На правах рукописи

Клинов Максим Сергеевич

АВТОМАТИЧЕСКОЕ РАСПАРАЛЛЕЛИВАНИЕ НЕКОТОРОГО КЛАССА ФОРТРАН-ПРОГРАММ. ОТОБРАЖЕНИЕ НА КЛАСТЕР

Специальность 05.13.11 – математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей

А в т о р е ф е р а т

диссертации на соискание ученой степени

кандидата физико-математических наук

Москва – 2009

Работа выполнена в Институте прикладной математики
им. М.В. Келдыша РАН.

Научный руководитель – доктор физико-математических наук, профессор
Крюков Виктор Алексеевич

Официальные оппоненты:

– доктор физико-математических наук, доцент
Якобовский Михаил Владимирович,

– кандидат физико-математических наук
Антонов Александр Сергеевич

Ведущая организация: Институт системного программирования РАН

Защита состоится «1» декабря 2009 г. в 11 часов на заседании Диссертационного совета Д 002.024.01 в Институте прикладной математики им. М.В. Келдыша РАН по адресу: 125047, Москва, Миусская пл., 4.

С диссертацией можно ознакомиться в библиотеке Института прикладной математики им. М.В. Келдыша РАН.
Автореферат разослан «29» октября 2009 г.

Ученый секретарь диссертационного совета,
доктор физико-математических наук Т.А. Полилова

Общая характеристика работы

Объект исследования и актуальность темы

Разработка программ для высокопроизводительных кластеров и других параллельных систем с распределенной памятью продолжает оставаться исключительно сложным делом, доступным узкому кругу специалистов и крайне трудоемким даже для них. Основная причина – это низкий уровень современной технологии автоматизации разработки параллельных программ. В настоящее время практически все параллельные программы для кластеров разрабатываются с использованием низкоуровневых средств передачи сообщений (MPI). MPI-программы трудно разрабатывать, сопровождать и повторно использовать при создании новых программ. Поэтому вполне естественно, что прикладной программист хотел бы получить либо инструмент, автоматически преобразующий его последовательную программу в параллельную программу для кластера, либо высокоуровневый язык параллельного программирования, обеспечивающий эффективное использование современных параллельных систем.

Однако проведенные в 90-х годах активные исследования убедительно показали, что полностью автоматическое распараллеливание для таких ЭВМ реальных производственных программ возможно только в очень редких случаях.

Сначала очень высокие требования к эффективности выполнения параллельных программ, а затем и изменения в архитектуре параллельных ЭВМ привели к тому, что до настоящего времени так и нет и общепризнанного высокоуровневого языка параллельного программирования, позволяющего эффективно использовать возможности современных ЭВМ.

Все больше специалистов предлагают использовать языки с неявным параллелизмом, при программировании на которых не требуется знать архитектуру параллельной ЭВМ. В качестве таких языков особенно привлекательно использовать Фортран и Си/Си++, поскольку их в основном и используют программисты при решении задач, наиболее остро требующих распараллеливания.

Написанные на таких языках последовательные программы, свойства которых либо автоматически извлечены посредством анализа текста программ или при их выполнении, либо описаны программистом с помощью специальных аннотаций, могут затем полностью автоматически распараллеливаться и эффективно выполняться на параллельных ЭВМ разной архитектуры.

Поэтому в настоящее время снова резко возрос интерес к исследованиям в области автоматического распараллеливания последовательных программ.

Цель работы

Задача автоматического распараллеливания последовательной программы распадается на две подзадачи – анализ последовательной программы и ее преобразование в параллельную программу.

Целью данной диссертационной работы являлись:

- разработка алгоритма моделирования параллельного выполнения программ на языке Фортран-DVM и прогнозирования его характеристик;
- разработка алгоритма преобразования последовательной программы на языке Фортран 77 (для класса задач, при решении которых используются разностные методы на статических структурных сетках) в параллельную программу для кластера на языке Фортран-DVM.

Научная новизна работы

1. Разработан алгоритм моделирования параллельного выполнения DVM-программы (программы на языке Фортран-DVM или Си-DVM), позволяющий прогнозировать время выполнения и другие характеристики эффективности параллельной программы
2. Разработан алгоритм автоматического отображения последовательной программы на кластер, состоящий из алгоритмов
   - отбора наиболее перспективных вариантов распределения данных
   - распределения вычислений и организации коммуникаций
   - выбора наиболее эффективной схемы распараллеливания программы посредством моделирования ее параллельного выполнения

Проведенные эксперименты с тестами и реальными приложениями показали, что для заданного класса задач можно писать программы на языках Фортран и Си, которые будут автоматически распараллелены и эффективно выполнены на кластерах.

Практическая значимость

На базе разработанного алгоритма моделирования параллельного выполнения DVM-программы создан инструмент (DVM-предиктор), позволяющий проводить на инструментальной ЭВМ предварительную отладку эффективности выполнения DVM-программы на кластерах с заданными характеристиками узлов и коммуникационной системы. Этот инструмент входит в состав DVM-системы и используется на факультете ВМК МГУ при проведении практикума по технологиям параллельного программирования.

Алгоритм автоматического отображения последовательной программы на кластер был реализован в экспериментальной версии распараллеливающего компилятора с языка Фортран (компилятора ПАРФОР), созданного в рамках программы Союзного государства “ТРИАДА”, и может служить базой для создания алгоритмов отображения последовательных программ на параллельные ЭВМ разной архитектуры.

Апробация работы и публикации

Основные результаты диссертации были доложены на российских и международных научных конференциях и семинарах:

1. Всероссийской научной конференции “Научный сервис в сети Интернет: технологии параллельного программирования”, сентябрь 2006 г., г. Новороссийск.
2. Всероссийской научной конференции “Научный сервис в сети Интернет: многоядерный компьютерный мир”, сентябрь 2007 г., г. Новороссийск.
3. Научной конференции “Тихоновские чтения”, октябрь 2008 г., г. Москва.
4. Международной научной конференции "Параллельные вычислительные технологии" (ПаВТ'2009), март 2009 г., г. Нижний Новгород.
5. XIV Байкальской Всероссийской конференции "Информационные и математические технологии в науке и управлении", июль 2009 г., г. Иркутск.

Имеется 5 публикаций, из которых две [3,4] – в журналах из списка ВАК.

Структура и объем работы

Диссертация состоит из введения, шести глав, заключения, списка литературы (29 наименований) и двух приложений. Общий объем работы составляет 108 страниц, работа содержит 8 иллюстраций и 10 таблиц.

Содержание работы

Во **введении** обосновывается актуальность темы исследования, формулируются цель и задачи диссертации. Приводятся основные результаты работы, их практическая значимость. Перечисляются выступления на конференциях и семинарах, кратко излагается содержание работы.

