Федеральное государственное учреждение «Федеральный
исследовательский центр Институт прикладной математики
им. М.В. Келдыша Российской академии наук»
Московский государственный университет им. М.В. Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра системного программирования

На правах рукописи

Колганов Александр Сергеевич

Автоматизация распараллеливания Фортран-программ для гетерогенных кластеров

Специальность 05.13.11 —
«Математическое и программное обеспечение вычислительных машин,
комплексов и компьютерных сетей»

Диссертация на соискание учёной степени
кандидата физико-математических наук

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

Москва — 2020

Оглавление

Стр.

Введение 5
Актуальность темы и цель работы 5
Постановка задачи 11
Научная новизна 13
Практическая значимость 14
Апробация работы и публикации 15
Краткое содержание работы 15

Глава 1. Обзор автоматизированных систем распараллеливания программ на распределенные системы 17
1.1 Проблемы отображения на кластер 17
1.2 Система САПФОР 21

Глава 2. Схема работы и структура системы SAPFOR 2 . . 24
2.1 Схема работы системы 24
2.2 Структура системы 28
2.2.1 Модуль анализа и преобразования кода 28
2.2.2 Структура организации алгоритмов анализа и преобразования 28
2.2.3 Структура организации менеджера проходов 31
2.2.4 Модуль визуализации результатов анализа и преобразования кода 32
2.2.5 Спецкомментарии системы SAPFOR 2 33
2.2.6 Возможности языка Fortran-DVMH 34

Глава 3. Построение дерева циклов и графа функций 37
3.1 Построение дерева циклов 37
3.2 Построение графа вызовов функций 39

Глава 4. Построение областей распараллеливания 45

Стр.

4.1 Понятие области распараллеливания 45
4.2 Правила расстановки областей распараллеливания и разрешения конфликтов 47
4.3 Объединение областей распараллеливания 49

Глава 5. Построение схем распределения данных и вычислений 51
5.1 Способы отображения на кластер 51
5.2 Анализ обращений к массивам в программе 52
5.2.1 Построение связей циклов и массивов 52
5.2.2 Построение графа массивов 56
5.2.3 Определение распределяемых массивов 61
5.2.4 Поиск оптимального связывания массивов 63
5.2.5 Создание вариантов распределения данных 68
5.2.6 Создание директив выравнивания данных 69
5.3 Построение распределения вычислений и доступа к удаленным данным 70
5.3.1 Создание директив распределения вычислений 70
5.3.2 Организация доступа к удаленным данным 77
5.4 Расстановка вычислительных регионов и директив актуализации данных 79
5.5 Оценка и выбор схем распараллеливания 82

Глава 6. Исследование алгоритмов автоматизированного распараллеливания 86
6.1 Автоматическое распараллеливание небольших тестовых программ 86
6.2 Результаты автоматизированного распараллеливания модельных задач 89
6.2.1 Сравнение систем автоматизации распараллеливания САПФОР и SAPFOR 2 89
6.2.2 Распараллеливание тестов NAS версии 3.3 92

Стр.

6.2.3 Распараллеливание программы моделирования распространения упругих волн в средах со сложной 3D геометрией 99
6.3 Опыт применения механизма областей распараллеливания . 102
6.3.1 Инкрементальное распараллеливание теста NAS BT . 102
6.3.2 Инкрементальное распараллеливание программного комплекса COMPOSIT 115

Заключение 119

Список литературы 122

Приложение А. Описание внутреннего представления кода в системе SAPFOR 2 в библиотеке SAGE++ 129

Приложение Б. Описание спецкомментариев в системе SAPFOR 2 134

Введение

Актуальность темы и цель работы

Развитие высокопроизводительных систем не стоит на месте. На сегодняшний день существует большое разнообразие параллельных архитектур – многоядерные процессоры на x86 и Power архитектурах, ARM процессоры, графические процессоры, Intel Xeon Phi процессоры, ПЛИС и т.д. Несмотря на такое разнообразие вычислителей с параллельной архитектурой, одного такого вычислителя недостаточно для решения многих задач из класса HPC или обработки больших данных (BigData), поэтому существует большое количество вычислительных кластеров, объединяющих данные параллельные вычислители между собой.

Очень распространены кластеры, в узлах которых содержатся вычислители разной архитектуры, например, центральные процессоры и графические ускорители. Такие кластеры называются гибридными. Данная архитектура кластера вместе с распределенной памятью сильно усложняет разработку параллельных программ или отображение существующих последовательных программ в эффективные параллельные.

Разработка программ для высокопроизводительных кластеров различной архитектуры продолжает оставаться исключительно сложным делом, доступным достаточно узкому кругу специалистов. Основная причина этого – низкий уровень современной технологии автоматизации разработки параллельных программ.

В настоящее время практически все параллельные программы для гибридных кластеров разрабатываются с использованием низкоуровневых средств передачи сообщений (например, MPI или Shmem), а также с использованием некоторых средств параллельного программирования на системах с общей памятью для задействования нескольких ядер центрального процессора (например, OpenMP, pthreads, TBB) или графического процессора (CUDA, OpenACC). Такие гибридные программы трудно разрабатывать, сопровождать и повторно использовать при создании новых программ.

Для решения определенного класса задач имеются специализированные библиотеки, которые упрощают процесс написания программ. В других случаях приходится использовать языки параллельного программирования. Но трудности возникают не только при написании программ на языках параллельного программирования, но и при отладке таких программ. Очень часто разработка параллельной программы начинается с написания и отладки последовательной. Процесс распараллеливания отлаженной последовательной программы целесообразно максимально автоматизировать, а в идеале – осуществлять полностью автоматически, без участия программиста.

Как известно, полностью автоматическое распараллеливание на кластер отчасти неэффективно, а зачастую практически невозможно, потому что при переходе от последовательной программы к параллельной в большинстве случаев требуется изменение алгоритма, либо его серьезное преобразование. В некоторых случаях если пользователь вычислительного кластера встретит распараллеливающий компилятор, откомпилирует им свою старую последовательную программу, то потом обнаружит, что программа работает медленнее. Не всегда даже самый интеллектуальный компилятор сможет в последовательной программе распознать скрытые возможности параллельного выполнения. Но, с другой стороны, далеко не всякий программист способен оптимизировать или распараллелить программу лучше компилятора.

Проблема автоматического распараллеливания исследуется достаточно давно. Для систем на общей памяти существует множество инструментов, позволяющих в той или иной степени преобразовать последовательную программу в параллельную автоматически. Среди таких инструментов можно отметить Polaris, CAPO, WPP, SUIF, VAST/Parallel, OSCAR, ParallelWare, Intel Parallel Studio XE. К примеру, Intel предоставляет достаточно мощные средства для анализа программы с целью ее последующего распараллеливания на многоядерные процессоры с помощью OpenMP.

Для распараллеливания программы на общей памяти требуется распределить на ядра процессора только вычисления, которые, в основном, сосредоточены в циклах. В отличие от систем с общей памятью, на системах с распределенной памятью необходимо произвести не только

распределение вычислений, а также распределение данных, которое должно быть согласовано с распределением вычислений. **Согласованным распределением вычислений** будем называть такое, при котором распределенные вычисления выполняются на тех процессорах, на которые распределены требуемые им данные. Помимо этого, необходимо обеспечить на каждом процессоре доступ к удаленным данным – данным, которые расположены на других процессорах.

Для обеспечения эффективного доступа к удаленным данным требуется определить те данные, которые должны пересылаться с одного процессора на другой. Для этого требуется производить анализ индексных выражений не только внутри одного гнезда циклов (как в случае распараллеливания на общую память), но и между разными циклами для оптимизации коммуникационных обменов (группировки обменов для уменьшения количества посылок данных и совмещения обменов с вычислениями).

На текущий момент можно выделить следующие понятия в области распараллеливания программ:

- **Распараллеливание программ** – процесс их адаптации для эффективного выполнения на вычислительной системе с параллельной архитектурой, который заключается в переписывании на язык параллельного программирования, который может быть низкоуровневым (например, MPI, CUDA, OpenCL) или высокоуровневым, основанным на привычном языке последовательного программирования, расширенном директивами компилятору (например, OpenMP, OpenACC, DVM [1; 2]);
- **Автоматическое распараллеливание** – процесс отображения последовательной программы компиляторами (например, Intel или PGI) на параллельную архитектуру, состоящий в её преобразовании без участия пользователя для эффективного выполнения на параллельной вычислительной системе;
- **Автоматизация распараллеливания** – процесс отображения последовательной программы на параллельную архитектуру, состоящий в ее преобразовании, в котором пользователь принимает активное участие. Автоматизированное распараллеливание может включать в себя процесс автоматического распараллеливания.

Последний подход является наиболее перспективным в силу обилия различных архитектур и низкоуровневых языков параллельного программирования [3—5]. Для автоматизации распараллеливания программ создаются высокоуровневые языки параллельного программирования, такие как HPF, OpenMP-языки, DVM-языки, CoArray Fortran, UPC, Titanium, Chapel, X10, Fortress, XcalableMP [6], а также создаются системы автоматизации распараллеливания программ, такие как CAPTools/Parawise, FORGE Magic/DM, BERT77, ParalWare Trainer, Appolo, САПФОР [7], ДВОР, АРС [8], которые используют для отображения на параллельные вычислительные системы как языки низкого уровня, так и языки высокого уровня.

Но, несмотря на разнообразие различных систем автоматического или автоматизированного распараллеливания, используемые решения в каждой из них для отображения последовательной программы в параллельную одинаковы. Все системы используют анализ и/или преобразование абсолютно всей исходной последовательной программы. В случае отображения на общую память отказаться от распараллеливания какой-то части программы менее болезненно, чем в случае отображения на кластер. А в некоторых случаях невозможность распараллелить какую-то часть программы при отображении на кластер приводит к отказу от распараллеливания всей программы.

Ясно, что в случае распараллеливания больших программных комплексов статический и динамический анализы всего кода являются трудоемкой задачей [9; 10]. И зачастую для частичного распараллеливания полный анализ кода не требуется, но на данный момент не существует средств, которые позволили бы отобразить на гибридный кластер только часть большой последовательной программы.

К системам распараллеливания с такими недостатками относилась в том числе и разработанная ранее система САПФОР (SAPFOR) [7]. Подробный обзор системы САПФОР будет сделан несколько ниже, будут также рассмотрены ее слабые стороны. Система SAPFOR 2, являющаяся результатом данной работы и описываемая в диссертации, является развитием системы САПФОР по нескольким направлениям.

Таблица 1 — Сравнительные характеристики систем автоматизации, ориентированных на кластеры

	Parawise	Paradigm APC	SAPFOR	**SAPFOR 2**
Автоматически распараллеливающий компилятор	нет	**есть**	**есть**	**есть**
Отображение на кластер	**есть**	**есть**	**есть**	**есть**
Поддержка ГПУ	нет	нет	нет	**есть**
Автоматизация преобразований	нет	нет	нет	**есть**
Диалоговый режим	**есть**	нет	**есть**	**есть**
Инкрементальное распараллеливание	нет	нет	нет	**есть**
Стандарт Фортрана	77	77/**90**	77	**95**

В Таблице 1 представлены сравнительные характеристики 4-х систем, ориентированных на кластеры и построенных на базе автоматически распараллеливающих компиляторов (Paradigm, APC, SAPFOR, SAPFOR 2), и наиболее известной системы автоматизированного распараллеливания для кластеров Parawise.

В состав системы SAPFOR 2 входит один из важных компонентов – автоматически распараллеливающий компилятор для систем с распределенной памятью, который переводит последовательную пользовательскую программу в параллельную программу для гибридных кластеров в модели DVMH [11]. Взаимодействие с пользователем осуществляется и на этапе подготовки последовательной программы к распараллеливанию, и в процессе этого распараллеливания. Можно отметить, что функции компилятора расширены средствами автоматического выполнения преобразований, необходимых для эффективной работы получаемых параллельных программ.

Как было отмечено выше, описываемая система автоматизированного распараллеливания ориентирована на преобразование уже существующих последовательных программ в эффективные параллельные в модели

DVMH. Такие программы могут содержать в себе отдельные модули, решающие определенные задачи, например, модуль ввода/вывода, модуль решения систем линейных уравнений, модуль, отвечающий за подготовительные работы.

В большинстве случаев такие программы не нуждаются в полном распараллеливании, а в некоторых случаях полное распараллеливание попросту невозможно без существенного переписывания кода. Обычно бывает достаточно распараллелить только модули, производящие сложные вычисления над уже подготовленными данными. С другой стороны, программа может быть настолько большой, что программисту необходимо понять, можно ли вообще получить эффективно работающую программу на конкретной архитектуре. Для выполнения такой оценки достаточно попытаться распараллелить только некоторые модули такой программы, которые могут занимать до 90% времени.

Для решения выше описанных проблем в данной работе рассматривается метод автоматизированного инкрементального распараллеливания программ на кластер. Инкрементальное распараллеливание широко применяется при разработке программ для мультипроцессора, но при использовании для систем с распределенной памятью его применение наталкивается на значительные трудности. Распределение данных по процессорам влечет за собой накладные расходы на коммуникации, для эффективной оптимизации которых, как правило, требуется рассмотрение всей программы в целом, а не отдельных ее частей.

