточно хорошо и заключается в нахождении переменных, которые могут быть объявлены локальными для каждой параллельно работающей ветви программы. Использованная техника приватизации массивов, так же как и техника исследования свойств локальности программы, опиралась на анализ формульного представления структуры графа алгоритма исследуемого фрагмента.

Была проведена серия экспериментов на компьютерах CRAY Y–MP M90/C90 и CRAY EL, результаты которых приведены в таблице 1. Четыре столбца с различными значениями производительности означают следующее:

- "Пиковая произв.": пиковая производительность компьютера в указанной конфигура-

ции;
- "Базовая произв.": производительность, полученная на программе, к которой не применялось никакой ручной оптимизации, за исключением использования подходящих флагов компилятора;
- "Ручная опт.": наилучшие значения производительности, полученные на момент выполнения наших исследований с помощью ручной оптимизации кода;
- "V-Ray опт.": значения производительности, полученные после преобразования программы TRFD согласно V-Ray технологии.

Сделаем несколько замечаний к табл. 1:
(+) данное значение получено не в монопольном режиме и заведомо может быть улучшено;
(*) цифры занимают седьмое место среди значений производительности всех тринадцати программ пакета Perfect Club;
(!) цифры занимают девятое место среди значений производительности "Ручная опт." всех оптимизированных программ данного пакета;
(†) цифры заняли первое место среди значений производительности всех оптимизированных ("Ручная опт.") программ пакета;
(‡) цифры заняли второе место среди значений производительности всех оптимизированных ("Ручная опт.") программ пакета Perfect Club, уступая лишь хорошо оптимизируемой программе MG3D.

Важно отметить и то, что вся оптимизация данной программы была выполнена на уровне Фортрана без использования ассемблерных вставок и обращений к высокоэффективным библиотечным подпрограммам.

3.3. TRFD и массивно-параллельная архитектура CRAY T3D

Методы оптимизации программы TRFD, которые мы использовали для компьютера CRAY T3D, коренным образом отличаются от методов оптимизации для векторно-конвейерных моделей CRAY Y-MP M90/C90 и CRAY EL. Это различие определяется тем, что CRAY T3D — это массивно-параллельный компьютер с распределенной памятью.

Для получения действительно масштабируемой программы с хорошей производительностью необходимо максимально интенсивно использовать все особенности данной архитектуры, а значит, в процессе исследования и преобразования исходной программы надо попытаться:
- выделить параллельные ветви вычислений, число которых должно быть сравнимо с числом процессоров;
- найти распределение массивов по процессорам, согласованное с параллелизмом вычислений для минимизации межпроцессорного обмена;
- провести оптимизацию на каждом процессоре для эффективного использования кэш-памяти и сбалансированной загрузки функциональных устройств процессора DEC ALPHA.

Ресурс параллелизма программы TRFD нам уже известен — его основная часть была показана на размеченном циклическом профиле подпрограммы OLDA. Для исследования множества возможных распределений данных по процессорам мы использовали средства V-Ray технологии, в результате чего было *доказано*, что:

- для каждого из трех гнезд подпрограммы OLDA существуют распределения данных, согласованные с использованием двумерного параллелизма (двумерный параллелизм предполагает независимость итераций двух циклов гнезда, а согласованность распределения, в данном случае, означает отсутствие пересылок данных между итерациями этих циклов);
- не существует единого для всех трех гнезд распределения данных, согласованного с использованием хотя бы одномерного параллелизма в каждом гнезде;

Располагая такой информацией, нам остается единственное решение: параллельное выполнение каждого гнезда и перераспределение данных между ними. Результирующая параллельная программа была написана в терминах специальной библиотеки коммуникационных примитивов, поскольку штатные средства (типа PVM для CRAY T3D) оказались очень неэффективными.

Результаты численных экспериментов показаны в табл. 2, столбцы 2-5 которой означают следующее:

Таблица 2.

CRAY T3D PEs	PVM опт. seconds	No cache опт. seconds	V-Ray опт. seconds	V-Ray опт. Mflop/s
4	11.35	11.03	6.013	71.66
8	6.05	6.04	3.215	134.01
16	3.23	3.23	1.702	253.43
32	1.68	1.72	0.896	480.85
64	1.02	0.93	0.494	871.44
128	0.79	0.56	0.301	1435.96
256	0.86	0.41	0.253	1700.93

Результаты оптимизации программы TRFD для массивно-параллельного суперкомпьютера CRAY T3D.