В **первой главе** приводится обзор систем автоматического и автоматизированного распараллеливания программ.

Появление многопроцессорных ЭВМ с распределенной памятью значительно расширило возможности решения больших задач, однако резко усложнило разработку программ. Программист должен распределить данные между процессорами, а программу представить в виде совокупности процессов, взаимодействующих между собой посредством обмена сообщениями.

Поэтому вполне естественно, что прикладной программист хотел бы получить инструмент, автоматически преобразующий его последовательную программу в параллельную программу для кластера, поскольку, скажем, для векторных процессоров подобные средства широко и успешно использовались. Однако автоматическое распараллеливание для кластеров гораздо сложнее по следующим причинам.

Во-первых, поскольку взаимодействие процессоров через коммуникационную систему требует значительного времени, то вычислительная работа должна распределяться между процессорами крупными порциями.

Для автоматического распараллеливания на векторных процессорах (векторизации) достаточно было проанализировать на предмет возможности параллельного выполнения (замены на векторные операции) только самые внутренние циклы программы.

В случае многопроцессорных ЭВМ с общей памятью (мультипроцессоров) приходилось уже анализировать объемлющие циклы для нахождения более крупных порций работы, распределяемых между процессорами.

Укрупнение распределяемых порций работы требует анализа более крупных фрагментов программы, обычно включающих в себя вызовы различных процедур. Это, в свою очередь, требует сложного межпроцедурного анализа. Поскольку в реальных программах на языке Фортран могут использоваться конструкции, статический анализ которых принципиально невозможен (например, косвенная индексация элементов массивов), то с увеличением порций распределяемой работы увеличивается вероятность того, что распараллеливатель откажется распараллеливать те конструкции, которые на самом деле допускают параллельное выполнение.

Во-вторых, в отличие от мультипроцессоров, на системах с распределенной памятью необходимо произвести не только распределение вычислений, но и распределение данных, а также обеспечить на каждом процессоре доступ к удаленным данным, т.е. данным, расположенным на других процессорах. Для обеспечения эффективного доступа к удаленным данным требуется производить анализ индексных выражений не только внутри одного цикла, но и между разными циклами. К тому же, недостаточно просто обнаруживать факт наличия зависимости по данным, а требуется определить точно тот сегмент данных, который должен пересылаться с одного процессора на другой.

В третьих, распределение вычислений и данных должно быть произведено согласованно. Несогласованность распределения вычислений и данных приведет к значительному увеличению времени коммуникаций. Согласование распределений вычислений и данных требует тщательного анализа всей программы, и любая неточность анализа может привести к катастрофическому замедлению выполнения программы.

Попытки разработать автоматически распараллеливающие компиляторы для параллельных ЭВМ с распределенной памятью, проведенные в 90-х годах (например, Paradigm, APC), привели к пониманию того, что полностью автоматическое распараллеливание для таких ЭВМ реальных производственных программ возможно только в очень редких случаях. В результате, исследования в области автоматического распараллеливания для параллельных ЭВМ с распределенной памятью практически были прекращены.

Исследователи сосредоточились на двух направлениях:

- разработка высокоуровневых языков параллельного программирования (HPF, OpenMP-языки, DVM-языки, CoArray Fortran, UPC, Titanium, Chapel, X10, Fortress);

- создание систем автоматизированного распараллеливания (CAPTools/Parawise, FORGE Magic/DM, BERT77), в которых программист активно вовлечен в процесс распараллеливания.

Однако изначально высокие требования к эффективности выполнения параллельных программ и последовавшие затем изменения в архитектуре параллельных ЭВМ привели к тому, что в настоящее время нет ни одного общепризнанного высокоуровневого языка параллельного программирования для современных кластеров.

В системах автоматизированного распараллеливания на программиста стали возлагаться ответственные решения не только по уточнению свойств его последовательной программы (результатов анализа), но и по ее отображению на параллельную ЭВМ. Это является серьезным недостатком, вызывающим огромные трудности для внедрения таких систем.

В настоящее время имеется только одна развивающаяся система автоматизированного распараллеливания для кластеров – Parawise (системы FORGE Magic/DM и BERT77 уже не развиваются и не поддерживаются). Только для нее имеется информация о применимости для реальных Фортран-приложений и эффективности выполнения распараллеленных программ.

Система Parawise является коммерческой системой, созданной компанией Parallel Software Products совместно с NASA Ames на базе системы CAPTools, разработанной в Лондонском университете Гринвича в середине 90-х годов.

Главным достоинством системы Parawise являются развитые возможности межпроцедурного анализа, которые создавались в течение многих лет. Главный недостаток – программист должен сам принимать ответственное решение о том, какие массивы и каким образом требуется распределить между процессорами.

Отсутствие общепризнанного высокоуровневого языка параллельного программирования для современных кластеров привело к тому, что многие специалисты стали предлагать использовать языки с неявным параллелизмом, программирование на которых не требует знания архитектуры параллельной ЭВМ. Такими языками являются широко распространенные языки Фортран и Си/Си++. Это означает, что задача автоматического распараллеливания снова стала вызывать огромный интерес, но она ставится теперь иначе – не пытаться полностью автоматически и эффективно распараллеливать любые существующие последовательные программы, а позволить создавать на языках последовательного программирования программы, автоматически преобразуемые в эффективные параллельные программы.

Такая постановка задачи автоматического распараллеливания вызывает изменение подхода к автоматизации распараллеливания программ: использовать диалог с программистом только для уточнения свойств последовательной программы, а выбор наилучших решений по ее распараллеливанию осуществлять полностью автоматически, без участия программиста.

Программист участвует, конечно, еще в проведении модификаций своей последовательной программы с целью повышения ее эффективности (в данном случае эффективности ее автоматического распараллеливания), но это свойственно программированию на любом языке.

В настоящее время можно утверждать, что для кластеров не существует систем, в основе которых есть автоматически распараллеливающий компилятор последовательных программ на языках Фортран или Си, и где диалог строится на указанном принципе.

Вторая глава посвящена описанию схемы построения автоматически распараллеливающего компилятора с языка Фортран (ПАРФОР), функций блока преобразования последовательной Фортран-программы в параллельную программу на языке Фортран-DVM, входной и выходной информации этого блока.

Автоматически распараллеливающий компилятор состоит из блоков анализа последовательной программы (анализаторов) и блоков ее преобразования в параллельную программу (экспертов). Их взаимодействие осуществляется через специальную базу данных (БД).