Для обеспечения возможности инкрементального распараллеливания на кластеры в системе SAPFOR 2 предлагается ввести понятие области распараллеливания. Это позволит последовательно переходить от рассмотрения отдельных небольших областей к более крупным областям, вплоть до целой программы, сохраняя при этом преемственность ранее принятых решений по распараллеливанию отдельных областей и уточняя их при необходимости. Области распараллеливания отражают те участки кода исходной программы, которые будут рассматриваться системой SAPFOR 2 (она будет строить схемы распараллеливания для каждой области независимо от наличия других областей). Можно отметить следующие достоинства такого подхода:

– возможность распараллелить не всю программу, а ее времяемкие фрагменты. Это упрощает работу системы SAPFOR 2 и/или программиста, так как существенно сокращается объем кода программы для анализа и распараллеливания, и позволяет потратить больше времени ЭВМ (и программиста) на то, чтобы найти лучшие схемы распараллеливания времяемких фрагментов;
– найденные решения для времяемких фрагментов могут быть использованы в качестве подсказки при исследовании оставшихся частей программы на следующих итерациях распараллеливания в системе SAPFOR 2;
– появляется возможность ручного распараллеливания некоторых фрагментов программы и учета принятых программистом решений при распараллеливании других фрагментов системой SAPFOR 2.

Постановка задачи

Структура системы САПФОР [7; 12] не позволила в полной мере реализовать метод инкрементального распараллеливания, а также расширить класс задач, которые могут быть автоматизированно отображены на кластер. Настоящая диссертационная работа посвящена дальнейшему развитию системы автоматизации распараллеливания Фортран-программ САПФОР в следующих направлениях:

– автоматизация выполнения преобразования исходных последовательных программ, что позволит облегчить и ускорить разработку эффективных программ для гибридных кластеров;
– обеспечение поэтапного (инкрементального) распараллеливания больших программ и программных комплексов, что позволит серьезно расширить класс программ, для которых можно успешно применять систему;
– повышение эффективности выполнения параллельных программ за счет развития алгоритмов распределения данных, распределения вычислений, организации доступа к удаленным данным, а

также выделения фрагментов кода, которые могут быть выполнены на графических процессорах.

Для достижения поставленной цели в работе решаются следующие основные **задачи**:

- анализ существующих решений в области автоматизированного и автоматического распараллеливания программ на кластер, выявление их достоинств и недостатков;
- исследование, разработка и реализация алгоритмов автоматически распараллеливающего компилятора – алгоритмов автоматического построения схем распараллеливания (схем отображения последовательной программы на гетерогенный кластер) и выполнения необходимых преобразований;
- разработка, проектирование и реализация метода поэтапного (инкрементального) распараллеливания последовательных Фортран-программ для гетерогенных кластеров;
- разработка новой системы автоматизации распараллеливания и исследование полученной системы на тестовых и реальных прикладных программах.

Ставятся следующие ограничения на применимость системы SAPFOR 2:

- вычислительные части программы, которые будут распараллелены системой, должны быть написаны на подмножестве языка Фортран 95 (нет поддержки указателей, производных типов данных, интерфейсных функций, функций с переменным числом параметров);
- вычислительная программа должна иметь главную программную единицу;
- распределение вычислений осуществляется путем распараллеливания витков циклов согласно модели DVMH. Распараллелены могут быть следующие циклы:
  - представленные в канонической форме (Canonical Loop Form из стандарта OpenMP [13]);
  - с прямоугольным итерационным пространством;
  - без операторов ввода/вывода;
  - с одним входом и одним выходом;

  - в которых допускаются зависимости вида редукция по следующим типам: максимум, минимум, сумма, произведение, логические OR, AND, EQV, NEQV, а также MINLOC, MAXLOC;
  - в которых допускаются зависимости по приватным переменным и регулярная зависимость (зависимость с постоянным шагом) по распределенному массиву.
- алгоритм распределения данных должен минимизировать коммуникационные обмены между узлами кластера.

Также допускается:

- вычислительный комплекс может содержать разного вида процедуры и функции;
- перераспределение данных до входа в область распараллеливания и после выхода из нее;
- перераспределение данных перед параллельными циклами, если распределение вычислений для данного цикла противоречит выбранному распределению данных для всей программы.

Научная новизна

В результате данной работы была разработана, спроектирована и реализована новая система автоматизации отображения Фортран-программ на гетерогенный кластер SAPFOR 2 с поддержкой инкрементального распараллеливания. Были разработаны алгоритмы построения распределения данных и построения распределения вычислений. Данные алгоритмы основываются на теории графов, распределение данных строится эффективным образом с учетом статической и динамической (в случае ее наличия) информации распараллеливаемого программного комплекса. Предложенная структура системы SAPFOR 2 позволяет расширять ее функциональность: для добавления новых возможностей по анализу или преобразованию кода можно использовать уже существующие требуемые для этого блоки анализа или преобразований, чтобы сосредоточиться на реализации новой функциональности.

Спроектирован, разработан и реализован метод инкрементального распараллеливания на кластер, использующий механизм областей распараллеливания, что существенно расширило класс задач, к которым можно применить систему SAPFOR 2. Механизм областей, позволяющий отделить вычислительную часть программы, которую необходимо выполнять параллельно, вместе с возможностью DVM-системы автоматизированно управлять перемещением данных между вычислительными узлами позволил реализовать для гетерогенных кластеров так называемое инкрементальное распараллеливание больших программных комплексов, что до сегодняшнего времени не удавалось ни одной системе автоматического или автоматизированного распараллеливания.

Проведенные исследования применимости системы SAPFOR 2 на тестах и реальных приложениях показали высокую эффективность предложенного подхода, позволяющего значительно упростить и существенно ускорить разработку эффективных параллельных программ для гетерогенных кластеров, а также показали, что для определенного класса задач можно писать последовательные программы на языке Фортран, которые будут автоматизировано (либо автоматически) распараллелены и эффективно выполнены на таких кластерах.

Практическая значимость

В рамках данной диссертационной работы был предложен новый, инкрементальный подход к автоматизации распараллеливания программ на гетерогенные кластеры. На базе разработанной и спроектированной архитектуры автоматизированного распараллеливания программ была создана новая система SAPFOR 2, включающая в себя множество алгоритмов анализа и преобразования исходного кода программы. Реализованная система позволяет выполнять инкрементальное автоматизированное распараллеливание класса программ, использующих структурные сетки и написанных на языке Фортран 95 с некоторыми ограничениями входного языка. Структура разработанной системы SAPFOR 2 позволяет легче расширять ее

функциональность новыми возможностями благодаря использованию модульной организации алгоритмов анализа и преобразований.

Апробация работы и публикации

Основные результаты научно-квалификационной работы были доложены и опубликованы в статьях на российских и международных научных конференциях и семинарах, таких как «Научно-практическая конференция Технологии параллельной обработки графов», «Параллельные вычислительные технологии», «Суперкомпьютерные дни в России», «Ломоносовские чтения», «Национальный Суперкомпьютерный Форум», «Научный сервис в сети Интернет». Имеется 28 публикаций, из которых девять – в журналах из списка ВАК и четыре – в изданиях из списков Web of Science и Scopus [3—5; 9; 10; 14—35].

Реализованная система SAPFOR 2 была опробована для распараллеливания широко известного пакета тестов NAS Parallel Benchmarks NPB 3.3. Сравнивались времена выполнения полученных параллельных программ и времена выполнения MPI-версий этих же программ, написанных разработчиками данных тестов. Также была проведена апробация на некоторых прикладных программах, разработанных в ИПМ им. Келдыша РАН, прикладной программы «Моделирования распространения упругих волн в средах со сложной 3D геометрией поверхности», разработанной в Институте вычислительной математики и математической геофизики СО РАН и на программном комплексе COMPOSIT, решающим задачу моделирования добычи залежей нефти и газа, разработанном в Федеральном научном центре НИИСИ РАН.

Краткое содержание работы

Основной текст данной работы состоит из введения, шести глав и заключения. В Главе 1 дается обзор существующих работ по данной

тематике. В Главе 2 приводится описание схемы работы реализованной системы SAPFOR 2, также дается описание структуры этой системы. В Главе 3 приводится описание алгоритмов построения основных структур данных для анализа и преобразований – дерева циклов и графа вызовов функций. В Главе 4 приводится описание понятия областей распараллеливания и алгоритмы их формирования в системе SAPFOR 2. В Главе 5 описывается алгоритм распределения данных, которое является ключевой проблемой при распараллеливании на кластер. Также описываются алгоритмы построения распределения вычислений и доступа к удаленным данным, отображения на графический процессор и методы оценки полученных параллельных версий программ. В Главе 6 рассматриваются результаты тестирования реализованной системы автоматизации распараллеливания SAPFOR 2 на различных программах, а также приводится сравнение времени работы и результатов распараллеливания с предыдущей системой автоматизации.

Глава 1. Обзор автоматизированных систем распараллеливания программ на распределенные системы

1.1 Проблемы отображения на кластер

Высокопроизводительные кластеры появились достаточно давно. Но тенденции построения суперкомпьютеров в настоящее время почти не изменились. Для того, чтобы достичь большой вычислительной мощности, используется большое количество узлов, объединенных между собой коммуникационной сетью. Каждый из узлов может использовать как процессоры общего назначения, так и процессоры иной архитектуры, например, графические процессоры, Intel Xeon Phi и др.

Использование различных архитектурных решений в узле кластера обусловлено, прежде всего, более высокой энергоэффективностью. Наращивание количества узлов в какой-то момент станет невозможным и экономически нецелесообразным, поэтому существует некоторый баланс между количеством узлов кластера и тем, из чего состоит вычислительный узел кластера.

Для использования таких больших вычислительных мощностей (порядка нескольких десятков петафлопс [36]) требуется не только распараллелить вычисления в узле кластера, но и также эффективно задействовать множество узлов. Тем самым разработка программ для кластеров оказывается значительно сложнее, чем, например, разработка программ для систем с общей памяти с использованием OpenMP или OpenACC расширений. Программист должен распределить данные между процессорами, а программу представить в виде совокупности процессов, взаимодействующих между собой посредством обмена сообщениями.

Ввиду такой сложности необходимо создание инструмента, который будет автоматически преобразовывать последовательную программу в параллельную программу для кластера. Известно, что для векторных процессоров подобные средства широко и успешно использовались и используются до сих пор. Однако автоматическое распараллеливание для кластеров гораздо сложнее по следующим причинам:

- взаимодействие процессоров через коммуникационную систему требует значительного времени, поэтому вычислительная работа должна распределяться между процессорами крупными порциями, чтобы иметь возможность минимизировать коммуникационные затраты;
- в отличие от систем с общей памяти, на системах с распределенной памятью для эффективного распараллеливания необходимо произвести не только распределение вычислений, но и распределение данных, а также обеспечить на каждом процессоре доступ к удаленным данным, то есть к таким данным, которые расположены на других процессорах;
- распределение вычислений и данных должно быть произведено согласованно. Несогласованность распределения вычислений и данных приведет к значительному увеличению времени коммуникаций для доступа к удаленным данным или для их перераспределения между процессорами. Согласование распределений вычислений и данных требует тщательного анализа всей программы, и любая неточность анализа может привести к катастрофическому замедлению выполнения программы или к невозможности ее распараллеливания.

Распараллеливание существующей последовательной программы распадается на две подзадачи – анализ последовательной программы и преобразование данной программы в параллельную. Автоматизировать можно каждую из этих подзадач. Попытки полностью автоматизировать обе подзадачи для параллельных вычислительных систем с распределенной памятью, проведенные в 90-х годах [37], привели к пониманию того, что для таких вычислительных систем полностью автоматическое распараллеливание реальных производственных программ возможно только в очень редких случаях. Поэтому исследования в области автоматического распараллеливания для параллельных вычислительных систем с распределенной памятью практически были прекращены, а результаты тех исследований, которые были продолжены [38; 39], оказались неудачными – при выполнении на кластере больших производственных программ не удалось получить большие ускорения.

Как уже было отмечено, автоматизация распараллеливания ведется в двух направлениях – создание высокоуровневых языков параллельного программирования, и также создание систем автоматизированного распараллеливания, например, BERT77 [40], Parawise [41], FORGE [42].

Высокие требования к эффективности выполнения параллельных программ и изменения в архитектуре параллельных вычислительных систем 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, но и по ее отображению на параллельную вычислительную систему. Это является серьезным недостатком, вызывающим огромные трудности при использования таких систем.

Таким образом, из всех известных крупных систем автоматизированного распараллеливания на кластер нет ни одной, которая стабильно развивается и поддерживается. Вышеперечисленные системы являются больше академическими проектами, нежели промышленными системами. Основными недостатками таких систем являются:

- попытка распараллелить всю программу целиком, составить для этой программы согласованное решение по распределению данных. А ведь, зачастую, распараллеливание всей программы не требуется;
- сильное вовлечение программиста в принятие ответственных решений по распределению данных. По сути, такие системы сложно назвать даже автоматизированными системами для распараллеливания на кластер, потому что они предоставляют некоторый анализ программ, а также помогают расставлять некоторые спецификации параллелизма в коде программы, но только по согласованию с программистом.

Основной проблемой, которая вызывает трудность автоматического отображения на кластер, является огромное количество возможных вариантов распределения данных, а также оценка их эффективности для их ранжирования.