"PVM опт." — результаты работы параллельной программы, написанной в терминах PVM, без выполнения скалярной оптимизации;
"No cache опт." — результаты работы параллельной программы, написанной в терминах нашей библиотеки коммуникационных примитивов, без выполнения скалярной оптимизации;
"V-Ray опт." — предыдущий вариант и скалярная оптимизация.

Данная таблица очень хорошо демонстрирует эффект, получаемый на разных этапах оптимизации программы. Несмотря на присутствие в исходной версии программы операции, аналогичной транспонированию (второе гнездо в подпрограмме OLDA), которая, по общему признанию, является крайне "неудобной" для массивно-параллельных компьютеров с распределенной памятью, очевидно высокое качество полученной параллельной программы.

4. ИССЛЕДОВАНИЕ И ОПТИМИЗАЦИЯ ПРОГРАММЫ FLO52

4.1. Общая характеристика и структура FLO52

Программа FLO52 представляет собой упрощенный вариант реализации одной из задач вычислительной гидродинамики. В отличие от TRFD, она достаточно велика и состоит из 31 подпрограммы общим объемом более 2000 строк на языке Фортран. Исходный вариант FLO52 уже был написан в расчете на векторно-конвейерную архитектуру, поэтому для нас основной интерес представляла возможность эффективной адаптации этой программы к принципиально иной архитектуре компьютера CRAY T3D.

Как и для любой программы, первое, что необходимо понять перед оптимизацией, — это ее общая структура: где сосредоточена наибольшая часть вычислений, где находится ввод/вывод, инициализация и т. д. Эту информацию можно получить разными способами, например, анализируя граф вызовов программы и графы управления отдельных подпрограмм. На рис. 3 показан граф вызовов процедур программы FLO52. Темные вершины соответствуют подпрограммам основного вычислительного ядра, на которое приходится почти 90% времени выполнения всей программы, поэтому их анализ надо производить в первую очередь.

Наклонной чертой в графе помечены подпрограммы, содержащие операции ввода-вывода. Поскольку параллельный ввод-вывод в CRAY T3D пока не реализован, мы были вынуждены выполнять эти подпрограммы в последовательном режиме. При последовательном выполнении всей программы на подпрограммы с вводом-выводом суммарно приходится около 2% времени, поэтому по закону Амдала получить ускорение больше 50-ти будет в принципе невозможно.

4.2. Потенциал параллелизма программы

После определения структуры программы и выделения наиболее важных ее участков возникает задача нахождения всего присущего программе параллелизма. Для оценки потенциала параллелизма исследуется информационная