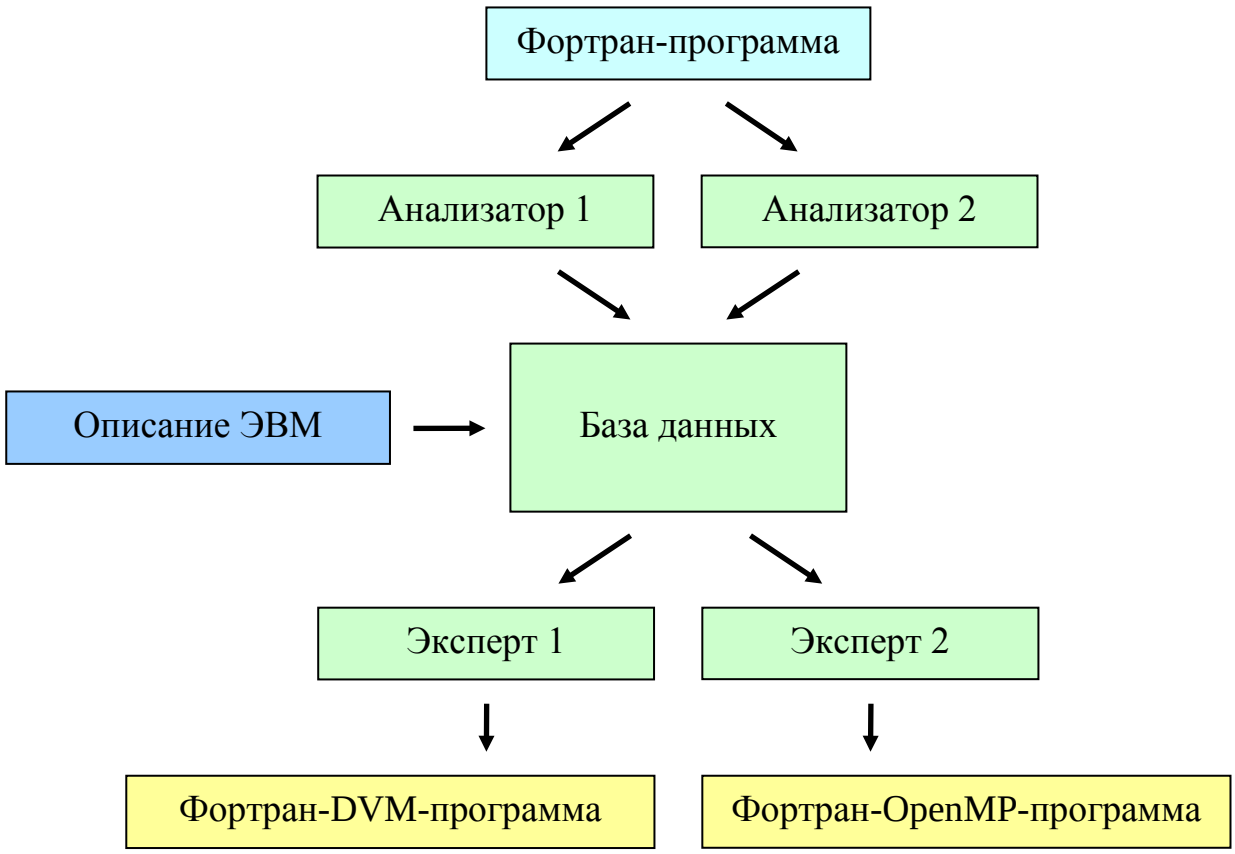

Рис. 1. Схема работы компилятора ПАРФОР.

Анализатор выявляет свойства программы, влияющие на ее распараллеливание. Допускается использование нескольких анализаторов для дополнения БД более точными результатами анализа. Анализаторы могут

различаться как методами анализа (статические или динамические), так и алгоритмами (более точными или более быстрыми).

Эксперты могут отличаться алгоритмами отображения на параллельные ЭВМ разной архитектуры и используемыми языками параллельного программирования. Функциями эксперта являются:

- построение схем распараллеливания
- оценка схем для заданной ЭВМ
- получение текста параллельной программы для лучшей схемы

Под *схемой распараллеливания* понимается набор правил преобразования последовательной программы в параллельную. ЭВМ задается своей архитектурой (мультипроцессор, кластер или SMP-кластер), числом процессоров, их производительностями и параметрами коммуникационной среды (например, латентностью и пропускной способностью).

База данных хранит всю необходимую информацию для работы компонент системы, тем самым обеспечивая их взаимодействие.

На вход эксперту поступает следующая информация:

- описание программных единиц программы
- описание переменных (скалярных, массивов, формальных параметров, common-блоков)
- дерево циклов
- граф переходов между операторами
- вызовы подпрограмм и фактические параметры
- особенности операторов ввода-вывода
- информация о возможности распараллеливания циклов
    - какие есть в программе зависимости между витками цикла. Для витков, между которыми есть зависимость, необходимо обеспечить определенный порядок выполнения и коммуникации, чтобы результат выполнения был корректным. Очень часто этот порядок сводится к последовательному выполнению
    - какие зависимости вызваны использованием приватных переменных (рабочих переменных, зависимость по которым не мешает выполнять цикл параллельно, если использовать несколько экземпляров таких переменных)
    - какие зависимости являются редукционными (зависимость по переменным, которая не мешает выполнять цикл параллельно, если использовать несколько экземпляров таких переменных и производить объединение их значений на выходе из цикла. Например, когда в цикле вычисляется сумма элементов массива)
- описание ЭВМ
- параметры задачи (значения переменных, которые определяют размеры массивов и количество витков циклов).

Результаты анализа могут быть скорректированы программистом с помощью специальных аннотаций, вставляемых в текст последовательной программы. Например, в случае косвенной индексации элементов массива статический анализатор должен сообщить о возможной зависимости между витками цикла, а программист может указать, что зависимости нет.

Выходом эксперта является текст параллельной программы.

Текущая версия компилятора ПАРФОР состоит из трех компонентов:

1. статический анализатор Фортран-программ ВерБа,
2. DVM-эксперт (преобразует исходные Фортран-программы в параллельные программы для кластера на языке Фортран-DVM),
3. Система управления базой данных компилятора.

Компилятор ПАРФОР на данный момент ориентирован на автоматическое распараллеливание программ из класса задач, при решении которых используются разностные методы на статических структурных сетках. К тому же эти программы должны удовлетворять следующим требованиям:

- быть написаны на языке Фортран 77;
- не содержать процедуры или допускать их инлайн-подстановку.

Эти ограничения были приняты, потому что статический анализатор компилятора ПАРФОР работает только с программами на языке Фортран 77 и не осуществляет межпроцедурный анализ зависимостей.

Выбор языка параллельного программирования Фортран-DVM для отображения последовательных программ на кластер обусловлен следующими причинами.