Для многомерных массивов обычно используется многомерное распределение, которое отображается на многомерную решетку процессоров. Использование многомерных распределений данных вызвано тем, что оно позволяет использовать больше процессоров, чем при одномерном распределении, при котором их количество ограничено размером одного измерения массива. А также для многих программ при многомерном распределении повышается отношение количества вычисляемых на процессоре элементов (объема многомерного параллелепипеда) к количеству пересылаемых с других процессоров граничных элементов (площади поверхности многомерного параллелепипеда).

Если рассматривать все массивы отдельно друг от друга, то количество вариантов распределения данных может быть очень велико. Обозначим через S_{dim} сумму количества всех измерений по всем распределяемым массивам (массивам, которые будут распределены на узлы кластера). Тогда, если строить все варианты только по правилу распределять ли очередное измерение или не распределять то количество вариантов будет $2^{S_{dim}}$.

При использовании многомерных распределений получается, что для каждого распределенного измерения массива нужно указать, на какое измерение процессорной решетки его отобразить. Поэтому, если размерность процессорной решетки равна N, то каждое измерение можно либо не распределять, либо распределить не более чем N способами, что в итоге даст $N + 1$ способ для каждого измерения массива. Подставляя в формулу, получим количество вариантов, которых будет порядка $(N + 1)^{S_{dim}}$.

К примеру, для программ LU, BT, SP из пакета тестов NAS [43] количество массивов для распределения порядка 30, а суммарное число измерений – 70. Даже построить все 2^{70} вариантов (распределять или не распределять измерение) не представляется возможным, не говоря уже о том, что каждый из них необходимо каким-либо образом оценить для выявления лучшего.

1.2 Система САПФОР

Для решения вышеописанных проблем в Институте прикладной математики им. М.В. Келдыша РАН была разработана система САПФОР [12] (Система Автоматизированного Распараллеливания ФОРтран программ). Особенностью данной системы является использование автоматически распараллеливающего компилятора, преобразующего потенциально параллельную программу (программу, которая может быть автоматически переведена в параллельную без участия пользователя) в ее параллельную версию для заданной ЭВМ. В состав данной системы САПФОР входят:

- база данных, которая содержит в себе краткую информацию о программе и результаты работы экспертов;
- анализаторы входной программы: построение усеченного внутреннего представления и поиск зависимостей;
- описание целевой ЭВМ, которое также хранится в базе данных;
- генератор параллельных версий программы;
- визуализатор процесса распараллеливания.

Процесс распараллеливания программы с помощью данной системы представляет собой последовательный запуск сначала всевозможных анализаторов для того, чтобы получить представление пользовательской программы в базе данных (БД), а затем – экспертов для получения параллельных версий. Анализ и распараллеливание выполняются над всей программой целиком, поэтому, любая проблема, препятствующая анализу или распараллеливанию, приводит к остановке всего процесса.

Сценарий использования системы САПФОР таков. Сначала исходный текст Фортран-программы подается на вход анализатору. Он записывает результаты анализа в БД. Содержимое БД можно посмотреть при помощи специальной утилиты, которая представляет его в текстовом виде. Полученная БД подается на вход эксперту. Эксперт находит схемы распараллеливания исходной программы, оценивает их для заданного кластера и строит текст программы в модели Fortran-DVM. В процессе работы эксперта производится выдача на экран и в файл информации о результатах распараллеливания программы и оценки схем распараллеливания.

Дальнейшая компиляция, выполнение и исследование эффективности полученной программы проводится на кластере, где установлена DVM-система. Если эффективность полученной программы не устраивает программиста, то он может вставить в текст исходной программы указания (директивы системе САПФОР) по корректировке результатов анализа, а может существенно изменить свою программу, и затем подать измененную программу на вход анализатору и повторить весь процесс заново.

В реализации эксперта рассмотренной системы САПФОР существуют следующие ограничения:

- программы должны быть написаны на языке Фортран 77;
- программы не должны содержать функций и процедур или должны допускать их инлайн-подстановку;
- отсутствие преобразований кода, только вставка директив языка Fortran-DVM (параллельная программа должна минимально отличаться от исходной последовательной);
- распараллеливание вычислений посредством распределения витков циклов по процессорам ЭВМ;
- требования к параллельным циклам, вытекающие из выбора языка Fotran-DVM в качестве языка параллельного программирования:
  - цикл является тесно-гнездовым циклом с прямоугольным индексным пространством;
  - измерения массивов индексируются только регулярными выражениями типа a*I + b , где I - индекс цикла;
  - между витками цикла могут быть только зависимости по редукционным и приватизируемым переменным, а также регулярные зависимости по распределенному массиву;
  - нет операторов ввода-вывода;
  - есть только один вход и один выход из цикла;
- распределение измерений массивов равными блоками;
- статическое распределение данных (считалось, что перераспределения требуются очень редко);
- отсутствие оптимизации коммуникаций (группировка, асинхронность, удаление ненужных обменов, замена коммуникаций дублирующими вычислениями).

Некоторые из приведенных ограничений очень сильно сужают класс задач, которые можно распараллелить такой системой. Подчеркнем те ограничения, которые существенно влияют на процесс распараллеливания.

1. Подстановка процедур. Система САПФОР требовала полной подстановки процедур. Если рассматривать большие программные комплексы, то произвести полную подстановку процедур не представляется возможным. А значит, распараллеливание серьезных программ попросту невозможно.

2. Использование языка Фортран 77. Данное ограничение также является неприемлемым для современных программ. К примеру, в данной версии языка нельзя использовать динамические массивы или модули.

3. Отсутствие преобразований кода. С одной стороны, преобразования кода могут сильно изменить программу, что приведет к нарушению концепции минимального отличия последовательной и параллельной программ, с другой – однотипные преобразования по всей программе могут быть выполнены автоматически и без участия программиста, что существенно сократило бы время на преобразования и количество внесенных программистом ошибок.

4. Распараллеливание полностью всей программы. Данный подход хорошо работает для небольших, обособленных программ, в которых решается какая-то одна конкретная задача. Для больших программных комплексов данный подход попросту неприменим, так как в них могут содержаться абсолютно разные модули, такие как ввод-вывод, реализации различных решателей, функции записи и чтения контрольных точек и т.д. И, зачастую, вычислительная часть, которую необходимо распараллелить, занимает порядка 20-30% всего кода.

5. Массивы могут быть связаны между собой только «один к одному», то есть все элементы массива $A[I]$ могут быть отображены на элементы массива $B[I]$. Данное ограничение существенно сужает класс задач, которые можно эффективно распараллелить на кластер. Причем такое отображение не всегда возможно, например, если размер массива A больше, чем размер массива B. Тогда требуется использовать средства DVM-шаблонов [11].

Глава 2. Схема работы и структура системы SAPFOR 2

2.1 Схема работы системы

Новая система SAPFOR 2 была разработана с целью устранить выше описанные недостатки предыдущей системы автоматизации распараллеливания САПФОР и расширить класс программ, использующих структурные сетки, к которым можно было бы ее применять. Система SAPFOR 2 состоит из модулей анализа и преобразования. Вместо базы данных и усеченного представления программы, все модули анализа и преобразования имеют доступ к полному дереву разбора исходной программы.

Доступ к полному дереву разбора позволяет реализовать межпроцедурный анализ (с учетом контекста вызовов функций) на основе графа потока управления. С помощью межпроцедурного анализа выполняется анализ приватизируемых переменных для циклов, выполняется связывание передаваемых параметров процедуры в точках их вызова, а также выполняется проверка свойств программы: анализ обращений к массивам, переданных в качестве параметров в процедуры, наличие или отсутствие рекурсии, проверка на необходимость подстановки процедуры, обобщение свойств циклов и процедур.

Применение межпроцедурного анализа отменяет требование полной подстановки процедур во всей программе. Подставлять процедуры требуется лишь в некоторых случаях, например, когда внутри одного потенциально параллельного цикла (такого цикла, который может быть отображен на параллельные архитектуры или его витки могут быть выполнены параллельно и независимо) есть вызов процедуры, в которой находится другой потенциально параллельный цикл. В таком случае требуется подстановка в точку вызова только в этом цикле для объединения двух циклов в тесно-вложенные. Также использование внутреннего представления программы в виде абстрактного синтаксического дерева (АСТ) позволяет выполнять более точный статический анализ кода программы по сравнению, например, с усеченным представлением программы в виде базы данных.

Все алгоритмы, которые выполняют какие-либо действия над программой, были разделены на алгоритмы анализа и алгоритмы преобразования кода. К алгоритмам анализа относятся те алгоритмы, в результате работы которых исходный код программы не меняется, а к алгоритмам преобразования относятся те алгоритмы, в результате работы которых исходный код программы меняется. Так, например, вставка DVM-директив приводит к изменению кода программы, а значит, относится к алгоритмам преобразования.

Рассмотрим сценарий использования системы SAPFOR 2. Исходный текст программы, представленный набором файлов на языке Фортран, подается на вход системе визуализации. В качестве входного языка используется Фортран 95. Для начала работы с программой строится системой SAPFOR 2 внутреннее представление (выполняется синтаксический анализ). Затем пользователь может выполнять доступные алгоритмы анализа и преобразования. Для получения параллельной версии программы необходимо выполнить два основных алгоритма: построение распределения данных (алгоритм из группы анализа) и построение параллельной версии (алгоритм из группы преобразований). Остальные алгоритмы анализа, которые требуются для выполнения основных, вызываются автоматически.

В процессе работы алгоритмов анализа и преобразований система визуализации сообщает пользователю обо всех проблемах, которые мешают построить распределение данных с привязкой к строкам исходной программы. К сожалению, на данный момент система не предлагает варианты решения возникающих проблем – программист должен принимать решения сам на основе выданных диагностик.

В случае неспособности проанализировать весь исходный код целиком пользователь имеет возможность использовать механизм областей распараллеливания – совокупность строк исходного кода программы, которые будут анализироваться и распараллеливаться системой. Данный механизм позволяет сузить действие системы SAPFOR 2 и направить все усилия на распараллеливание наиболее важного участка кода. Области распараллеливания могут быть расставлены пользователем вручную, либо может быть использован автоматический режим, который по полученному профилированию программы в результате ее запуска выделит наиболее значимые участки кода. Для использования автоматического режима, необходимо

выполнить запуск программы в режиме профилирования для сбора статистики с помощью инструмента GCov предоставляемого компилятором GCC.

Для согласованного построения распределения данных и сокращения возможных вариантов отображения данных на узлы кластера используется граф массивов. Граф массивов отражает связи между измерениями массивов согласно их использованию в циклах программы. Так как распределение вычислений происходит посредством распределения витков циклов в модели DVMH, то необходимо, чтобы процессор модифицировал только те данные, которые находятся на нем. Таким образом, требование связи одного измерения массива с другим диктуется использованием данных массивов в циклах программы. Также граф массивов отражает однозначное отображение всех массивов в дерево выравнивания в модели DVMH, которое обозначается в программе с помощью директив DISTRIBUTE и ALIGN.

Вершинами графа являются измерения массивов, а дуги графа показывают связь между этими измерениями. Каждой дуге приписан некоторый вес, показывающий, насколько часто осуществляется обращение к данным измерениям массивов и насколько существенна данная связь. Также дуге приписаны коэффициенты $(A,\ B)$ в линейном обращении $A*x+B$ к измерению массива, где $A,\ B$ – константы $(A \neq 0)$, а x – переменная цикла. В данный момент нелинейные обращения не поддерживаются системой SAPFOR 2 и не добавляются в граф. В зависимости от того, является ли данное обращение записью или чтением, косвенной адресацией, или зависит от нескольких итерационных переменных циклов, система SAPFOR 2 классифицирует такие обращения как конфликтные. Конфликты могут быть устранимыми (например, с помощью доступа к удаленным данным), и неустранимыми, препятствующими распараллеливанию цикла и требующими преобразования кода.

Для построения распределения данных необходимо удалить все циклы в графе массивов. Циклы в графе массивов могут отражать конфликты в распределении данных. Конфликтом будем называть ситуацию, когда нельзя однозначно отобразить элементы одного массива на элементы другого массива по какому-либо правилу.

Для устранения циклов в графе массивов, в том числе образующих конфликты, было разработано два алгоритма. Первый алгоритм является переборным и состоит из двух этапов – поиска всех простых циклов в графе и выбора набора дуг с минимальным совокупным весом для устранения найденных циклов. Данный алгоритм является точным, но сложность алгоритма экспоненциальна, поэтому его использование на большом количестве массивов сильно ограничено. Второй алгоритм является линейным – поиск максимального остовного дерева. Данный алгоритм является приближенным, так как не осуществляется перебор всех вариантов, но позволяет получить приемлемое решение при использовании большого количества массивов (когда достаточно много циклов и дуг в графе).

Так как вес дуги отражает ее значимость в тексте программы и коммуникационные затраты в случае «отказа» от данной дуги, то, выбрав набор дуг с минимальным весом, можно достичь лучшего распределения данных – попытаться удовлетворить требование всех циклов программы и минимизировать ширину теневых граней и, следовательно, объем межпроцессорных коммуникаций.

На основе графа массивов, не содержащего циклы и кратные дуги, строится распределение данных. Для каждой группы связных вершин графа создается DVM-шаблон (некоторый виртуальный массив, не занимающий место в памяти), измерения которого будут распределенными или нераспределенными. Все массивы в графе массивов выравниваются на данный шаблон по соответствующим правилам, задаваемым дугами графа. Для каждой области распараллеливания строится свой граф массивов и свое независимое распределение данных.