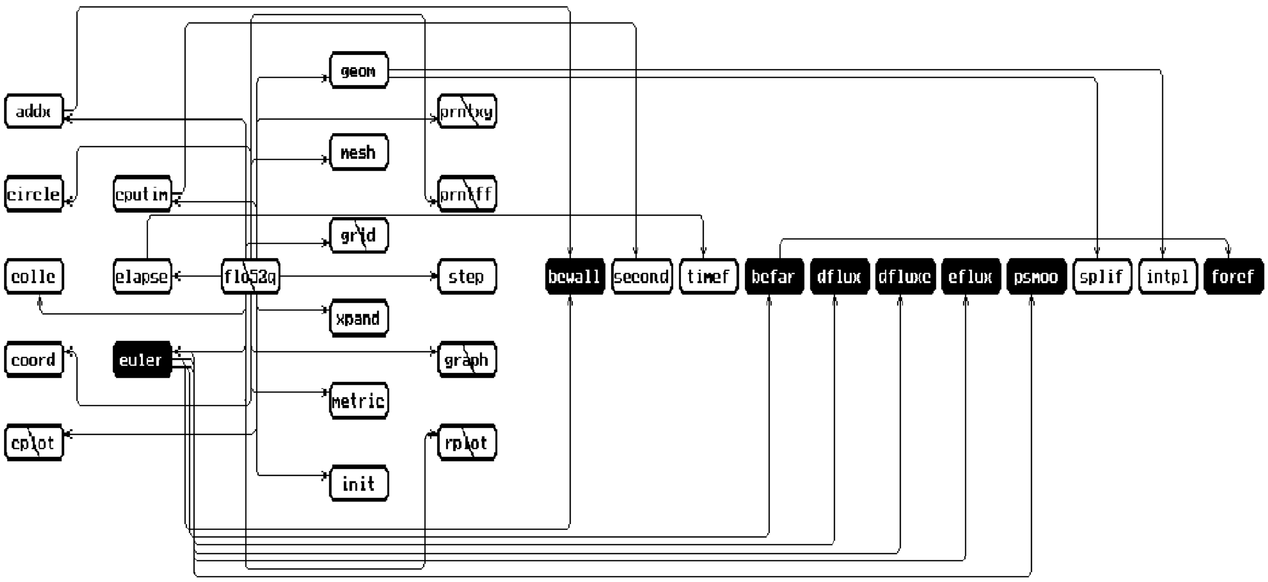

Рис. 3.

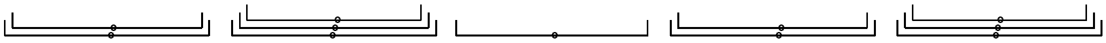

Рис. 4.

структура отдельных подпрограмм, что позволяет найти резерв основных видов параллелизма: независимых ветвей вычислений, координатного и скошенного параллелизма циклов, а также выделить последовательные циклы, реализующие глобальные операции, для которых можно будет использовать специальные методы обработки.

Применительно к FLO52 было обнаружено, что в программе есть **большой резерв** простого для использования **координатного параллелизма**. Практически все циклы ее вычислительного ядра обладают свойством **ParDo**, которое означает, что на последовательность выполнения итераций этих циклов не накладывается никаких ограничений, и, в частности, их можно выполнять параллельно. Возьмем, например, циклический профиль подпрограммы EFLUX (рис. 4), из которого видно, что этим свойством обладают все циклы данной подпрограммы, и такая ситуация является типичной для всего ядра.

Последовательные циклы — кандидаты в глобальные операции. Даже если цикл является последовательным, иногда его можно выполнить более эффективно, используя другой алгоритм для заложенной в нем операции. В подпрограммах EULER, FORCF и STEP были обнаружены фрагменты, в которых выполняются глобальные операции над целыми массивами (нахождение глобальной суммы, глобальных минимума и максимума). При реализации таких операций на компьютере с распределенной памятью потребуются существенные накладные расходы, поэтому обнаруженные фрагменты были заменены на вызовы подпрограмм, оптимизированных специально для архитектуры CRAY T3D. Поскольку подобная ситуация является типичной для массивно-параллельных компьютеров, то практически везде заранее предусмотрены процедуры для глобальных операций.

Есть ли дополнительные резервы параллелизма? Этот вопрос принципиально важен при адаптации любой программы. Используя аналитические и визуальные средства анализа системы V-Ray, можно показать, что другой возможности, кроме использования координатного параллелизма подпрограмм, нет. С другой стороны, использование всего найденного параллелизма нецелесообразно из-за сложности необходимых пересылок данных, поэтому в дальней-

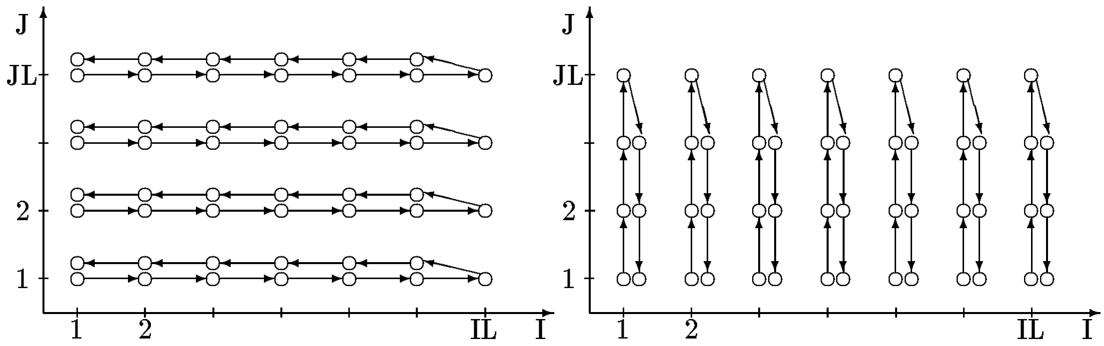

Рис. 5.