Язык Фортран-DVM является расширением стандартного языка Фортран директивами параллелизма. В результате компиляции программа на языке Фортран-DVM преобразуется в программу на стандартном языке Фортран, которая выполняется на процессорах кластера и использует библиотеку MPI для организации взаимодействия между процессорами. Компилятор с языка Фортран-DVM применяет готовые приемы распараллеливания и освобождает программиста от рутинной работы, приводящей к многочисленным ошибкам распараллеливания. Использование языка Фортран-DVM значительно упрощает преобразование последовательной программы в параллельную.

К тому же для выбранного языка есть развитые средства функциональной отладки и отладки эффективности, что облегчает поиск ошибок в алгоритмах DVM-эксперта.

В **третьей главе** описывается алгоритм моделирования параллельного выполнения DVM-программы. В процессе его работы вычисляются характеристики эффективности параллельного выполнения DVM-программы (прогнозируемые характеристики).

Алгоритм моделирования изначально создавался для отладки эффективности выполнения DVM-программ, основанной на вычислении характеристик для определенных частей программы (интервалов выполнения). Вообще, вычисление характеристик эффективности может осуществляться как

по результатам реальных запусков программы на целевой ЭВМ, так и путем моделирования параллельного выполнения программы для целевой ЭВМ. Под целевой ЭВМ будем понимать машину, на которой планируется запускать программу на счет. Инструментальная машина используется для отладки программы.

Вычисление характеристик эффективности по результатам реальных запусков имеет ряд недостатков: для сбора необходимой информации используется высокопроизводительная ЭВМ, запуски программ обычно связаны с большими временами ожидания их результатов. Но самый главный недостаток - характеристики выполнения реальных запусков нестабильны (различаются от запуска к запуску, особенно при использовании большого количества узлов кластера).

Для моделирования используется только инструментальная ЭВМ. Путем прогнозирования получаются более стабильные характеристики, чем при реальных запусках, что позволяет скорее увидеть влияние модификаций, направленных на повышение эффективности программы.

Набор характеристик эффективности выполнения для каждого интервала выполнения выглядит следующим образом:

- **время выполнения**, равное максимальному значению среди времен выполнения данного интервала на каждом процессоре;
- **общее время** использования процессоров, то есть произведение времени выполнения на число процессоров;
- **полезное время** – предполагаемое время выполнения на одном процессоре, равное сумме следующих характеристик:
  - **полезное процессорное время**;
  - **полезные системные расходы**;
  - **время ввода-вывода** - сумма времен выполнения операций ввода-вывода, без учета затрат на коммуникации;
- **коэффициент эффективности**, равный отношению полезного времени к общему времени;
- **потерянное время**, равное сумме следующих характеристик:
  - **время потерь из-за недостаточного параллелизма** (дублирования вычислений на нескольких процессорах);
  - **время коммуникаций**;
  - **время простоев** процессоров;
- **разбалансировка загрузки** процессоров при параллельных вычислениях - это потенциальные потери, которые могут быть вызваны неравномерностью загрузки процессоров;
- **общее время перекрытия** обменов сообщениями и вычислений.

Алгоритм моделирования связан с особенностями DVM-системы.

Как было сказано выше, язык Фортран-DVM представляет собой язык Фортран, расширенный спецификациями параллелизма. Эти спецификации можно условно разделить на три подмножества:

- распределение данных;

- распределение вычислений;
- спецификация удаленных данных (данных с других процессоров, необходимых для выполнения вычислений на каждом процессоре).

На выходе компилятора Фортран-DVM получается программа на языке Фортран с вызовами функций системы поддержки параллельного выполнения (библиотеки Lib-DVM), основные функции которой:

- Построение абстрактной параллельной машины (идеальной машины, наиболее подходящей для выполнения программы).
- Отображение абстрактной параллельной машины на заданную при запуске решетку процессоров.
- Создание и уничтожение распределенного массива.
- Отображение распределенного массива на абстрактную параллельную машину.
- Отображение параллельного цикла на абстрактную параллельную машину.
- Отображение параллельных задач на абстрактную параллельную машину.
- Выполнение редукционных операций.
- Обновление теневых граней распределенных массивов (теневые грани – это расширения локальной части массива для организации доступа к расположенным на соседних процессорах элементам этого массива).
- Создание и загрузка буферов для доступа к удаленным данным.

На вход алгоритму моделирования параллельного выполнения DVM-программы поступает последовательность вызовов функций Lib-DVM, которая не зависит от количества используемых процессоров и может быть получена при выполнении на одном процессоре (посредством трассировки). Эта последовательность характеризует не только работу, выполняемую системой поддержки, но и вычисления, выполняемые пользовательской программой между вызовами функций Lib-DVM.

Алгоритм моделирования выполнения DVM-программы состоит из алгоритмов моделирования параллельных и непараллельных участков программы, а также из алгоритмов моделирования коммуникационных операций.

Моделирование выполнения непараллельных участков заключается в добавлении времени работы этого участка к временам работы всех процессоров, которые выполняют данных участок. Наличие непараллельных участков в программе приводит к увеличению значения характеристики “недостаточный параллелизм”, поскольку одни и те же вычисления выполняются на нескольких процессорах.

Для моделирования выполнения параллельных циклов определяется количество итераций цикла, выполняемое каждым процессором, и

осуществляется соответствующее разделение общего времени работы цикла между процессорами.

В DVM-программе могут использоваться несколько видов коммуникационных операций:

- обмен данными через теневые грани массивов,
- редукционные операции,
- копирование секций массивов,
- удаленный доступ к секциям массива,
- обмен данными при организации конвейерного выполнения витков цикла.

Для моделирования этих операций используются следующие алгоритмы.

Для операций обновления теневых граней определяется размер грани (в байтах), заполняется матрица пересылок, элементы которой задают объем пересылки с одного процессора на другой. Затем моделируется запуск пересылок данных по всей матрице, определяются загрузки коммуникационных каналов кластера и времена завершения обновления теневых граней для каждого процессора в отдельности. В момент завершения обновления теневых граней для каждого процессора будет вычислено время, потраченное на ожидание, и время совмещения коммуникаций с предшествующими вычислениями.

Для редукционных операций моделируется сбор на одном из процессоров значений редукционных переменных от всех процессоров, участвующих в операции редукции. Время, затрачиваемое на вычисление редукционного значения по собранным значениям, является незначительным и считается нулевым. После этого моделируется рассылка полученного значения всем процессорам и вычисляется время завершения данной редукционной операции на каждом процессоре.

При копировании определяется объем передаваемых данных и заполняется матрица пересылок. Дальнейшее моделирование похоже на моделирование обновления теневых граней.

Моделирование удаленного доступа к секциям массива подобно моделированию копирования.