На основе построенного распределения данных строится распределение вычислений и организуется доступ к удаленным данным – создается параллельная версия программы. Если полученная параллельная версия программы не устраивает пользователя, он может внести изменения в полученное распределение данных в исходную версию программы или добавить спецкомментарии системе SAPFOR 2, а затем построить новое распределение данных и новую параллельную версию (при этом предыдущая параллельная версия не удаляется).

2.2 Структура системы

Система SAPFOR 2 представляет собой совокупность программных модулей, которые направлены на автоматизацию распараллеливания больших программных комплексов, написанных на языке Фортран, на гибридные кластеры, содержащие в узлах как многоядерные процессоры, так и ускорители различной архитектуры. Система состоит из модуля анализа/преобразования кода и модуля визуализации результатов анализа/преобразования кода. В данной работе рассматривается реализация модуля анализа и преобразования кода, являющегося ядром системы SAPFOR 2.

2.2.1 Модуль анализа и преобразования кода

Данный модуль позволяет выполнять анализ исходного кода на уровне внутреннего представления программы в виде абстрактного синтаксического дерева, а также выполнять source-to-source преобразования. Идея организации выполнения анализа и преобразований программы была заимствована из LLVM.

2.2.2 Структура организации алгоритмов анализа и преобразования

Рассмотрим структуру организации алгоритмов анализа и преобразования системы SAPFOR 2. Все алгоритмы анализа и преобразования организованы в виде, так называемых, проходов. Проходом будем называть совокупность некоторых простых функций, выполняющих некоторый анализ или преобразование исходного кода пользовательской программы. Каждый проход в общем случае состоит из двух фаз: независимая обработка всех файлов проекта (программы) и объединение результатов обработки

по файлам (так называемая агрегирующая фаза). Наличие обеих фаз является не обязательным требованием к созданию проходов.

Проходы могут иметь зависимости и связи между собой. Например, для построения графа функций необходимо выполнить проход, который строит граф циклов. Для указания зависимостей между проходами предназначен менеджер проходов. Менеджер проходов представляет собой структуру, которая хранит прямые зависимости между теми или иными проходами. Косвенные зависимости будут выведены автоматически при выполнении того или иного прохода. Зависимости между проходами возникают из-за того, что полученные данные анализа одного прохода необходимы для выполнения другого прохода.

Алгоритм анализа или преобразования может представлять собой совокупность проходов. Это может быть удобным для использования уже реализованных проходов при добавлении нового, а также для упрощения реализации того или иного прохода. К проходам анализа относятся те алгоритмы, которые выполняют некоторый анализ программы, не изменяя ее исходный код. И, напротив, к проходам преобразования относятся те алгоритмы, которые выполняют изменение исходного кода программы. Например, вставка директив распараллеливания является проходом преобразования, так как происходит изменение текста программы. На данный момент система SAPFOR 2 включает в себя 85 проходов. Перечислим ключевые из них:

– **Проходы анализа:**
    1. Построение графа вызовов функций. Реализация межпроцедурного анализа на основе графа функций для определения неиспользуемых функций и рекурсий в программе;
    2. Построение графа циклов;
    3. Набор проходов, которые проверяют для системы SAPFOR 2 входные ограничения анализируемой программы, например, наличие или отсутствие циклов с метками, наличие include-файлов с исполняемыми операторами и т.д.;
    4. Обработка спецкомментариев для системы SAPFOR 2;
    5. Совокупность проходов, определяющих зависимости в программе;

6. Построение директив распределения данных и вычислений в модели DVMH;
7. Построение директив доступа к удаленным данным в модели DVMH;
8. Обработка областей распараллеливания, считывание существующих DVMH-директив в программе;
9. Расстановка регионов DVMH-модели для выполнения участков кода на ГПУ или многоядерном процессоре, расстановка директив актуализации для корректного переноса данных между ГПУ и ЦПУ.

– **Проходы преобразования:**

1. Преобразование циклов с метками в DO–ENDDO формат;
2. Вставка директив распределения данных и вычислений в модели DVMH;
3. Вставка директив доступа к удаленным данным в модели DVMH;
4. Подстановка процедур;
5. Подстановка выражений;
6. Разделение/слияние циклов;
7. Расширение/сужение приватных переменных;
8. Устранение конфликтов областей распараллеливания.

Построение абстрактного синтаксического дерева (AST, ACT) выполняется отдельным компонентом анализа – фронтендом или парсером исходного кода. Библиотека Sage++ [44] представляет собой реализацию данного дерева разбора. Представление создается с помощью парсера, который использует грамматику для синтаксического анализа. Данная грамматика расширена для поддержки директив (спецкомментариев) DVMH и SAPFOR 2. Полное описание возможностей библиотеки Sage++ приведено в Приложении А.

2.2.3 Структура организации менеджера проходов

Менеджер проходов является компонентом системы SAPFOR 2, который отвечает за запуск тех или иных проходов анализа и преобразования. Данный компонент содержит внутри себя информацию о том, какие прямые зависимости есть между существующими проходами. Реализация компонента содержит две функции. Первая функция является рекурсивной и запускает все необходимые проходы для выполнения данного. Вторая функция выполняет непосредственно сам проход по двум фазам: в первой фазе происходит обработка AST отдельно и независимо по файлам программы, во второй фазе происходит агрегация результатов работы либо первой фазы, либо других проходов. Проход может содержать в себе реализацию как двух фаз, так и какой-то одной.

Для добавления нового прохода необходимо создать его объявление, добавить создаваемому проходу зависимости от других проходов (если они имеются) и описать его в основном цикле запуска проходов. Для объявления прохода необходимо сопоставить ему уникальное имя, а также добавить его в структуру менеджера проходов.

Для добавления зависимостей существуют некоторые правила. Например, $Pass(A) \leqslant Pass(B)$ означает, что проход B зависит от прохода A, то есть при выполнении прохода B менеджер проходов автоматически запустит проход A. $List(Pass_1, Pass_2, ..., Pass_N) \leqslant Pass(A)$ означает, что проход зависит от всех проходов из списка $List$. $Pass(A) \leqslant List$ означает, что каждый проход из списка зависит от прохода A. $List(A) << List(B)$ означает, что каждый проход из списка B зависит от всех проходов из списка A. Далее необходимо создать функцию, выполняющую алгоритм прохода, и вставить ее в одну или другую фазу запуска проходов.

Реализованная структура организации проходов позволяет легко расширять функционал системы SAPFOR 2, используя результаты уже существующих проходов. Для добавления нового алгоритма анализа или преобразования совсем не обязательно разбираться в существующем коде всей системы SAPFOR 2.

2.2.4 Модуль визуализации результатов анализа и преобразования кода

Неотъемлемой частью системы SAPFOR 2 является модуль визуализации результатов анализа и преобразований. По сути, этот модуль представляет собой отдельную систему, которая содержит в себе три основные возможности. Визуализация результатов работы модуля анализа и преобразований не является предметом данной диссертации, поэтому в диссертации приводится только описание концепций построения визуальной части системы.

Во-первых, система визуализации отображает проект пользователя и фактически заменяет собой текстовый редактор кода. Конечно, набор функций и возможностей такого редактора ограничены (в отличие от, например, Visual Studio, Code Blocks и др.).

Во-вторых, система позволяет запускать основные этапы анализа и преобразования посредством обращения к модулю анализа и преобразования кода с помощью интерфейсных функций, обеспечивает отображение их результатов, а также обеспечивает удобную работу с полученными после преобразований новыми версиями программы.

И наконец, третья, немаловажная функция – система визуализации позволяет компилировать и выполнять текущую программу и ее параллельные версии на локальной или удаленной машинах.

Использование интерактивных средств взаимодействия с пользователем в процессе распараллеливания позволяет снизить сложность ручного распараллеливания, прибегая к автоматизации рутинной работы, например, однотипному преобразованию кода.

Система SAPFOR 2 нацелена на автоматизацию отображения последовательных программ на гибридные вычислительные кластеры с использованием модели DVMH. Можно выделить следующие задачи, для решения которых в системе требуется интерактивность [20]:

– исследование информационной структуры распараллеливаемой программы, представление ее в виде различных абстракций, включающих текстовое и графовое представления;

– задание свойств программы (спецкомментарии), знание которых необходимо для автоматического принятия решений по распараллеливанию, но которые не могут быть получены автоматически, например, из-за требования консервативности средств статического анализа;
– ручной выбор преобразований программы, которые должны быть выполнены с целью устранения проблем, препятствующих распараллеливанию, если автоматически устранить данные проблемы не удается;
– частичное распараллеливание программы пользователем, например, за счет выбора распределения данных или ручного распараллеливания отдельных фрагментов программы;
– инкрементальное распараллеливание на кластер с использованием механизма областей распараллеливания.

2.2.5 Спецкомментарии системы SAPFOR 2

Для преодоления ограничений анализа системы SAPFOR 2 были введены спецкомментарии. Данные спецкомментарии пользователь вставляет в исходный код программы в случае консервативности или невозможности статического анализа. Также пользователь может корректировать принятые системой решения по своему усмотрению.

Всего было введено четыре группы спецификаций: анализ (ANALYSIS), спецификации для потенциально параллельного цикла (PARALLEL), преобразования (TRANSFORM) и спецификации для расстановки областей распараллеливания (PARALLEL_REG). Полный синтаксис спецификаций приведен в Приложении Б. Данные директивы могут быть расширены в случае необходимости, например, для реализации новых преобразований программы без внутреннего анализа.

Листинг 2.1 — Пример расстановки спецкомментариев системы SAPFOR 2

```
1 !$SPF ANALYSIS(PRIVATE(K,J,I,i1))
2 !$SPF PARALLEL(REDUCTION(MAX(EPS))))
3 DO K = 2, L-1
```

```
4     DO J = 2, L-1
5       DO I = 2, L-1
6         i1 = i
7         EPS = MAX(EPS, ABS(B(I,J,K) - A(I,J,K)))
8         A(I,J,K) = B(i1,J,K)
9       ENDDO
10    ENDDO
11  ENDDO
```

В Листинге 2.1 приведен пример кода, демонстрирующий применение спецкомментариев в системе SAPFOR 2. В случае тесного гнезда циклов спецкомментарии могут быть распространены на вложенные циклы автоматически.

2.2.6 Возможности языка Fortran-DVMH

Язык Fortran-DVMH (FDVMH) [1] представляет собой язык Фортран 95, расширенный спецификациями параллелизма. Эти спецификации оформлены в виде специальных комментариев, которые называются директивами. Директивы FDVMH можно условно разделить на три подмножества:

- Распределение данных (директивы DISTRIBUTE, REDISTRIBUTE, ALIGN, REALIGN)
- Распределение вычислений (директива PARALLEL)
- Спецификация удаленных данных (директивы SHADOW_RENEW, REMOTE_ACCESS, ACROSS, REDUCTION)

Модель параллелизма FDVMH базируется на специальной форме параллелизма по данным: одна программа – множество потоков данных. В этой модели одна и та же программа выполняется на каждом процессоре, но каждый процессор выполняет свое подмножество операторов в соответствии с распределением данных.

В модели FDVMH пользователь вначале определяет массивы, которые должны быть распределены между процессорами (распределенные

данные). Эти массивы специфицируются директивами отображения данных (DISTRIBUTE, ALIGN). Остальные переменные (распределяемые по умолчанию) отображаются по одному экземпляру на каждый процессор (размноженные данные). Размноженная переменная должна иметь одно и то же значение на каждом процессоре за исключением переменных в параллельных конструкциях. Модель FDVMH определяет два уровня параллелизма: параллелизм по данным и параллелизм задач. Системой SAPFOR 2 реализуется только первый уровень параллелизма.

Параллелизм по данным реализуется распределением витков тесногнездового цикла между процессорами. При этом каждый виток такого параллельного цикла полностью выполняется на одном процессоре. Операторы вне параллельного цикла выполняются по правилу **собственных вычислений** – каждый процессор выполняет только вычисления собственных данных, то есть тех данных, которые распределены на этот процессор. Это реализовано следующим образом.

Рассмотрим условный оператор, который находится вне параллельного цикла (см. Листинг 2.2), где $COND$ – логическое выражение, $LeftPart$ – левая часть оператора присваивания (ссылка на скаляр или элемент массива), $RightPart$ – правая часть оператора присваивания (выражение).

Листинг 2.2 — Условный оператор

```
1 IF  ( COND )  THEN
2    LeftPart = RightPart
3 ENDIF
```

Тогда этот оператор будет выполняться на процессоре, где распределены данные со ссылкой $LeftPart$. Все данные в выражениях $COND$ и $RightPart$ должны быть размещены на этом же процессоре. Если какие-либо данные из выражений $COND$ и $RightPart$ отсутствуют на этом процессоре, то их необходимо указать в директиве удаленного доступа (REMOTE_ACCESS) перед этим оператором.

Если $LeftPart$ является ссылкой на распределенный массив и существует зависимость по данным между $RightPart$ и $LeftPart$ (за исключенем зависимостей с одинаковыми обращениями к массиву, например, $A(5) = A(5) + 10$), то распределенный массив необходимо размножить с помощью директивы REDISTRIBUTE или REALIGN перед выполнением рассматриваемого оператора.

При вычислении значения собственной переменной в параллельных циклах процессору могут потребоваться как значения собственных переменных, так и значения несобственных (удаленных) переменных. Все удаленные переменные должны быть указаны в директивах доступа к удаленным данным (REMOTE_ACCESS, SHADOW_RENEW, ACROSS, REDUCTION).