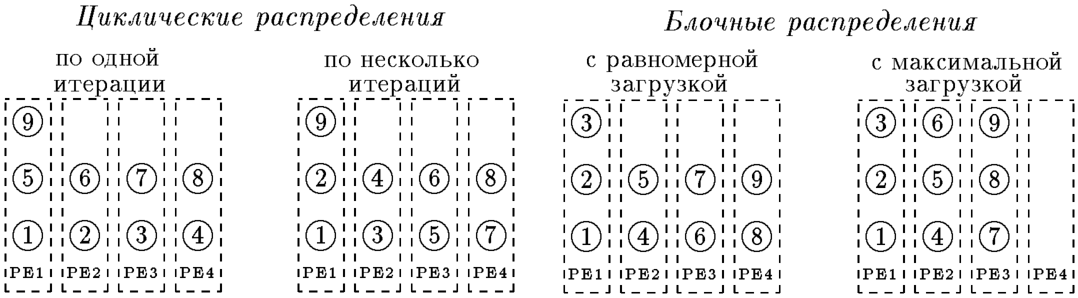

Рис. 6.

шем надо определить ту часть параллелизма, на которой и будет базироваться параллельная версия программы.

4.3. Какое распределение данных использовать?

После нахождения ресурса параллелизма программы надо определить, каким образом его лучше использовать, т. е. как распределить данные программы и выполняемые операции между ПЭ (процессорными элементами). Цель оптимизации программы — минимизация времени ее исполнения. Этого можно достичь с помощью равномерного распределения работы между процессорами и минимизации межпроцессорного взаимодействия.

Тривиальное распределение массивов. Часть фрагментов FLO52 устроена так, что для них одинаково хорошо подходит любое равномерное распределение данных. При этом легко обеспечить равномерную загрузку процессоров и организовать параллельное выполнение без пересылок вообще. Структура этих фрагментов примерно такова (пример взят из подпрограммы EULER):

```
      DO 42 J=2,JL
      DO 42 I=2,IL
      WR(I,J,1) = WR(I,J,1)  -DW(I,J,1)
      WR(I,J,2) = WR(I,J,2)  -DW(I,J,2)
      WR(I,J,3) = WR(I,J,3)  -DW(I,J,3)
      WR(I,J,4) = WR(I,J,4)  -DW(I,J,4)
   42 CONTINUE
```

В таких фрагментах на каждой итерации требуются только значения с “одноименных” итераций предыдущих циклов, и при едином для всех циклов распределении итераций по процессорам пересылок не требуется совсем.

Координатное распределение. Для большей части подпрограмм вычислительного ядра лучшим распределением массивов является координатное, т. е. распределение одной или нескольких размерностей ключевых массивов. При таком распределении реализация подобных циклов требует минимального количества пересылок, являющихся регулярными. Например:

```
      DO 20 J=2,JL
      DO 20 I=2,IL
      FS(I,J,N) = DIS2(I,J)*FS(I,J,N)-
     *            DIS4(I,J)*DW(I,J,N)
   20 CONTINUE
      DO 30 J=2,JL
      DO 30 I=2,IL
```

```
      FW(I,J,N) = SFIL*FW(I,J,N)-
     *            FS(I,J,N)+FS(I-1,J-1,N)
   30 CONTINUE
```

Поскольку оба двумерных цикла являются параллельными, пересылки при согласованном распределении основных массивов могут потребоваться только перед выполнением второго цикла, так как в нем на (I, J)-ой итерации требуются элементы массива FS, вычисленные в предыдущем цикле на $(I-1, J-1)$-ой итерации. Такие пересылки потребуются при любом распределении данных, но при координатном распределении они становятся проще.

PSMOO — узкое место для FLO52. На подпрограмму PSMOO на последовательной машине приходится около 20% времени работы всей программы, однако она становится еще более значимой при реализации на параллельном компьютере. Как показывает детальный анализ, не существует такого распределения данных и итераций, при котором возможно выполнение в параллельном режиме всей подпрограммы без пересылок вообще: две ее части обладают параллелизмом по разным размерностям, оставаясь существенно последовательными по другим.

Две части рис. 5 схематично изображают информационные зависимости в пространстве итераций двух частей PSMOO. Первая часть содержит последовательные вычисления вдоль оси I, а вторая — вдоль оси J. Вершины, расположенные на рисунке рядом, а также в соответствующих позициях двух частей рисунка, отвечают вычислению одних и тех же элементов массива. Все элементы массива (кроме последних) в каждой из двух частей PSMOO вычисляются дважды: при вычислении вдоль соответствующей оси и обратно. Использование единого распределения для обеих частей PSMOO приведет к параллельному выполнению только одной половины подпрограммы, следовательно, на любом числе процессоров невозможно будет получить ускорение больше, чем в два раза. По этой причине для параллельного выполнения PSMOO необходимо проводить перераспределение данных, причем перераспределение потребуется перед выполнением той части PSMOO, которая использует иное по отношению к основной схеме распределение, и после нее для восстановления исходного распределения.