Конвейер используется для распараллеливания циклов с регулярными зависимостями между витками, когда длина зависимости не более некоторой константы. Главной задачей алгоритма моделирования конвейера является определение количества витков цикла (кванта конвейера), которые будут выполняться процессорами на каждом шаге конвейера. Размер кванта конвейера выбирается таким образом, чтобы время выполнения цикла стало минимальным. Этот размер зависит от времени выполнения одного витка цикла, количества витков, количества процессоров, объема передаваемой между процессорами информации и характеристик коммуникационной среды.

Во всех описанных выше операциях вычисление времен коммуникаций опирается на следующую многоуровневую модель. На самом нижнем уровне находятся ядра одного процессора. На более высоком уровне находятся ядра

процессоров, находящихся на одном узле кластера. Еще выше ядра процессоров разных узлов кластера и так далее. Количество уровней не ограничено. Каждый уровень имеет несколько каналов связи с заданной для каждого латентностью и временными расходами на передачу каждого байта информации.

Необходимо заметить, что точное прогнозирование времени параллельного выполнения практически невозможно для современных ЭВМ. Но описанные выше алгоритмы моделирования позволяют получать характеристики, с помощью которых можно оценить влияние тех или иных модификаций программы, и сравнить эффективность предлагаемых схем распараллеливания.

Четвертая глава посвящена описанию алгоритма построения схем распараллеливания, состоящего из алгоритма отбора наиболее перспективных вариантов распределения данных, алгоритма распределения вычислений и организации коммуникаций.

Подход компилятора ПАРФОР основан на построении множества наиболее перспективных (лучших по грубым оценкам) схем распараллеливания и выборе наилучшей схемы (с наименьшим прогнозируемым временем выполнения по более точным оценкам).

Такой подход обладает важными достоинствами. Во-первых, по мере увеличения производительности инструментальных ЭВМ можно увеличивать и количество перспективных схем. Во-вторых, прогнозирование времен и других характеристик выполнения программы очень полезно для объяснения программисту, почему выбрана та или иная схема распараллеливания.

Важнейшим условием применения этого подхода является отбор приемлемого количества перспективных схем распараллеливания, чтобы моделирование могло быть выполнено за приемлемое время.

Отбор схем начинается с отбора вариантов распределения данных.

Отбор перспективных вариантов распределения данных

Перспективность того или иного варианта распределения данных определяется эффективностью последующего распределения вычислений (распределения витков циклов между процессорами) и организации коммуникаций.

Для распределения витков цикла между процессорами требуется, чтобы все элементы массивов, которые модифицируются на каком-то витке цикла, находились на процессоре, который выполняет этот виток. Такое требование было принято при разработке языка Фортран-DVM, и позволяет компилятору не заботиться о выполнении правила собственных вычислений (когда процессор производит вычисления только тех данных, которых у него имеются), а переложить эту заботу на плечи программиста. Для того, чтобы выполнить это требование, некоторые элементы некоторых модифицируемых массивов можно размножить (завести копии одного и того же элемента массива на нескольких процессорах). Но тогда придется дублировать вычисления

(повторять одни и те же вычисления на нескольких процессорах) или на выходе из цикла приводить все копии в согласованное состояние.

Если все модифицируемые элементы массивов каждого витка цикла не удается расположить на одном процессоре, то эксперт не будет распараллеливать такой цикл, и тогда все витки этого цикла будут выполняться на каждом процессоре. Такой способ выполнения цикла крайне неэффективен.

Поясним выше сказанное на примере.

Пример распределения данных и вычислений.

```
      DO  1  I = 2, 99
      DO  1  J = 2, 99
          B(I,J) = A(I,J)
          D(I)=  MAX(D(I), A(I,J))
1     CONTINUE
```

Рассмотрим для данного примера использование двумерного распределения витков цикла на 4 процессора. Тогда

- первый процессор будет выполнять витки по I от 2 до 50 и J от 2 до 50,
- второй процессор по I от 2 до 50, а по J от 51 до 99,
- третий по I от 51 до 99, а по J от 2 до 50,
- четвертый по I от 51 до 99, а по J от 51 до 99.

Для эффективного параллельного выполнения такого цикла все процессоры должны иметь данные, в которые необходимо произвести присваивания, а именно:

- на первом процессоре элементы массива B с индексами по первому измерению от 2 до 50 и по второму измерению от 2 до 50, и элементы массива D с индексами от 2 до 50;
- на втором процессоре элементы массива B с индексами по первому измерению от 2 до 50 и по второму измерению от 51 до 99, и элементы массива D с индексами от 2 до 50;
- на третьем процессоре элементы массива B с индексами по первому измерению от 51 до 99 и по второму измерению от 2 до 50, и элементы массива D с индексами от 51 до 99;
- на четвертом процессоре элементы массива B с индексами по первому измерению от 51 до 99 и по второму измерению от 51 до 99, и элементы массива D с индексами от 51 до 99.

Заметим, что элементы массива D размножены.

Кроме того, для распределения витков цикла между процессорами требуется, чтобы в обращениях к модифицируемым элементам массивов использовались индексные выражения, которые линейно зависят от одной итерационной переменной – параметра цикла. В таком цикле не должно быть также операторов ввода-вывода и операторов выхода из цикла.

При распределении данных и вычислений необходимо минимизировать объем удаленных данных для каждого процессора, пытаясь распределять на

каждый процессор все те данные, которые ему нужны для выполнения витков цикла. Для данного примера можно распределить массив A по процессорам так же как и массив B, тогда для каждого процессора не будет удаленных данных.

Для многомерных массивов обычно используется многомерное распределение их элементов по процессорам, а процессоры тогда организуются логически в многомерную решетку процессоров. Это позволяет использовать большее количество процессоров по сравнению с одномерным распределением, а также часто позволяет уменьшить объем коммуникаций.

Покажем, что количество всевозможных вариантов распределения данных очень велико. Пусть сумма всех измерений по всем массивам равна Nd, тогда если строить варианты только по правилу, распределять ли очередное измерение или не распределять, то количество вариантов будет 2^{Nd}. При использовании многомерных распределений получается, что для каждого измерения массива нужно указать, на какое измерение процессорной решетки его распределить. Поэтому, если размерность процессорной решетки равна K, то каждое из M измерений массива можно либо не распределять, либо распределить не более чем K способами, поскольку разные измерения одного массива не могут быть распределены на одно измерение процессорной решетки. Общее количество таких вариантов распределения всех массивов можно грубо оценить как $(K+1)^{Nd}$. Т.е. количество вариантов огромно, и необходимо применение эвристик для его сокращения.

Главным методом сокращения количества вариантов распределения данных является связывание измерений массивов в группы и построение вариантов распределения данных не посредством перебора различных способов распределения каждого измерения массива, а путем перебора различных способов распределения групп. Измерения массивов одной группы будут распределяться на одно и то же измерение процессорной решетки.