Глава 3. Построение дерева циклов и графа функций

Дерево циклов и граф вызовов функций программы являются базовыми и необходимыми структурами для реализации разного рода анализов и преобразований. Рассмотрим построение дерева циклов и графа вызовов функций.

3.1 Построение дерева циклов

Для каждого файла строится свое дерево циклов. Оно представляет собой список из циклов верхнего уровня, которые содержат в себе вложенные циклы. В данное дерево объединяются только циклы с прямой вложенностью. Прямая вложенность – такая вложенность циклов без учета процедур и функций (внутри которых также могут быть циклы). Цикл описывается следующей основной информацией:

- номер строки начала цикла и номер строки конца цикла;
- уровень тесной вложенности. Если цикл не тесно-вложенный, то уровень равен 1, иначе – количеству циклов в гнезде тесной вложенности;
- информация об операторах *GOTO* в цикле, а также список строк таких операторов (отмечаются переходы из цикла и извне в цикл);
- информация об операторах ввода/вывода, а также список номеров строк таких операторов;
- информация об операторах останова, а также список номеров строк таких операторов;
- информация о зависимостях по скалярам неопределенного типа – зависимости, которые не относятся к следующим типам устранимых зависимостей: приватная и редукционная (1);
- информация о зависимостях по массиву неопределенного типа – такие зависимости, которые не являются регулярными зависимостями по массиву (2);

- информация о присваиваниях в элементы массива неопределенного типа – такие присваивания в элементы массива, которые система SAPFOR 2 не может связать с итерационной переменной цикла (3);
- директива распределения вычислений в модели DVMH для этого цикла, которая может быть получена в результате его распараллеливания. По умолчанию какая-либо директива отсутствует;
- список вложенных циклов (ссылка на такую же структуру) и ссылка на цикл-родитель;
- список вызовов функций из данного цикла;
- информация об обращениях к массиву с использованием итерационной переменной цикла в индексных выражениях на чтение/запись;
- информация о конфликтных обращениях к массивам с использованием итерационной переменной этого цикла в индексных выражениях. Например, конфликт может возникать в том случае, если две итерационные переменные двух разных циклов присутствуют в одном индексном выражении в обращении к массиву (4);
- ссылка на область распараллеливания (если цикл включен в область, то она будет указывать на нее, иначе – пустая ссылка);
- ссылка на результаты анализа зависимостей в цикле.

Среди всех типов циклов, которые можно использовать в языке Фортран (в том числе циклы, определяемые через $GOTO$), в описанную структуру преобразуются только циклы $DO - ENDDO$, у которых обязательно есть границы (шаг может отсутствовать и по умолчанию равен единице). Циклы DO с меткой переводятся в системе SAPFOR 2 с помощью реализованного преобразования в циклы $DO - ENDDO$. Данное преобразование необходимо делать из-за удобства анализа циклов в системе Sage++.

Потенциально параллельным циклом будем называть такой цикл в формате $DO-ENDDO$ с ненулевым количеством витков, который может быть автоматически преобразован в параллельный цикл системой SAPFOR 2 с помощью вставки перед его описанием DVM-директивы распределения вычислений. Такой цикл должен обладать следующими дополнительными свойствами:

- не содержать операторов перехода $GOTO$ внутрь цикла и за его пределы;

- не содержать операторов ввода/вывода (*PRINT*, *WRITE*, *OPEN*, и т.д.) (5);
- не содержать операторов останова (6);
- не содержать описанные выше свойства (1)-(4), обозначим данное требование как (7).

Ясно, что переходы, операции ввода/вывода и операторы останова делают невозможным параллельное выполнение цикла, так как будет нарушена логическая структура программы в силу независимого выполнения витков цикла разными нитями или процессами. Свойства (1)-(4) делают невозможным параллельное выполнение цикла по причине наличия зависимостей неопределенного типа, которые нельзя устранить существующими средствами DVMH-модели, так как статический анализ в некоторых случаях неспособен определить тип, вид или отсутствие зависимости.

Построение дерева циклов выполняется в два прохода. Первый проход выполняется над всеми файлами проекта и строит деревья циклов для каждой функции независимо. При первом проходе также формируются связи *родитель-потомок* и добавляются вызываемые из тела цикла функции. Можно заметить, что для описанных выше свойств (5)-(7) требуется межпроцедурный анализ, так как операторы печати или останова программы могут содержаться внутри вызываемых функций из тела цикла. Второй проход реализует описанный межпроцедурный анализ, который дополняет свойства цикла с учетом вложенных вызовов функций.

3.2 Построение графа вызовов функций

В отличие от дерева циклов, граф вызовов функций представляет собой неориентированный граф, узлами которого являются объявления функции или процедуры, а ребра отражают связи между этими функциями, порожденные операторами вызова процедур или функций в исходном коде программы. Граф вызовов функций строится для всего программного комплекса, который может состоять из нескольких файлов. Поэтому построение графа функций состоит также из двух проходов. Первый проход

обрабатывает каждый файл по отдельности, затем второй проход выполняет объединение результатов выполненного в первом проходе анализа.

Далее для удобства восприятия будем отождествлять понятия подпрограммы (subroutine) и функции (function), так как в Фортране функция является частным случаем подпрограммы, имеющим возвращаемое значение. Узел графа вызовов функций описывается следующей информацией:

- имя функции;
- строки начала и конца функции в исходном коде программы;
- имя файла, в котором данная функция объявлена;
- указатель на функцию в системе Sage++;
- список функций, которые вызываются из данной функции;
- список функций, которые вызывают данную функцию;
- список common-блоков, используемых в данной функции;
- информация о параметрах функции;
- следующие флаги:
  - флаг, который показывает необходимость или отмену подстановки функции (назовем его *doNotInline*). Данный флаг получает положительное значение в результате применения директивы SAPFOR 2, которая отменяет подстановку функции;
  - флаг, который показывает необходимость или отмену анализа данной функции, например, на предмет подстановки (назовем его *doNotAnalyze*);
  - флаг, показывающий требование подстановки функции (назовем его *needToInline*). Данный флаг выставляется в положительное значение в результате межпроцедурного анализа в системе SAPFOR 2;
  - флаг, показывающий, является ли данная функция «мертвой» в рассматриваемом программном комплексе (назовем его *deadFunction*). «Мертвой» называется функция, которая объявлена, но не используется ни в одном операторе программы. Анализировать такие функции не имеет смысла.

Во время первого прохода для каждого файла выполняется анализ всех объявленных функций. Каждая функция добавляется в граф вызовов функций. Если при функции есть директива $!$*SPF TRANSFORM*(*NO_INLINE*), то для этой функции добавляется флаг *doNotInline*. Далее осуществляется проход по всем операторам данной функции и поиск всех функций, которые вызываются из рассматриваемой. Они будут добавлены в список вызываемых из данной функции. Можно заметить, что объект в графе функций создается только во время просмотра функции в файле, следовательно, пока не будут обработаны все файлы проекта, нельзя выполнять какую-либо обработку функций, например, нельзя сформировать список функций, которые вызывают рассматриваемую функцию.

После того, как был выполнен первый проход, выполняется второй проход, объединяющий и дополняющий результаты. Сначала выполняется отбраковка функций, которые содержатся в проекте, но не вызываются ни из одной функции. Если найдена такая функция, для нее выставляются флаги *deadFunction* и *doNotAnalyze*. Такая функция является «*мертвой*» (за исключением главной программной единицы – *PROGRAM* в языке Фортран). Если функция *мертвая*, то выдается предупреждение о том, что функция не вызывается в данном проекте. Таким образом, для анализа и распараллеливания системой SAPFOR 2 допускаются только полноценные программные комплексы, имеющие главную программную единицу.

Далее выполняется проверка того, что функции не являются рекурсивными, а также не содержатся в цепочке вызовов, приводящей к циклической рекурсии. Для этого строятся цепочки вызовов всех функций по построенной в первом проходе информации, и ищутся циклы в цепочке вызовов. Если цикл найден, то выдается предупреждение о том, что данная функция (с которой начинался поиск) участвует в рекурсии, такой функции выставляется флаг *doNotAnalyze*. Для каждой функции алгоритм пытается проверить, является ли она рекурсивной, либо состоит в рекурсивной цепочке. Рекурсивные функции не поддерживаются системой SAPFOR 2 и исключаются из дальнейшего рассмотрения.

Наиболее важным анализом является проверка функции на необходимость её подстановки в местах вызова. На данный момент как система

DVM, так и система SAPFOR 2 имеют ограничения на вызовы пользовательских процедур или функций внутри параллельного цикла (цикла, перед которым стоит DVM-директива распределения вычислений), а также на использование таких циклов внутри пользовательских процедур или функций. Данное ограничение, прежде всего, связано с организацией работы с массивами в программе. Рассмотрим данные ограничения подробнее.

Листинг 3.1 — Пример ошибочного использования DVM-массивов и процедур

```
1  SUBROUTINE someProc (a)
2  ...
3  !DVM$  PARALLEL(k,j,i) ON a(i,j,k)
4  DO k = 1, nz
5    DO j = 1, ny
6      DO i = 1, nx
7        A(i,j,k) = ...
8      END DO
9    END DO
10 END DO
11 END
12
13 SUBROUTINE errorCall
14 !DVM$  DISTRIBUTE(BLOCK, BLOCK, BLOCK) :: a_dist
15 double precision a_dist(nx, ny, nz)
16 double precision a_notDist(nx, ny, nz)
17   ...
18   CALL someProc(a_dist)
19   CALL someProc(a_notDist) ! ERROR
20 END
```

В модели DVMH существуют понятия *распределенный* массив и *нераспределенный* массив. Распределенным массивом называется такой массив, который был распределен средствами DVM с помощью директив *DISTRIBUTE* или *ALIGN*. Нераспределенным называется такой массив, который не участвует в распределении средствами DVM. Таким образом, несмотря на то, что текстуально использование распределенных и нераспределенных массивов в программе не отличается, существует отличие в момент отображения программы в ее параллельную версию в модели MPI, OpenMP и CUDA с помощью DVMH-компилятора.

После того, как DVMH-программа будет преобразована в параллельную программу с использованием средств MPI, OpenMP и CUDA, а также в набор вызовов функций системы поддержки DVMH, распределенные и нераспределенные массивы будут представлены по-разному. Таким образом, использование одного «экземпляра» процедуры (см. Листинг 3.1), в параметры которой может быть передан как распределенный, так и нераспределенный массив, недопустимо в рамках DVMH-модели. Решением данной проблемы является подстановка таких процедур в точки их вызовов в циклах, либо дублирование данной функции и замена соответствующих вызовов.

Для реализации проверки на необходимость подстановки функций в точки их вызова непосредственно в телах циклов был реализован следующий проход, который позволяет проверить, что все функции пользователя, участвующие в распараллеливании, удовлетворяют ограничениям, введенными в системах SAPFOR 2 и DVM. Если ограничения не выполнены, система SAPFOR 2 через диалог информирует о том, что необходимо подставить те процедуры, которые препятствуют продолжению анализа и распараллеливанию, при этом анализ прекращается. Накладываются следующие ограничения:

- в потенциально параллельных циклах нет вызовов пользовательских функций, в которые в качестве аргументов передаются массивы (или их элементы), которые, в свою очередь, не являются приватизируемыми для данного цикла (приватизируемые массивы не будут распределенными в терминах DVMH-модели);
- в потенциально параллельных циклах нет вызовов пользовательских функций, в которые передается итерационная переменная этого цикла, которая, в свою очередь, в теле функции содержится в индексных выражениях в обращениях к распределяемым массивам.

Для проверки данных ограничений анализируются все циклы из дерева циклов. Для каждого цикла, входящего в какую-либо область распараллеливания, просматриваются все функции, которые вызываются в данном цикле. Если функция была помечена признаком NO_INLINE, то она пропускается. Иначе анализируется список фактических аргументов в вызове функции:

– если среди фактических аргументов функции присутствует итерационная переменная цикла и в теле функции с учетом вложенности вызовов данная переменная содержится в индексных выражениях в обращениях к распределяемым массивам, то выдается сообщение об ошибке, которое содержит в себе информацию о том, что функция должна быть подставлена, так как внутрь передается итерационная переменная цикла;
– если есть обращение к массиву и этот массив является распределяемым, то выдается сообщение об ошибке, которое содержит в себе информацию о том, что функция должна быть подставлена, так как есть обращение к массиву.

Во всех остальных случаях функция не нуждается в подстановке и никаких диагностик выдаваться не будет.

Также проблему подстановки функций можно решить с помощью их дублирования (см. Листинг 3.1). В таком случае требуется сделать две копии процедуры на время анализа, а затем выполнить объединение копий процедур после создания параллельной версии. Дублирование процедур позволяет разрешить конфликты анализа, возникающие при передаче различных данных (в основном – распределяемых массивов).

Дублирование процедур и их объединение были реализованы в качестве дополнительных проходов преобразования и анализа соответственно в системе SAPFOR 2. В общем случае выполняется полное дублирование всех цепочек вызовов функций. Для этого анализируются все цепочки вызовов функций, выполняется физическое их дублирование и замена всех вызовов функций из дублируемой функции на соответствующие им копии.

Главный недостаток такого подхода – сильное разрастание кода в случае большой вложенности вызовов функций, но большим плюсом является «уникальность» каждой процедуры и передаваемых ей данных. Таким образом разрешаются конфликты в случае невозможности межпроцедурного анализа.