Выбор распределения. Возможность использования параллелизма только по одной размерности в одной из наиболее важных подпрограмм (PSMOO) и малое число итераций по другим размерностям определили то, что в программе FLO52 пришлось использовать координатный параллелизм только по самой большой размерности. Вместе с тем, число итераций по этой размерности, меняющееся от 21 до 161 в малом варианте FLO52, оказалось существенным сдерживающим фактором при выполнении программы даже на относительно небольшом числе процессоров, так как уже при 32 ПЭ не всегда хватает итераций для загрузки всех процессоров полезной работой.

4.4. Как распределять итерации циклов?

Для использования выбранной части параллелизма нужно выбрать метод распределения итераций циклов между процессорами вычислительной системы. Возможные распределения — циклически по одной итерации, циклически по несколько итераций, равными максимальными последовательными порциями и т. д., могут минимизировать количество необходимых пересылок. Для этого необходимо выделить такие группы итераций, внутри которых было бы сосредоточено наибольшее число зависимостей по данным. На рис. 6 представлены возможные схемы распределения итераций. Конечно, выбор схемы распределения итераций полностью определяется особенностями каждой конкретной программы. Во FLO52 большинство пересылок связывают соседние итерации распределяемых циклов, поэтому выгоднее использовать блочные распределения, при которых максимальное число последовательных итераций попадает на один ПЭ. Из блочных распределений следует выбрать распределение с максимальной загрузкой. Неравномерность загрузки процессоров не скажется на общем времени выполнения цикла, так как определять время будут процессоры с наибольшим числом итераций. В то же время, при распределении с максимальной загрузкой потребуется меньше пересылок, чем при распределении с равномерной загрузкой, что снижает нагрузку на коммуникационную сеть и, как следствие, позволяет уменьшить общее время выполнения программы. Как показала практика, использование распределения с максимальной за-

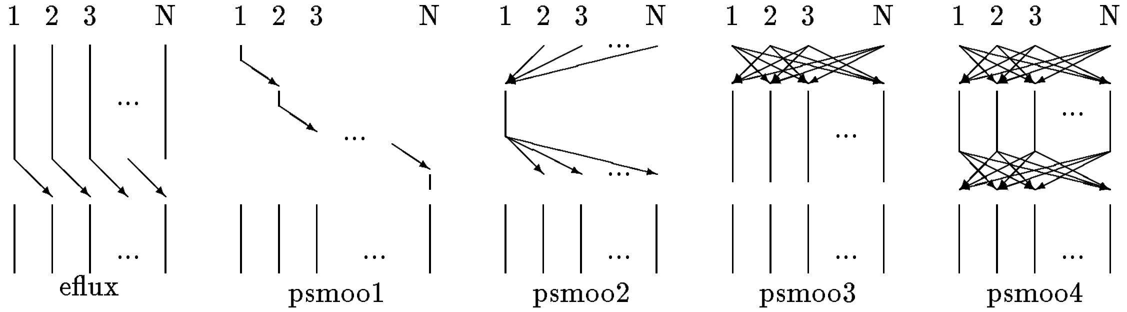

Рис. 7.

Таблица 3.

П/п	Количество ПЭ								
	1	2	4	8	16	32	64	128	256
psmoo1	27.5	31.0	28.8	29.9	34.5	43.2	58.6	81.4	107.6
psmoo2	27.5	35.7	32.9	34.2	39.5	41.7	44.2	50.3	58.2
psmoo3	27.5	30.3	30.3	33.9	37.1	39.7	44.5	51.3	65.6
psmoo4	27.5	22.6	14.2	9.23	7.15	7.03	8.69	12.0	15.0
psmoo5	27.5	166.7	—	—	41.96	—	15.91	10.25	8.22

Времена выполнения вариантов PSMOO, с. (средний вариант).

грузкой дает на реальных подпрограммах выигрыш примерно 10% по сравнению с распределением с равномерной загрузкой.