Для связывания измерений в группы производится анализ индексации элементов разных массивов в витках циклов, в результате которого формулируются требования к взаимному расположению массивов.

Связывание измерений массивов в группы можно рассматривать, например, как распознавание взаимосвязей между разными физическими величинами, относящимися к одной точке физического пространства. Эти взаимосвязи проявляются в индексации разных измерений массивов внутри одного цикла. Последующее распределение данных и вычислений будет соответствовать часто применяемому программистами методу распараллеливания по пространству.

Также при построении групп применяется другая эвристика: не добавлять измерение массива ни в одну группу, если элементы этого измерения не индексируются нигде по переменным цикла. В этом случае распределять такое измерение имеет смысл только для сокращения памяти, а никакого ускорения выполнения программы не будет, поскольку нельзя будет соответствующим образом распределить витки какого-либо цикла. Такая ситуация часто возникает в программах на языке Фортран 77, поскольку там нет другого

способа объединить разные физические величины, относящиеся к одной точке пространства, кроме выделения для них специального измерения массива.

Отметим, что эти эвристики приводят к существенному сокращению количества вариантов распределения данных. Например, в тестах NAS (LU, BT и SP) Nd изменяется от 64 до 76. А при объединении в группы, получается 5-6 групп измерений, что позволяет уменьшить количество вариантов с 3^{76} до 3^{6} при размерности процессорной решетки К=2.

При связывании измерений массивов в группы, естественно, могут быть обнаружены противоречия, требующие своего разрешения. Существуют два вида таких противоречий.

Во-первых, два разных измерения одного массива не могут быть отображены на одно и то же измерение процессорной решетки. Такой способ распределения смысла не имеет и поэтому запрещен и в DVM-системе и в других языках, таких как HPF.

Во-вторых, взаимное расположение элементов двух массивов может быть задано только один раз, поскольку динамическое перераспределение массивов не поддерживается в текущей версии DVM-эксперта. Поэтому наличие нескольких различающихся требований к взаимному расположению элементов двух массивов со стороны одного и того же цикла или нескольких разных циклов является противоречием, которое необходимо устранить.

При разрешении таких противоречий используются оценки важности удовлетворения требований к взаимному расположению массивов, например, сколько времени будет потеряно из-за отказа от распараллеливания цикла или на дополнительные коммуникации.

Пример возможных противоречий и способов их разрешения

```
**********Гнездо циклов 1 - циклы 1i и 1j
      DO  1  I = 2, 99
      DO  1  J = 2, 99
          B(I,J) = A(I,J)
          D(I)=  MAX(D(I), A(I,J))
1     CONTINUE
********** Гнездо циклов 2 - циклы 2i и 2j
      DO  2  I = 2, 99
      DO  2  J = 2, 99
          A(I,J) = B(I-1,J)
2     CONTINUE
********** Гнездо циклов 3 - циклы 3i и 3j
      DO  3  I = 2, 99
      DO  3  J = 2, 99
          D(J)=  MAX(D(J), A(I,J))
3     CONTINUE
```

Цикл 1i требует расположить I-ую строку матрицы A вместе с I-ой строкой матрицы B и вместе с I-ым элементом вектора D. Обозначим эти

требования “A(I,)-B(I,)”, “A(I,)-D(I)”, “D(I)-B(I,)”. При этом требование D(I)-B(I,) является “жестким” (имеющим очень большой вес) - если оно не будет выполнено, то цикл должен выполнятся последовательно, поскольку при его параллельном выполнении будет нарушено правило собственных вычислений.

Цикл 1j требует расположить J-ый столбец матрицы A вместе с J-ым столбцом матрицы B. Требований к расположению элементов вектора D от этого цикла нет, поскольку вектор D не индексируется переменной цикла J.

Противоречий в требованиях циклов 1i и 1j нет.

Цикл 2i требует расположить I-ую строку матрицы A вместе с (I-1)-ой строкой матрицы B. Это требование противоречит требованию цикла 1i – нельзя задать два разные взаимные расположения строк двух массивов. Нарушение этого требования в этом цикле приведет к коммуникациям и пересылке элементов массива B. К такого рода коммуникациям приведет и нарушение требования в цикле 1i. Поэтому победит то взаимное расположение, которое приведет к минимальным затратам для всей программы.

Цикл 2j требует расположить J-ый столбец матрицы A вместе с J-ым столбцом матрицы B. Что не противоречит требованиям циклов 1i, 2i, 1j.

Требований к взаимному расположению элементов вектора D и элементов матрицы A от цикла 3i нет, поскольку вектор D не индексируется по переменной цикла I.

Цикл 3j требует расположить J-ый элемент вектора D вместе с J-ым столбцом матрицы A (обозначим D(J)-A(,J)). Это требование противоречит требованию цикла 1i , поскольку из них вытекает, что должны быть одинаково расположены строки и столбцы матрицы A, что невозможно осуществить. В данном примере три требования вступили в противоречие. Каждое требование имеет вес. Вес требования D(I)-B(I,) велик, а веса требований A(I,)-B(I,) и D(J)-A(,J) основаны на затратах на коммуникации. Поэтому будет отброшено скорее всего либо A(I,)-B(I,), либо D(J)-A(,J). Но возможны случаи, когда коммуникационные издержки в программе слишком велики, и тогда выгоднее пренебречь распараллеливанием циклов. В таком случае будет отброшено требование D(I)-B(I,).

После разрешения всех противоречий формируются перспективные варианты распределения данных (они отличаются друг от друга тем, что каждая группа связанных измерений либо распределяется по измерению процессорной решетки, либо не распределяется) и строятся соответствующие DVM-директивы, задающие распределение массивов между процессорами.

Распределение вычислений и организация коммуникаций

Дальнейшее построение схемы распараллеливания начинается с того, что для каждого варианта распределения данных определяются потенциально параллельные циклы. Такими циклами могут быть циклы без зависимостей по данным, циклы с редукционными зависимостями, зависимостями по приватным переменным и регулярными зависимостями по массиву (для них возможно конвейерное выполнение). Для параллельного цикла в DVM должно

соблюдаться правило собственных вычислений. Это правило фактически определяет принятие решения о распараллеливании цикла – все решает распределение модифицируемых в цикле данных (весь виток цикла будет выполняться на том процессоре, на котором находятся модифицируемые данные).

При формировании многомерных параллельных циклов из гнезда тесно вложенных циклов эксперт пытается задействовать все циклы гнезда, допускающие распараллеливание. Если среди них есть циклы с регулярными зависимостями по массиву, то для организации их конвейерного выполнения эксперт включает в многомерный цикл и несколько циклов по нераспределенным измерениям массивов, даже если эти циклы и не могут выполняться параллельно.