После того, как была построена параллельная версия программы, выполняется объединение процедур. Объединяются процедуры, которые имеют полностью одинаковые тела с учетом вызова внутренних процедур. В результате будет создано столько копий процедур, сколько необходимо для корректной расстановки директив распределения вычислений.

Глава 4. Построение областей распараллеливания

4.1 Понятие области распараллеливания

Областью распараллеливания (ОР) будем называть совокупность исполняемых операторов исходного кода программы, описываемых с помощью фрагментов. Фрагмент – это множество исполняемых операторов в рамках одной области вложенности (функция, процедура, цикл, условный оператор и т.д.) с одним входом и несколькими выходами. По умолчанию (если не было обозначено никаких других областей) вся программа рассматривается как одна область распараллеливания с именем $DEFAULT$, которая содержит в себе все исполняемые операторы.

В случае наличия пользовательских ОР область по умолчанию не рассматривается. Введем понятие вложенности и пересечения областей. В рамках одной области видимости определение данных понятий является тривиальным. Для процедур и их вызовов данные понятия определяются следующим образом.

Листинг 4.1 — Пример вложенных областей распараллеливания

```
1  SUBROUTINE A
2  !$SPF PARALLEL_REG regA
3  ...
4  call B
5  !$SPF END PARALLEL_REG
6  END
7
8  SUBROUTINE B
9  !$SPF PARALLEL_REG regB
10 ...
11 !$SPF END PARALLEL_REG
12 END
```

Пусть есть три процедуры A, B, C. Явным вызовом процедуры B из процедуры A будем называть вызов процедуры B непосредственно из тела процедуры A. Косвенным вызовом процедуры B из процедуры A будем называть такой вызов процедуры B, который содержится в каком-либо

теле процедур, вызываемых непосредственно из тела процедуры A, но при этом отсутствует явный вызов процедуры B из процедуры A.

Под явным объявлением области распараллеливания будем понимать выделение последовательности исполняемых операторов с помощью пары директив !$*SPF PARALLEL_REG* – !$*SPFEND PARALLEL_REG*. Две различные ОР будут вложенными в том случае, если явное объявление ОР #1 находится в процедуре A, и явное объявление ОР #2 находится в процедуре B, причем B вызывается из процедуры A явно или косвенно (см. пример в Листинге 4.1).

ОР будут пересекающимися по процедуре C, если явное объявление ОР #1 находится в процедуре A, и явное объявление ОР #2 находится в процедуре B, причем A не вызывает B, а B не вызывает A ни явно, ни косвенно, но и A и B вызывают процедуру C явно или косвенно (см. пример в Листинге 4.2).

Листинг 4.2 — Пример пересекающихся областей распараллеливания

```
1  SUBROUTINE A
2  !$SPF PARALLEL_REG regA
3        ...
4        CALL C
5  !$SPF END PARALLEL_REG
6  END
7
8  SUBROUTINE B
9  !$SPF PARALLEL_REG regB
10       ...
11       CALL C
12 !$SPF END PARALLEL_REG
13 END
14
15 SUBROUTINE C
16       ...
17 END
```

Каждая область имеет свой уникальный идентификатор, называемый именем области. Совокупность разных фрагментов (внутри функции, а также в разных файлах) с одинаковым именем рассматриваются системой SAPFOR 2 как одна область распараллеливания. Для каждой из областей

распараллеливания с разными именами строятся разные и независимые решения по распределению данных и вычислений.

4.2 Правила расстановки областей распараллеливания и разрешения конфликтов

Вложенные области распараллеливания в системе SAPFOR 2 не допускаются ни в рамках одной области видимости, ни между процедурами. Поэтому для снятия данных ограничений, а также реализации концепции областей распараллеливания, в системе SAPFOR 2 были реализованы преобразования, приводящие программу в соответствующий вид. Рассмотрим данные алгоритмы преобразования.

Как известно, *распределенные* и *нераспределенные* массивы в модели DVMH отличаются по использованию их в программе. Чтобы ограничить область действия распределенных средствами DVM массивов, введем соответствующие правила преобразования явных областей распараллеливания. Для каждой ОР необходимо заменить используемые в ней массивы на массивы-копии по следующим правилам. Для случая массивов, передаваемых в качестве параметров процедуре или функции:

- необходимо добавить локальные массивы такого же размера, как и массивы, передаваемые в процедуру в качестве параметров. Такие массивы мы будем называть **массивами-копиями**;
- для всех параметров процедуры, которые являются массивами, необходимо сделать их переименование в теле процедуры для тех операторов, которые используются в области распараллеливания;
- необходимо добавить копирование входных массивов в новые массивы-копии после входа в область распараллеливания, и добавить копирование выходных массивов из массивов-копий после выхода из нее.

Если необходимо заменить глобальные массивы, используемые в common-блоке, то создается новый common-блок и туда помещаются массивы-копии для каждого массива из соответствующего common-блока. В случае использования модуля достаточно создать массивы-копии в данном

модуле. Для каждого массива из common-блока или модуля создается только один массив-копия вне зависимости от количества ОР, в котором он может использоваться. По аналогии с локальными массивами в глобальные массивы-копии копируются исходные массивы до входа в область, и выполняется обратное копирование после выхода из области. Таким образом, массивы-копии призваны «ограничить» действия системы SAPFOR 2 в рамках выделенных фрагментов распараллеливания.

Если проект состоит из нескольких функций и нескольких файлов, то возникает возможность разной расстановки ОР. Считается, что, если ОР содержит в себе вызовы процедур или функций, то тела данных функций или процедур автоматически включаются в данную область. Если ОР располагается внутри процедуры, и она покрывает все исполняемые операторы, то все вызовы этой процедуры автоматически считаются включенными в эту область. В связи с этим возникают следующие случаи:

- первый и самый простой случай – области распараллеливания не пересекаются даже с учетом вложенности процедур. В данном случае ничего дополнительно предпринимать не нужно;
- области не пересекаются, но есть процедура, вызов которой есть и вне всех ОР, и внутри одной ОР. По правилам языка FDVMH данная процедура не может принимать и распределенные, и нераспределенные массивы. Таким образом, в этой процедуре нельзя использовать $INHERIT$ указание (в модели DVMH данное указание позволяет унаследовать распределение массива, переданного в качестве параметра в процедуре). Значит необходимо сделать копию процедуры с учетом вложенных процедур, а также заменить вызов процедуры в тех ОР, где она используется;
- области пересекаются по некоторым процедурам, то есть имеются общие процедуры, которые вызываются из разных областей. Данные процедуры будем называть конфликтными. В общем случае каждую из конфликтных процедур необходимо продублировать необходимое количество раз, и заменить соответствующие вызовы в выделенных областях распараллеливания. Дублирование процедуры не требуется, если конфликтные процедуры содержат

обращения только к нераспределенным массивам (например, приватным или локальным в этой процедуре) или вообще не содержат обращений к массивам.

Создание массивов-копий в процедурах выполняется на верхнем уровне, то есть создавать массивы-копии для массивов, которые передаются через параметры, во вложенных процедурах нет необходимости, так как такие массивы-копии уже были созданы ранее, а необходимые процедуры – продублированы.

4.3 Объединение областей распараллеливания

Объединение двух областей позволяет распространить найденные в одной области решения на другую область. Область распараллеливания, в которую делается объединение, будем считать главной ОР, а область распараллеливания, которая объединяется – поглощаемой ОР. Можно отметить, что найденные решения по распараллеливанию в главной ОР не обязаны совпадать с найденными решениями в поглощаемой области.

Для того, чтобы принятые решения в главной области были определяющими, необходимо выполнить следующее объединение данных главной и поглощаемой областей распараллеливания:

- уникальный идентификатор поглощаемой области становится таким же, как и у главной области (аналогично для пользовательского идентификатора);
- диапазоны строк пользовательской программы в главной области дополняются диапазонами строк поглощаемой области;
- список всех вызываемых функций главной области дополняется списком вызываемых функций поглощаемой области;
- в полный граф массивов главной области добавляется информация из полного графа массивов поглощаемой области, а также к усеченному графу массивов главной области добавляется информация из полного графа массивов поглощаемой области только тех массивов, которые не входили в главную область изначально. Последнее требуется для переноса решений, принятых в поглощаемой области по

отношению к тем массивам, которые не были включены в главную область. После этого вызывается алгоритм устранения конфликтов (см. Глава 5) для объединенного усеченного графа массивов для его уточнения;

- список распределенных массивов главной области дополняется списком распределенных массивов поглощаемой области;
- во всех циклах, которые ссылаются на поглощаемую область распараллеливания, необходимо поменять ссылки на главную область.

После выполнения данной процедуры объединения, поглощаемая область удаляется из структур системы SAPFOR 2, а главная область теперь содержит объединенную информацию по двум областям.

Можно отметить, что в случае рассмотрения двух областей распараллеливания по отдельности, а затем их объединения, принятые решения могут отличаться от случая, если бы эти две области были изначально определены как одна область распараллеливания, так как, например, количество циклов и их значимость в зависимости от расстановки областей будут разные.

Глава 5. Построение схем распределения данных и вычислений

5.1 Способы отображения на кластер

Распределение данных – одна из основных проблем отображения последовательной программы на кластер. Для эффективного использования мощности кластера требуется учитывать специфику обработки данных в циклах последовательной программы, так как зачастую в них содержится основная вычислительная нагрузка. Рассмотрим следующие варианты отображения последовательной программы на кластер:

- отображение выполняется с размножением данных, но с распределением вычислений. При таком подходе существенно упрощается преобразование последовательной программы в параллельную для кластера, так как все данные дублируются на каждом узле кластера. В свою очередь, каждый цикл может «требовать» свое распределение, что легко реализуется данной моделью;
- отображение выполняется с распределением данных и вычислений. При таком подходе необходимо учитывать интересы всех циклов, участвующих в распараллеливании. В силу того, что на каждом узле присутствует только часть данных, необходимо выполнять их пересылки для корректного выполнения последовательных участков программы.

Система SAPFOR 2 преобразует исходную программу в параллельную программу с использованием модели DVMH. Данная модель использует второй вариант – отображение как данных, так и вычислений на узлы кластера. В связи с этим необходимо обеспечить такое распределение данных, чтобы количество пересылок, а также их объем между процессорами были как можно меньше. Можно отметить, что с помощью модели DVMH можно реализовать и первый вариант, но, во-первых, параллельная программа будет требовать столько же памяти, что и последовательная, и, во-вторых, эффективность выполнения такой программы может быть ниже из-за больших коммуникаций.

Алгоритм распределения данных состоит из двух этапов. На первом этапе выполняется проход по всем файлам и анализируются циклы и используемые в них массивы. На втором этапе – обработка полученных результатов и поиск оптимального распределения данных.

5.2 Анализ обращений к массивам в программе

5.2.1 Построение связей циклов и массивов

Для каждой функции файла программы выполняется анализ всех ее операторов. Перед началом анализа операторов функции строится множество ссылок на common-блоки, содержащиеся в данной функции. Далее выполняется проход по всем операторам данной функции. Рассматриваются только те операторы, которые попадают в какую-либо область распараллеливания. Если оператор не попадает ни в какую область, то он пропускается. Для всех операторов, которые попадают в область, выполняется следующее:

- если оператор является циклом – запоминаем его в список текущих операторов цикла. Также пополняем список приватных переменных из спецкомментариев перед циклом (!$SPF ANALYSIS(PRIVATE(...)));
- если оператор является концом составного оператора (CONTROL_END, см. Приложение A), то, если это конец цикла – убираем последний цикл из списка текущих операторов цикла;
- если оператор является обращением к процедуре/функции – проверяем все фактические аргументы функции: при передаче в функцию неприватного массива или его элементов – выдается ошибка и анализ прекращается (массив считается приватным, если он указан в директиве в !$SPF ANALYSIS(PRIVATE(...)) перед циклом);
- если оператор – присваивание или IF, или ELSEIF, или логический IF, то анализируем левую часть оператора (в случае присваивания)

и правую часть, а также условия для IF-операторов на предмет поиска обращений к массиву. Для каждого обращения к массиву отдельно и независимо для каждого из измерений выполняется следующий анализ:

- проверяем, есть ли косвенная адресация в обращении. Если нет, то пытаемся найти итерационные переменные циклов в обращении по рассматриваемому измерению (используется список циклов, который формируется по операторам цикла);
- если найдена связь с более чем одним циклом, то выводится предупреждение, а также для каждого цикла из списка текущих циклов добавляется информация о том, что по рассматриваемому массиву по обрабатываемому измерению есть неопределенная операция чтения/записи (в данном случае присутствует более, чем одна итерационная переменная цикла в индексном выражении в обращении к массиву);
- если невозможно определить связь итерационных переменных циклов с индексными выражениями в обращении к массиву, то выводится предупреждение;
- если имеется связь с одним циклом, то пытаемся сопоставить индексное выражение с итерационной переменной цикла x шаблон $D * x + E$ и вычислить данные коэффициенты. Если вычислить не удалось, а также, если коэффициент $D < 0$ (инверсное распределение данных не поддерживается компилятором Fortran-DVMH), то выводится предупреждение, иначе добавляем информацию о том, что итерационная переменная цикла содержится в индексном выражении в обращении к массиву на запись/чтение с коэффициентами (D, E);
- после проверки всех измерений массива, если данное обращение к массиву используется на запись, проверяем факт того, что все индексные выражения в обращении к массиву имеют хотя бы одну итерационную переменную цикла. Если итерационная переменная цикла не встречается в

индексных выражениях, то добавляем в граф циклов информацию о том, что была неопределенная запись в массив по рассматриваемому циклу.