4.5. Согласование распределений данных и итераций

После принятия решения о методе распределения данных и итераций циклов необходимо согласовать указанные распределения и определить, где в тексте программы требуются пересылки данных. Для иллюстрации структуры пересылок можно использовать схемы параллельного взаимодействия процессоров. Для большинства точек параллельного взаимодействия программы FLO52 характерны одинаковые схемы, определяющие передачу данных либо с каждой итерации цикла на следующую или предыдущую, либо пересылку граничных значений. На рис. 7 приведены обсуждаемые ниже схемы. Вертикальные линии показывают последовательные вычисления на соответствующем ПЭ, а стрелки изображают необходимые пересылки.

Первая схема (eflux) является типичной для FLO52 и описывает ситуацию, при которой каждый процессор осуществляет пересылки только логически следующему ПЭ. Это простая и легко реализуемая операция. Интереснее посмотреть, какие варианты параллельной реализации можно предусмотреть для подпрограмм со структурой вычислений, подобной подпрограмме PSMOO. Не только для FLO52, но и для любой другой программы подобные фрагменты будут серьезным препятствием для получения эффективного параллельного варианта.

В первом варианте (psmoo1 на рис. 7) каждый процессор просчитывает свою порцию итераций, передает данные с последней итерации следующему, после чего тот начинает работать. В каждый момент времени работает только один процессор, однако с ростом числа процессоров растет число необходимых пересылок, и, следовательно, катастрофически растет время исполнения. Времена выполнения этого и всех последующих вариантов PSMOO приводятся в табл. 3. Время исполнения на одном процессоре указано для исходного последовательного варианта программы.

Второй вариант (psmoo2) предусматривает исполнение всех итераций первой половины PSMOO только на первом процессоре. Вначале все ПЭ отсылают первому свои порции исходных данных, а после выполнения первой половины подпрограммы этот процессор рассылает всем остальным их порции результатов. Здесь требуется всего две, хотя и весьма объемных, пересылки, что не позволяет значительно улучшить ситуацию по сравнению с предыдущим вариантом.

В третьем варианте (psmoo3) все итерации первой половины PSMOO выполняются на каждом ПЭ (полное дублирование работы). Для этого перед выполнением подпрограммы требуется единственная рассылка порций исходных данных от каждого ПЭ каждому. Если на малом числе ПЭ этот вариант незначительно превосходит предыдущий, то из-за перегрузки коммуникационной сети на 128 и 256 ПЭ оказывается хуже его.

Четвертый вариант предполагает перераспределение данных так, чтобы использовать в первой части PSMOO параллелизм по одной размерности, а во второй — по другой (psmoo4). Для этого до и после первой части требуется рассылка от каждого процессора каждому небольшой части входных данных. Кроме того, в отличие от предыдущих вариантов, первая часть PSMOO исполняется действительно параллельно, и этот вариант оказался в итоге наиболее удачным.

В пятом варианте реализована параллельная схема вычисления рекурсии первого порядка, подобно схеме каскадного суммирования (psmoo5 — ее схемы нет, но есть результат в табл. 3). При вычислении рекурсии глубины n количество вычислений возросло в $1 + \log_2 n$ раз, поэтому при реализации на малом числе ПЭ время вычисления заметно увеличилось. Вместе с тем, этот вариант отличается от всех предыдущих своей масштабируемостью, так как время его работы постоянно уменьшается с ростом числа ПЭ. В зависимости от числа процессоров можно вызывать либо этот вариант PSMOO, либо предыдущий.

Конечно, не во всех подпрограммах требуются пересылки лишь по регулярной схеме. В частности, подпрограммы ADDX и COLLC осуществляют перераспределение данных при изменении размеров области вычисления, а потому требуют нерегулярных пересылок и являются еще одним узким местом при реализации параллельного варианта FLO52. Если на подпрограмме COLLC получается небольшое ускорение, то соотношение между объемом пересылок и числом арифметических операций не позволяют достичь и этого в подпрограмме ADDX.

4.6. Оптимизация работы программы на каждом процессоре

После получения распределенного варианта программы и оптимизации схем параллельного взаимодействия процессоров необходимо произвести тонкую настройку программы. Она заключается в оптимизации вычислений на каждом процессоре и, в частности, для DEC ALPHA предполагает использование таких особенностей архитектуры, как кэш-память команд, кэш-память данных и раздельные функциональные устройства. На практике это требует увеличения локальности данных, сбалансированности вычислений, оптимизации структуры управления программы, оценки целесообразности использования временных массивов и многих других действий.