Кроме того, эксперт стремится уменьшить суммарный объем накладных расходов на организацию выполнения всех параллельных циклов программы. Дело в том, что некоторые параллельные циклы могут оказаться внутри последовательных циклов и не быть тесно вложенными в них. В таком случае каждый параллельный цикл будет организовываться заново на каждом витке охватывающего его цикла. Для уменьшения суммарного объема таких расходов надо отдавать предпочтение распараллеливанию внешних циклов.

Каждая схема распараллеливания связана с одним из вариантов распределения данных и поэтому имеет уже построенные директивы распределения данных. Эксперт дополняет их директивами распределения вычислений и директивами организации доступа к удаленным данным. На данном этапе строятся директивы, которые обеспечивают корректное распараллеливание вычислений в соответствии с заданным распределением данных.

Для параллельных циклов сначала выбирается одно из измерений массива, в соответствии с распределением которого будет проводиться распределение витков цикла. Это может быть модифицируемый массив, либо используемый только на чтение (если нет модифицируемых массивов). Причем, при выборе такого массива предпочтение отдается массиву, по которому есть регулярная зависимость.

Для всех редукционных зависимостей этого многомерного цикла строится спецификация REDUCTION и дописывается к текущей директиве объявления параллельного цикла.

Далее анализируются все операции чтения элементов массивов в этом цикле для обнаружения обращений к удаленным данным.

Если такие обращения к удаленным данным обнаружены, то организуется доступ к ним одним из трех возможных способов.

Если используемые элементы массива распределены с некоторым (константным) смещением относительно витков цикла и в цикле нет обращений к рассматриваемому массиву на запись, то доступ к удаленным данным осуществляется через теневые грани, обновление которых перед циклом задается директивой SHADOW_RENEW.

Если смещение константное, но в цикле есть обращения к рассматриваемому массиву на запись, то это задается директивой ACROSS. В этом случае доступ к удаленным данным осуществляется тоже через теневые грани, но их обновление осуществляется и перед циклом, и между квантами цикла в случае организации его конвейерного выполнения.

В противном случае доступ к удаленным данным осуществляется через специальные буфера, что задается директивой REMOTE_ACCESS.

Таким же образом, посредством задания директивы REMOTE_ACCESS обеспечивается доступ к удаленным данным и для последовательных циклов, не находящихся внутри параллельных циклов.

В пятой главе описывается алгоритм оценки эффективности разных схем распараллеливания.

В отличие от описанного в третьей главе алгоритма моделирования параллельного выполнения DVM-программы, в DVM-эксперте используется упрощенное моделирование параллельного выполнения программы. Это позволило сократить время оценки схем. К тому же моделирование производится по другой входной информации: не по трассе выполнения программы, а по полученной в результате анализа структуре программы и схеме распараллеливания. Поэтому необходимо знать размеры массивов, параметры циклов (начальное значение, конечное значение, шаг цикла) и свойства условных операторов (вероятность выполнения каждой ветки).

Для упрощения моделирования было решено воспользоваться предположением (эвристикой), что значения соответствующих характеристик выполнения каждой итерации последовательного цикла, охватывающего один или несколько параллельных циклов, начиная со второй итерации, будут близкими по величине, и их можно считать одинаковыми. В таком случае характеристики можно экстраполировать, а не моделировать выполнение каждой итерации.

Аналогичный прием применен и для упрощенного моделирования работы конвейера. Все процессоры представлены в виде логической решетки процессоров, которая используется для многомерных распределений данных. Решетка задает, на сколько процессоров распределяется каждое измерение данных. Для ускорения моделирования по каждому измерению процессорной решетки берется только по 4 процессора: 3 первых и последний. Загрузка других процессоров будет экстраполирована линейно (по второму и третьему процессору).

Помимо этого для каждого процессора конвейера детально моделируются выполнение только 4-х квантов конвейера (3 первых и последний). По характеристикам второго и третьего кванта экстраполируются характеристики последующих квантов от четвертого до предпоследнего. Характеристики первого и последнего квантов могут отличаться от характеристик средних квантов, и поэтому рассматриваются отдельно.

Помимо упрощенного моделирования параллельного выполнения программы, для ускорения оценки схем распараллеливания был предложен эвристический алгоритм поиска оптимальных решеток процессоров.

Поскольку построенные схемы распараллеливания позволяют использовать разные решетки процессоров заданной размерности, то для каждой схемы и заданного числа процессоров нужно находить оптимальную решетку процессоров, на которой время выполнения программы будет минимальным. Удалось найти некоторые эвристики, которые позволяют сократить область перебора и значительно ускорить поиск оптимальной решетки. В качестве одной такой эвристики предложено использовать тот факт, что при увеличении числа процессоров по какому-то измерению решетки некоторые процессоры не имеют элементов распределенных массивов (например, если количество процессоров больше количества элементов). Можно считать, что такое распределение данных и, как следствие, распределение вычислений, будет менее эффективным, чем более равномерное распределение, но использующее меньшее количество процессоров. В качестве второй эвристики предложено использовать то обстоятельство, что если при увеличении количества процессоров по какому-то одному измерению процессорной решетки время выполнения программы начинает увеличиваться, то нет смысла дальше увеличивать по этому измерению количество процессоров. Это связано с тем, что при росте числа процессоров, начиная с некоторого момента, потери из-за коммуникаций начинают возрастать быстрее, чем выигрыш от распараллеливания. Первая эвристика позволяет отбросить некоторые варианты решеток еще до их рассмотрения, а вторая используется во время поиска оптимальной решетки.

Шестая глава содержит результаты экспериментальной проверки алгоритма распараллеливания на тестах и реальных приложениях.

Компилятор ПАРФОР был испытан на ряде простых тестов, проверяющих различные механизмы распараллеливания программ. Также были получены результаты по распараллеливанию реальных приложений. Среди них тесты NAS BT, LU, SP, а также разработанные в ИПМ им.М.В.Келдыша РАН программы MHPDV и ZEBRA. Тексты этих всех программ были получены из их DVM-версий путем удаления всех DVM-директив и инлайн-подстановки процедур в тело головной подпрограммы.

Программа BT (Block Tridiagonal Solver) – нахождение конечно-разностного решения 3-х мерной системы уравнений Навье-Стокса для сжимаемой жидкости или газа. Используется трех диагональная схема, метод переменных направлений. При выполнении программы задавался класс C (сетка 162x162x162 и 250 итераций).

Программа LU (Lower-Upper Solver) отличается от BT тем, что применяется метод LU-разложения с использованием алгоритма SSOR (метод верхней релаксации), а программа SP (Scalar Pentadiagonal Solver) – тем, что используется скалярная пяти диагональная схема.

Программа MHPDV – трехмерное моделирование сферического взрыва во внешнем магнитном поле с помощью решения уравнений идеальной магнитной гидродинамики.