– для всех остальных исполняемых операторов выполняется заполнение списка приватных переменных из директивы !$SPF ANALYSIS (PRIVATE(...)).

Важно отметить, что вся информация привязывается к циклам, то есть для каждого цикла формируется список всех обращений к массивам. Для того, чтобы цикл был оптимально распараллелен системой SAPFOR 2, требуется, чтобы каждое индексное выражение в обращении по конкретному измерению массива содержало только одну итерационную переменную цикла или иными словами должно быть однозначное отображение измерений массива на циклы.

После обработки всех операторов функции будет построена информация о «хороших» обращениях к массивам с привязкой этих обращений к циклам с коэффициентами $D * x + E$, где D, E – вычисленные константы, $D \neq 0$, а x – итерационная переменная цикла. Остальные обращения к массивам будут порождать неизбежные обмены между узлами кластера. Рассмотрим пример получаемых данных по вышеописанному алгоритму (см. Листинг 5.1).

Листинг 5.1 — Пример цикла последовательной программы

```
1 DO K = 2, L-1
2   DO J = 2, L-1
3     DO I = 2, L-1
4       B(I,J,K)=( A(I,J,K-1) + A(I,J-1,K) + A(I-1,J,K) +
5                  A(I+1,J,K) + A(I,J+1,K) + A(I,J,K+1)  )/6.
6     ENDDO
7   ENDDO
8 ENDDO
```

В рассматриваемом фрагменте кода есть три цикла с итерационными переменными K, J, I. Также есть один оператор присваивания, в левой части которого есть обращение к массиву B, а в правой – к массиву A. Для каждого индексного выражения массивов A и B будет сформирована информация с привязкой к соответствующему циклу. Например, для

выражения $B(*,J,*)$ будут вычислены коэффициенты $D = 1, E = 0$ и привязаны к информационной структуре цикла по J.

Всего будет вычислено три пары коэффициентов для массива B, которые будут равны (1,0) для каждого индексного выражения, и девять пар для массива A: (1, 0), (1, 1), (1, -1) – по три пары для каждого из изменений в обращениях. Все индексные выражения в обращениях к массивам обрабатываются независимо, так как они связываются с различными циклами.

После того, как все функции файла будут обработаны, происходит объединение полученной информации о записях/чтениях/конфликтах для каждого из циклов во всем дереве циклов. Затем происходит объединение информации о записях по дереву циклов – для всех тесно-гнездовых циклов информация с более низких потомков поднимается к родителю.

Для поиска зависимостей в циклах используется реализованный в библиотеке Sage++ анализ зависимостей на базе Омега теста [45; 46]. Данный анализ запускается для каждого цикла программы (если цикл содержится в какой-либо области распараллеливания). Полученная информация (граф зависимостей) сохраняется в дереве циклов в том случае, если найденные зависимости не влияют на свойство потенциальной параллельности цикла, то есть не отменяют возможность его распараллеливания (см. Глава 3). Анализ позволяет находить зависимости по массивам и по скалярам. По скалярам зависимости классифицируются следующим образом:

- устранимая зависимость по скаляру с помощью приватизации переменной;
- устранимая зависимость по скаляру с помощью редукционной операции;
- неустранимая зависимость по скаляру неопределенного типа – данная информация добавляется в дерево циклов, если эта переменная не была специфицирована пользователем как приватная через !$SPF ANALYSIS (PRIVATE(...)).

По массивам зависимости классифицируются следующим образом:

- если есть зависимость по массиву с расстоянием неопределенного размера, то в дерево циклов добавляется информация о том, что есть зависимости по массивам неопределенного типа;
- если известен тип зависимости и ее длина, а также, это FLOW или ANTI зависимость, такая зависимость классифицируется как

ACROSS-зависимость по массиву в терминах DVMH-модели в соответствующем измерении массива, только если в индексном выражении в данном измерении содержится итерационная переменная цикла (то есть присутствует однозначное отображение на данный цикл).

5.2.2 Построение графа массивов

Граф массивов является очень важной структурой данных, так как все алгоритмы распределения данных и вычислений используют данную структуру. Граф массивов заполняется после анализа всех обращений к массивам в циклах программы. Для того, чтобы выбрать определенный формат хранения графа, а также реализовать обработку полученного графа эффективным образом, были выполнены следующие исследования в области параллельной обработки графов [14—19].

Построение графа массивов обусловлено выбором целевой модели при создании параллельной версии программы. DVMH-модель требует нахождения всех элементов массива, модифицируемых в параллельных циклах, на данном процессоре. Для этого необходимо выполнение правила собственных вычислений (см. Глава 2, раздел 2.2.6). Данное правило накладывает ограничения на распределение данных между узлами кластера, для его выполнения необходимо использовать взаимное выравнивание массивов между собой с помощью директивы ALIGN. После распределения массивов с помощью DISTRIBUTE и ALIGN получается дерево выравнивания, которое описывает связи между всеми массивами. Правило собственных вычислений требует, чтобы все массивы, используемые в одном цикле, принадлежали одному дереву выравнивания.

Для реализации был выбран формат графа CSR (Compressed Sparse Rows). Данный формат получил широкое распространение для хранения разреженных матриц и графов. Для неориентированного графа с N вершинами и M ребрами необходимо два массива: X (массив указателей на смежные вершины) и A (массив списка смежных вершин). Массив X имеет размер $N + 1$, а массив $A - 2 * M$, так как в неориентированном графе

для любой пары вершин необходимо хранить прямую и обратную дуги. В массиве X хранятся начало и конец списка соседей, находящиеся в массиве A, то есть весь список соседей вершины J находится в массиве A с индекса $X[J]$ до $X[J+1]$, не включая его.

Каждое измерение массива становится узлом графа, а дуги показывают, как одно измерение массива связано с другим. Для заполнения графа массивов необходимо обработать информацию о вычисленных коэффициентах и связи с циклом. Для этого необходимо пройти по дереву циклов для данного файла в программе и, если цикл содержится в области распараллеливания, добавить необходимые дуги в граф массивов. Для того, чтобы это выполнить, необходимо следующее.

Каждая дуга в графе массивов связывает одно измерение массива A с измерением массива B. На входе в этот алгоритм у нас есть информация о том, как индексные выражения массивов связаны с итерационными переменными цикла. Данная информация содержит только «хорошие» выражения с вычисленными коэффициентами $D * x + E, D > 0$. Дуги добавляются по следующему принципу (W или Write означает обращение на запись, R или Read означает обращение на чтение):

- связываются измерения массивов, обращения по которым присутствуют в левой части операторов присваивания в цикле с типом дуги запись-запись (связь $W - W$) и весом $LW * N$;
- связываются измерения массивов A и B, причем обращение к массиву A содержится в левой части операторов присваивания, а обращение к массиву B содержится в правой части операторов присваивания или в условиях IF, причем не обязательно, чтобы массивы A и B были в одном операторе. Связь создается с типом дуги запись-чтение (связь $W - R$) по данному циклу с весом $LW * N$;
- в случае отсутствия операций записи в массивы связываются измерения массивов, обращения по которым присутствуют в правой части операторов присваивания или условиях IF в данном цикле, причем два обращения к разным массивам не обязаны быть в одном операторе. Связь создается с типом дуги чтение-чтение (связь $R - R$) по данному циклу с весом $LW * N$.

Под N понимается совокупное количество байт, которые потребуется передать другим процессорам в случае неудовлетворения связи с циклом.

В худшем случае необходимо передать целиком все измерение массива, отображенное на соответствующий цикл в случае нарушения обозначенной связи. Количество байт вычисляется из размерности типа используемого массива и номера измерения массива. Например, для такого массива $A(1 : 10, 1 : 20, 1 : 30)$ и типа double precision (sizeof = 8), количество элементов, необходимых для передачи другим процессорам по второму измерению, будет равно $N = 10 * 20 * 8$. Данные о размерах массивов всегда известны системе SAPFOR 2 и должны быть получены либо от статического и/или динамического анализов, либо от пользователя, иначе невозможно построить дерево выравнивания в модели DVMH.

Под LW понимается вес цикла. В зависимости от количества арифметических операторов и обращений к массивам в телах циклов тот или иной цикл с меньшим количеством витков может выполняться дольше аналогичного цикла, но с большим количеством витков. Вес цикла оценивается статическим образом, либо путем динамического профилирования (получение времени выполнения данного цикла). Вес цикла показывает сколько раз цикл был выполнен за все время работы в программе. Например, если цикл выполняется всего один раз (цикл инициализации), то можно пожертвовать количеством коммуникаций в пользу итерационного цикла, который может выполняться сотни, а то и тысячи раз, где каждый лишний переданный байт будет серьезно сказываться на производительности программы в целом. В случае недостаточности информации для оценки веса цикла система SAPFOR 2 полагает $LW = 1.0$, что означает равенство всех циклов в программе.

Дуга в графе связывает два измерения разных массивов и содержит следующую информацию:

- связываемые массивы A и B;
- номер измерения, по которому было обращение по массиву A;
- номер измерения, по которому было обращение по массиву B;
- вес дуги;
- атрибут дуги в виде найденных коэффициентов $D * x + E$ для A и B. Атрибут представляет собой пару из двух пар найденных коэффициентов для соответствующих массивов: $((D_A, E_A), (D_B, E_B))$;
- тип связи ($R - R$ / $W - R$ / $W - W$).

По информации об объявлении массива формируется его уникальное имя следующим образом. Если массив содержится в common-блоке, то его уникальное имя будет следующим: $common\_commonName\_arrayName$, иначе имя массива будет таким: $lineDefinition\_fileDefinition\_arrayName$. Данный способ создания имени позволяет однозначно идентифицировать все массивы в программе (с учетом одинакового именования в разных модулях и функциях). После присвоения имени данный массив регистрируется и добавляется в список всех массивов.

В случае добавления дуги с одинаковыми вершинами происходит увеличение веса данной дуги путем суммирования текущего веса и веса добавляемой дуги. Для каждой области распараллеливания создается свой отдельный граф массивов. Для кода, приведенного в Листинге 5.2, будет построен следующий граф массивов (см. Рисунок 1).

Листинг 5.2 — Фрамент исследуемой программы

```
1  PROGRAM JAC3D
2  PARAMETER (L=20)
3  REAL*4 A(L,L,L),B(L,L,L)
4  DO K = 2, L-1
5    DO J = 2, L-1
6      DO I = 2, L-1
7        B(I,J,K)=( A(I,J,K-1) + A(I,J-1,K) + A(I-1,J,K) +
8                   A(I+1,J,K) + A(I,J+1,K) + A(I,J,K+1)  )/6.
9      ENDDO
10   ENDDO
11 ENDDO
12 END
```

В данном примере размерность массивов задается через $PARAMETER$ в виде константного значения 20. Коэффициенты (D, E) однозначно вычисляются для линейных обращений $D * x + E$ и отображаются на соответствующие циклы. Важно отметить, что порядок обхода циклов не влияет на формирование весов графа для тесно-гнездовых циклов. Для простоты будем считать, что данное гнездо циклов выполняется всего один раз, тем самым на рисунке будет отображен только вес N, равный количеству байт, которое необходимо передать в случае исключения соответствующей связи.

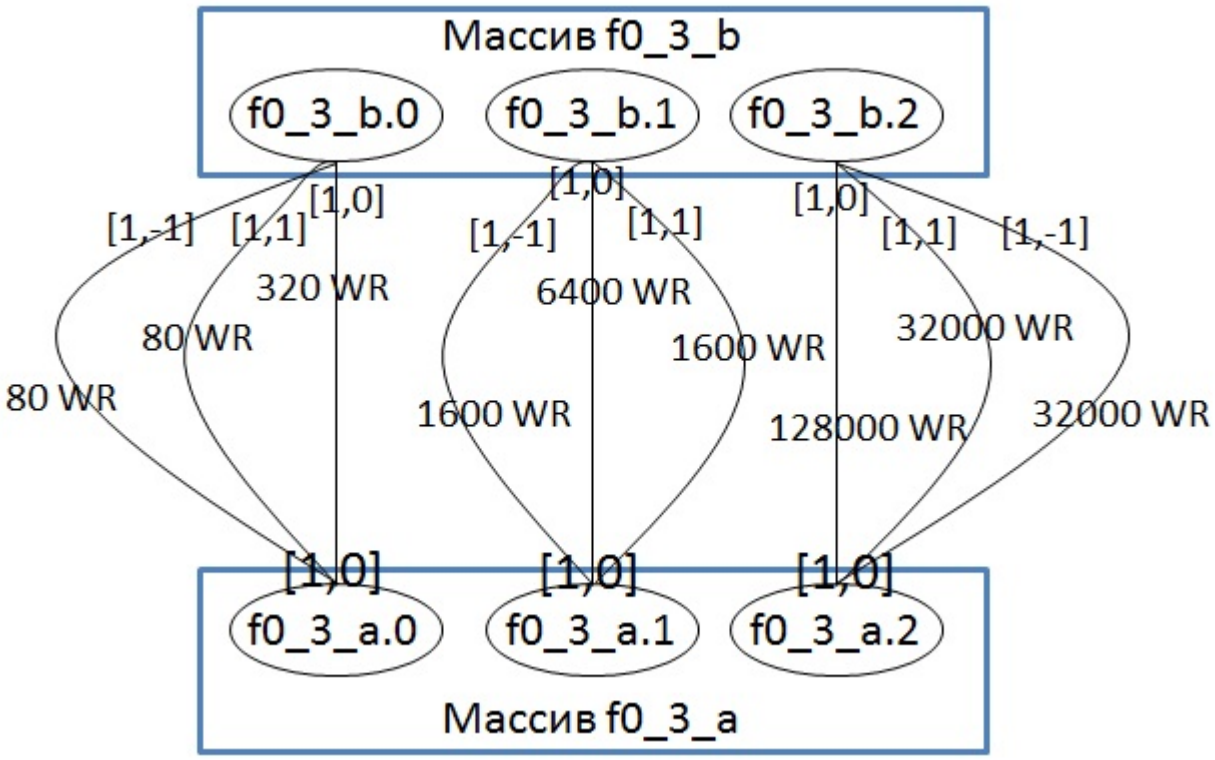

Рисунок 1 — Граф массивов для исследуемой программы до выделения приоритетных дуг

На Рисунке 1 отображены узлы графа для массива A и B до выделения приоритетов. Имена массивов сокращены так, что $f0$ – имя файла, 3 – номер строки в файле, где этот массив объявлен. Массиву $B(L, L, L)$ соответствует три узла в графе с именами $f0\_3\_b.0$, $f0\_3\_b.1$, $f0\_3\_b.2$. Измерения массива нумеруются слева направо, в соответствии с объявлением массива.