4.7. Результаты оптимизации FLO52

Эксперименты производились с программой FLO52 в трех конфигурациях: в малой, средней и большой, которые задают вычисления в областях размером $X \times Y$, $2X \times 2Y$ и $4X \times 4Y$ соответственно. Для организации пересылок использовалась та же библиотека коммуникационных примитивов, что и при оптимизации программы TRFD.

В табл. 4 приведены времена исполнения некоторых распределявшихся подпрограмм в зависимости от числа процессоров для среднего варианта программы FLO52. Время исполнения на одном процессоре указано для исходного последовательного варианта программы. Из таблицы видно, что ускорение получено почти на всех рассмотренных подпрограммах, однако упоминавшиеся в предыдущих пунктах узкие места оказывают значительное влияние на времена работы отдельных фрагментов. Табл. 5 содержит окончательные результаты, полученные на всех трех вариантах.

График на рис. 8 демонстрирует ускорение, полученное на трех вариантах в зависимости от числа используемых ПЭ. Очевидно, что полученная программа обладает хорошими "парал-

Таблица 4.

П/п	Количество ПЭ								
	1	2	4	8	16	32	64	128	256
eflux	68.7	38.2	19.3	10.2	5.74	3.49	2.32	1.69	1.41
dflux	53.6	29.0	15.2	8.27	4.83	3.11	2.20	1.76	1.57
euler	56.1	24.6	13.5	7.88	5.14	3.72	2.95	2.64	2.43
psmoo4	27.5	22.6	14.2	9.23	7.15	7.03	8.69	12.0	15.0
collc	3.93	2.43	1.61	1.43	1.42	1.05	0.73	0.70	0.65
addx	3.79	3.45	2.47	2.16	2.31	2.82	2.80	4.01	4.51

Времена выполнения подпрограмм, с. (средний вариант).

Таблица 5.

Вариант	Количество ПЭ								
	1	2	4	8	16	32	64	128	256
Малый	59.7	37.2	23.4	16.4	13.6	13.0	14.0	13.6	13.7
Средний	235.6	136.3	76.5	46.1	31.9	25.9	24.3	25.4	23.7
Большой	999.3	566.7	301.8	167.0	101.5	70.3	56.3	52.2	57.4

Времена выполнения вариантов FLO52, с.

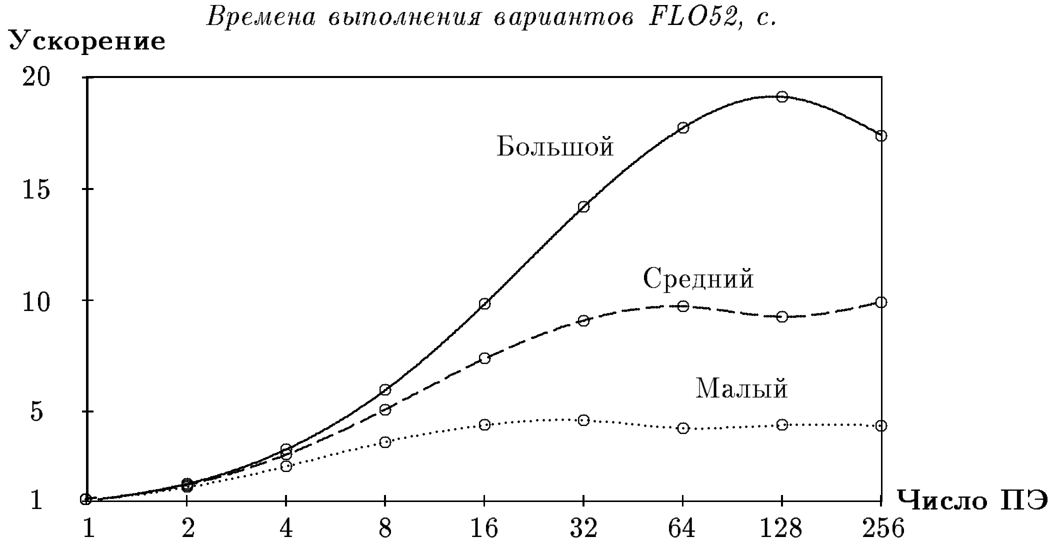

Рис. 8.