Программа ZEBRA – расчет нейтронных полей атомного реактора в диффузионном приближении.

Ниже приводятся времена выполнения исходных вариантов (распараллеленных вручную) программ, а также их вариантов, полученных в результате автоматического распараллеливания.

Таблица 1. Времена выполнения вариантов программ на МВС-100К (сек) (“авт” - автоматическое распараллеливание, “ручн” - ручное)

Варианты	Число процессоров 1	Число процессоров 8	Число процессоров 64	Число процессоров 256	Число процессоров 1024
BT-авт	мало памяти	1255.97	182.70	54.64	21.36
BT-ручн	мало памяти	817.88	128.01	30.27	7.19
LU-авт	3482.40	1009.49	148.78	40.33	25.55
LU-ручн	2103.14	858.26	122.61	34.99	19.97
SP-авт	1982.00	-	-	-	-
SP-ручн	2601.85	-	-	-	-
MHPDV-авт	3703.23	500.78	89.32	34.75	12.78
MHPDV-ручн	3574.29	486.74	79.63	32.15	10.98
ZEBRA-авт	75.09	11.13	1.96	-	-
ZEBRA-ручн	75.62	10.18	1.85	-	-

Таблица 2. Времена выполнения на Blue Gene/P (ВМК МГУ)

Варианты	Число процессоров 1	Число процессоров 8	Число процессоров 64	Число процессоров 256	Число процессоров 1024
BT-авт	мало памяти	2814.59	473.37	166.66	70.18
BT-ручн	мало памяти	1620.97	240.55	72.67	26.97
LU-авт	мало памяти	1897.53	352.28	125.53	61.16
LU-ручн	мало памяти	1670.60	259.09	92.40	47.80
SP-авт	мало памяти	-	-	-	-
SP-ручн	мало памяти	-	-	-	-
MHPDV-авт	12619.35	1604.29	228.56	59.97	18.14
MHPDV-ручн	12627.59	1606.61	233.33	61.05	18.28
ZEBRA-авт	569.73	74.45	10.83	-	-
ZEBRA-ручн	587.02	76.85	11.06	-	-

Программу SP распараллелить не удалось (никаких DVM-директив в нее не было вставлено). Для распараллеливания необходима ее серьезная модификация, чтобы избавиться от имеющихся там зависимостей между витками циклов, препятствующих их параллельному выполнению (разные витки одного цикла модифицируют один элемент массива, который не является редукционной или приватной переменной).

Увеличение времени выполнения автоматически распараллеленных вариантов программ BT и LU на малом количестве процессоров объясняется

тем, что на этих программах подстановка процедур сильно повлияла на оптимизацию кода компиляторами. Увеличение времени выполнения на большом количестве процессоров происходит из-за того, что DVM-эксперт не сумел организовать доступ к удаленным данным так же эффективно, как это смогли сделать разработчики DVM-системы вручную.

На программах MHPDV и ZEBRA времена выполнения автоматически распараллеленных вариантов близки к временам выполнения вручную распараллеленных вариантов.

Увеличение количества процессоров для программы ZEBRA (свыше 64-х) уже не ускоряет ее выполнение.

Таблица 3. Времена работы DVM-эксперта и характеристики программ

	BT	LU	SP	MHPDV	ZEBRA
Время работы DVM-эксперта на ПЭВМ (сек)	1855	96	879	41	753
Количество строк	10442	3635	5950	1878	2426
Количество циклов	504	424	499	116	49
Количество параллелизуемых циклов DVM-экспертом	481	410	293	115	28
Количество обращений к массивам	30530	6638	11015	4643	2680
Количество массивов	34	30	37	33	40
Суммарное число измерений	75	64	70	78	49
Распределение массивов по измерениям (начиная с одномерных массивов)	16 5 7 3 2 1	22 3 0 5 4	21 3 9 4	16 0 6 11	36 0 3 1
Количество групп измерений	6	5	5	5	6
Количество построенных схем	16	16	16	16	64
Среднее время построения одной схемы (сек)	38,56	3,93	2,62	1,43	0,29
Среднее время оценки эффективности одной схемы (сек)	77,39	2,06	52,31	1,13	11,47
Общее количество перебранных процессорных решеток	127	127	182	144	543

В **заключении** сформулированы основные результаты работы.

Основные результаты работы

1. Разработан алгоритм моделирования параллельного выполнения DVM-программы, позволяющий прогнозировать время выполнения и другие характеристики эффективности параллельной программы
2. Разработан алгоритм автоматического отображения последовательной программы на кластер, состоящий из алгоритмов
   - отбора наиболее перспективных вариантов распределения данных
   - распределения вычислений и организации коммуникаций
   - выбора наиболее эффективной схемы распараллеливания программы посредством моделирования ее параллельного выполнения
3. Разработанные алгоритмы программно реализованы в экспериментальной версии распараллеливающего компилятора с языка Фортран.

На базе разработанного алгоритма моделирования параллельного выполнения DVM-программы был также создан инструмент (DVM-предиктор), позволяющий на инструментальной ЭВМ проводить предварительную отладку эффективности выполнения DVM-программы на кластерах с заданными характеристиками узлов и коммуникационной системы. Этот инструмент используется на факультете ВМК МГУ при проведении практикума по технологиям параллельного программирования.

Публикации по теме диссертации

1. М.С. Клинов, В.А. Крюков. “Прогнозирование характеристик параллельного выполнения DVM-программ”. // Труды Всероссийской научной конференции “Научный сервис в сети Интернет: технологии параллельного программирования”, сентябрь 2006 г., г. Новороссийск. – М.: Изд-во МГУ, 2006, стр. 113-114
2. М.С. Клинов, В.А. Крюков. Система автоматизированного распараллеливания программ на языке Фортран. DVM-эксперт. // Труды Всероссийской научной конференции “Научный сервис в сети Интернет: многоядерный компьютерный мир”, сентябрь 2007 г., г. Новороссийск. – М.: Изд-во МГУ, 2007, стр. 95-97.
3. М.С. Клинов, В.А. Крюков. Автоматическое распараллеливание Фортран-программ. Отображение на кластер. Вестник Нижегородского университета им. Н.И. Лобачевского, 2009, № 2, с. 128–134.
4. М.С. Клинов. Прогнозирование характеристик эффективности выполнения DVM-программ на кластере. Вестник Нижегородского университета им. Н.И. Лобачевского, 2009, №. 4, с. 190-197.
5. М.С. Клинов. Алгоритмы автоматического отображения последовательных Фортран-программ на кластер. // Труды Всероссийской научной конференции “Научный сервис в сети Интернет: масштабируемость, параллельность, эффективность”, сентябрь 2009 г., г. Новороссийск. – М.: Изд-во МГУ, 2009, с. 298-299.