лельными" свойствами, так как при достаточном объеме вычислений позволяет получать существенное ускорение на большом числе ПЭ. Относительно небольшое ускорение на малом и среднем вариантах объясняется малым размером задачи, большим числом точек параллельного взаимодействия (более 60 по всей программе, причем суммарное число их прохождений за время работы программы превышает 35000) и наличием сугубо последовательных подпрограмм. Дополнительное ускорение, полученное на 128 и 256 ПЭ на малом и среднем вариантах, объясняется использованием другого алгоритма при реализации подпрограммы PSMOO.

5. ЗАКЛЮЧЕНИЕ

В заключении мы хотели бы отойти от традиционной формы изложения и дать краткую формулировку основных задач, возникших в процессе адаптации программ к архитектуре параллельных компьютеров. Часть из этих задач необходимо решать всегда, другие потребуются только для определенного класса архитектур. Для части из перечисленных проблем уже проводились обширные исследования (например, [6-11]), однако некоторые требуют серьезного математического исследования.

1. Исследование общей структуры программы: выделение основного вычислительного ядра, локализация ввода-вывода, инициализации (необходимо помнить, что реальные приложения могут объединять сотни процедур и десятки тысяч строк текста).
2. Определение потенциального параллелизма фрагмента, а в дальнейшем и выбор той его части, которую разумно использовать для исследуемой программы.
3. Выделение чисто последовательных фрагментов вычислений, распознавание идиом и использование альтернативных алгоритмов для таких фрагментов.
4. Исследование множества возможных распределений данных, поиск распределения, минимизирующего число или объем пересылок, определение и минимизация числа точек перераспределения.
5. Доказательство возможности выполнения традиционных преобразований циклов в процессе преобразования программы: перестановка циклов, слияние, расщепление, объединение и т. д.
6. Минимизация числа и объема временных массивов для оптимизации работы с кэш-памятью и векторными регистрами.
7. Переход от исходной программы, работающей с полными массивами, к программе, обрабатывающей только локальную порцию, распределенную на процессор: изменение размеров массивов и соответствующее преобразование текста программы.
8. Приватизация массивов — выделение массивов, локальных для каждого процессора.
9. Для данного распределения массивов и итераций сгенерировать отвечающий заданному циклу фрагмент параллельной программы, перебирающий локальную порцию итераций и содержащий примитивы коммуникаций типа Send-Receive.
10. Увеличение локальности использования данных программы для оптимизации работы кэш-памяти.

СПИСОК ЛИТЕРАТУРЫ

1. *Voevodin V.V.* Mathematical foundations of parallel computing// Computer Science Series, Vol.33, World Sci. Publ. Co., Singapore, 1992.

2. *Воеводин Вл.В.* Теория и практика исследования параллелизма последовательных программ// Программирование. 1992. N^o 3. с.38-53.

3. *Cybenko G., Kipp L., Pointer L., and Kuck D.* Supercomputer performance evaluation and the Perfect Benchmarks, Tech. Rep. 965, CSRD, Univ. of Illinois at Urbana, Technical Report, 1990.

4. *Grassl C.M.* Parallel Performance of Applications on Supercomputers// Parallel Computing v.17 (1991), pp.1257-1273.

5. *Воеводин Вл.В.* Легко ли получить обещанный гигафлоп? //Программирование. 1995. N^o 4. С.13-23.

6. *Wolfe M.J.* Optimizing Supercompilers for Supercomputers, The MIT Press, Cambridge, Massachusetts, 1989.

7. *Polychronopoulos C.D. et al.* Parafrase–2: An Environment for Parallelizing, Partitioning, Synchronizing, and Scheduling Programs on Multiprocessors// Proc. of the 1989 Int. Conf. on Parallel Processing, St. Charles, Illinois, 1989.

8. *Feautrier P.* Some efficient solutions to the affine scheduling problem, part I/II. //Tech. Rep. 92.28/78, IBP/MASI, France, 1992.

9. *Kelly W., Pugh W.* A Framework for Unifying Reordering Transformations. Tech. Rep. CS-TR-2995.1. Dept. of Comp. Science, University of Maryland, College Park, 1993.

10. *Maydan D., Amarasinghe S., Lam M.* Array Data-flow Analysis and Its Use in Array Privatization. // Proc. of the 1993 ACM Conference on Principles of Programming Languages, 1993.

11. *Torrellas J., Lam M., Hennessy J.* False Sharing and Spatial Locality in Multiprocessors Caches// IEEE Trans. on Computers, Vol.43, No.6, June, 1994, p.651-663.