Например, для узлов $f0\_3\_b.2$ и $f0\_3\_a.2$ (которые соответствуют последнему измерению массива) в приведенном примере кода есть всего три уникальные дуги с атрибутами (1;-1), (1;0), (1;1), образованные соответствующими обращениями в коде $B(*, *, K) - A(*, *, K-1)$, $B(*,*,K) - A(*,*,K)$, $B(*,*,K) - A(*,*,K+1)$. В случае «потери» данной связи дуги (1;0)-(1;0) ее вес будет равен размеру всего массива, умноженному на размерность типа и на количество таких дуг в цикле: $N = 20^3 * 4 * 4$ или 128000 байт. Аналогично получены и остальные веса.

Для того, чтобы отличать дуги по типу связи $(W-W, W-R, R-R)$, расставим следующие приоритеты. Тип дуги с типом связи $W - W$ имеет

самый высший приоритет, однако все дуги с типом $W - W$ имеют равный приоритет между собой. Это объясняется тем, что неудовлетворение данной дуги приведет к невозможности распараллеливания данного цикла, так как не будет выполнено правило собственных вычислений для связываемых массивов, что в итоге повлечет за собой большие коммуникации (так как все обращения к распределенным массивам на чтение в таком цикле в худшем случае должны быть «покрыты» с помощью доступа к удаленным данным). Дуги с типом связи $W - R$ имеют приоритет над дугами с типом связи $R - R$. Дуги каждого из типов $W - R$ и $R - R$ имеют также равный приоритет между собой.

Чтобы обеспечить приоритет дуг, был реализован следующий алгоритм выделения дуг по приоритетам. Сначала посчитаем общую сумму всех дуг с типом $R - R$, $S1 = \sum_{[R-R]}$. Затем прибавим число $S1$ к дугам с типом $W - R$. Тем самым мы выделим приоритет $W - R$ типа над дугами типа $R - R$, сохранив между тем равный приоритет между схожим типом. Затем вычислим общую сумму весов всех дуг с типами связи $W - R$ и $R - R$, $S2 = \sum_{[W-R,R-R]}$ и добавим теперь число $S2$ ко всем дугам с типом $W - W$. Таким образом, будет выполнено правило приоритета дуг: вес любой дуги с типом $W - W$ будет больше, чем $W - R$ и $R - R$, вес любой дуги с типом $W - R$ будет больше, чем $R - R$.

5.2.3 Определение распределяемых массивов

По умолчанию система SAPFOR 2 считает все массивы в программе распределяемыми. Распределяемый массив – это такой массив, для которого необходимо построить распределение данных с помощью DVMH-директив и выполнить соответствующее выбранному распределению отображение вычислений в параллельных циклах и одиночных операторах. Но не все массивы надо распределять, и задача системы SAPFOR 2 распознать такие ситуации либо в автоматическом, либо в полуавтоматическом режиме. Рассмотрим, какие массивы не требуют распределения.

К первой категории массивов, которые не требуется распределять, относятся те массивы, которые указаны в спецификациях приватизации (!$SPF PRIVATE) и редукции (!$SPF REDUCTION). Данные массивы являются вспомогательными в местах использования (в основном в циклах). К тому же использовать в циклах распределенный массив как приватизируемый или редукционный запрещено DVM-системой. На данный момент приватизацию и редукцию по массивам для циклов статический анализ системы SAPFOR 2 определить не способен. Такие указания могут быть вставлены в программу пользователем и в дальнейшем учтены при определении класса массива (распределяемый или не распределяемый).

Ко второй категории относятся массивы, которые участвуют в операторах ввода/вывода, а также массивы, которые передаются как параметры во внешние процедуры. В операторах ввода/вывода разрешается задавать только по одному массиву и только целиком из-за ограничений DVM-системы. Все остальные случаи отменяют распределение данного массива. Для того, чтобы не отменять распределение данных массивов, которые участвуют в сложных операторах ввода/вывода, а также во внешних процедурах, требуется заводить массивы-копии и вставлять копирования до и после соответствующих операторов. На данном этапе такое преобразование не выполняется автоматически системой SAPFOR 2.

К третьей категории относятся массивы, которые система SAPFOR 2 автоматически отфильтровывает по тем или иным причинам. Для выполнения фильтрации необходимо построение дерева циклов и графа вызовов функций, а также анализ обращений к массивам и связи данных обращений с циклами, описанный в подразделах 5.2.1 и 5.2.2.

Процесс фильтрации состоит в том, чтобы запретить использовать распределяемые массивы, которые в дальнейшем будут отображены на разные деревья выравнивания и соответственно на разные шаблоны DVM в потенциально параллельных циклах. Разные деревья выравнивания образуются из несвязанных между собой графов массивов. Также отфильтровываются массивы, объявленные в common-блоках, которые могли быть использованы в процедурах, вызываемых из цикла, но не были связаны с этим циклом.

После выполнения фильтрации происходит распространение состояния распределяемости массивов в соответствии со связями, которые были

построены в результате вызовов процедур в программе. И в заключение весь анализ, описанный в подразделах 5.2.1 и 5.2.2, выполняется заново с уже отфильтрованным списком массивов. Для всех массивов, которые не могут быть распределены по тем или иным причинам, выводятся соответствующие диагностические сообщения.

5.2.4 Поиск оптимального связывания массивов

На втором этапе, когда уже все файлы программы обработаны, необходимо для каждой области распараллеливания и для каждого графа массивов в ней запустить алгоритм создания усеченного графа массивов (такого графа массивов, который не содержит циклов, порождающие конфликтные ситуации) для последующего создания распределения данных.

Если для какой-то области распараллеливания не удалось найти массивы для распределения, то выдается соответствующая диагностика об ошибке и область исключается из анализа. Если не осталось активных областей распараллеливания, то анализ останавливается.

Помимо выделения типов дуг по приоритетам, необходимо удалить кратные дуги, возникающие между двумя парами вершин. На Рисунке 1 кратные дуги возникают из-за разных обращений к массивам: $B(*,*,K) - A(*,*,K-1)$, $B(*,*,K) - A(*,*,K)$, $B(*,*,K) - A(*,*,K+1)$. Удаление кратных дуг выполняется с учетом максимизации веса, что позволит выбрать наиболее значимую связь между данными массивами в цикле и попытаться уменьшить коммуникационные затраты.

Переборный алгоритм поиска наиболее важных дуг в графе

Алгоритм поиска оптимального выравнивания также состоит из двух этапов. Первый этап – получение всех простых циклов в графе массивов. Простой цикл в графе – это замкнутый цикл без повторного прохода по

ребру или посещения вершины дважды, за исключением начальной и конечной вершин. В системе SAPFOR 2 цикл представляет собой набор дуг графа массивов с сохранением веса и атрибутов.

Нахождение всех простых циклов в графе массивов является NP-трудной задачей, поэтому для ограничения поиска по времени и ресурсам в случае больших графов вводится два параметра, которые позволят ограничить этот поиск, – максимальный размер цикла (размером цикла будем называть количество входящих в него дуг), который необходимо добавить в список простых циклов, а также – максимальная длина просматриваемой цепочки при рекурсивном поиске таких циклов в графе.

Первый параметр используется для ограничения по памяти, второй – по времени поиска всех простых циклов. Можно отметить, что накладываемые ограничения не всегда позволяют найти все простые циклы в графе массивов, а это значит, что алгоритм не всегда может найти точное решение. С другой стороны – чем больше размер (длина) цикла, тем больше массивов он связывает между собой. Тем самым для получения оптимального выравнивания массивов между собой не обязательно находить все простые циклы в графе.

Второй этап – обработка найденных простых циклов. После нахождения всех простых циклов (далее просто циклов), происходит их сортировка по размеру (длине), а также разделение на независимые группы по длине. Затем для каждого цикла выполняется сортировка его дуг по весу, и все циклы в каждой группе упорядочиваются по суммарному весу. Данные сортировки позволят наиболее быстрым образом обеспечить поиск и удаление конфликтных дуг в графе массивов.

Далее необходимо выполнить поиск конфликтных циклов. Конфликты могут быть двух типов. Конфликтом в графе массивов будем называть следующие ситуации:

- наличие цикла, для вершин которого нельзя построить единственный вариант выравнивания измерений массивов (где каждая вершина соответствует измерению массива). Такие конфликты будем называть конфликтами первого типа (1);
- присутствует явная или косвенная (через другие дуги графа) дуга между двумя измерениями одного и того же массива. Такие конфликты будем называть конфликтами второго типа (2).

Для устранения конфликта (1) можно удалить любую из дуг цикла, например, с минимальным весом. Для устранения конфликта (2) необходимо удалить такую дугу, чтобы явная или косвенная связь между двумя измерениями массивов, которая выражена дугами графа, пропала. Если есть конфликтные циклы, то запускается алгоритм устранения конфликтов. Рассмотрим такой алгоритм, который позволяет удалять конфликтные и неконфликтные дуги. После применения данного алгоритма мы получим усеченный граф массивов, который не содержит циклов.

Каждый цикл характеризуется суммарным весом всех его дуг или просто общим весом. Ранее все циклы были отсортированы с учетом их веса и размерности от самых маленьких до самых больших циклов по размерности (длине), а циклы с одинаковой размерностью были отсортированы по убыванию их веса.

Для устранения конфликтов запускается рекурсивная процедура. Цель данной процедуры – удалить дуги с минимальным суммарным общим весом, что позволит получить наиболее оптимальное с точки зрения обменов между узлами распределение данных на основе построения графа массивов и задания соответствующих весов. В начале процедуры имеем нулевой общий вес удаленных дуг и пустой список удаляемых дуг. Выбираем очередной цикл из списка полученных конфликтных циклов. Пытаемся по очереди удалить каждую из дуг данного цикла и смотрим, что получается:

- если это первая дуга цикла, то удаляем эту дугу, прибавляем вес этой дуги к общему весу всех удаленных дуг и заносим эту дугу в список удаляемых. После удаления данной дуги некоторые циклы из полученного списка перестанут быть циклами и, соответственно, не будут принимать участие в дальнейшем выборе. Далее рекурсивно вызываем эту процедуру. Рекурсивный вызов завершается в том случае, если нет больше циклов с конфликтами. В этом случае получен суммарный вес и список необходимых для удаления дуг;
- если эта дуга цикла не первая, то у нас уже имеется общий вес и список удаляемых дуг на каком-то конкретном уровне рекурсивной вложенности вызова рассматриваемой процедуры. Также мы имеем текущий суммарный вес на текущем уровне рекурсивной вложенности. Если сумма веса очередной удаляемой дуги и текущего общего суммарного веса больше, чем найденный наименьший общий вес

удаляемых дуг, то рекурсивный вызов удаления этой дуги не выполняется, так как удаление этой дуги повлечет за собой увеличение общего веса всех удаляемых дуг (таким образом выполняется отсечение перебора всех вариантов наборов дуг для удаления). Если мы завершили рекурсивный вызов и получили меньший вес, чем был найден до этого, то корректируется общий вес и соответствующий список дуг для удаления.

Физически дуги не удаляются, на следующую глубину рекурсивного вызова передается локальный список дуг для удаления. В результате выполнения такой процедуры мы получим совокупность дуг для разрешения конфликтов с минимальным суммарным весом. Остальные циклы устраняются произвольным образом с учетом минимизации суммарного веса, так как такие циклы не влияют на оптимальность распределения данных.

После того, как мы получили список дуг для удаления, происходит формирование усеченного графа массивов (исходный граф не портится), такого графа, который не содержит циклов. Затем необходимо проверить, не возникнут ли конфликты второго типа в усеченном графе, или нет ли таких путей в графе, которые косвенно (через другие массивы и соответствующие им дуги) связывают измерения одного и того же массива. Данная конфликтная ситуация не образует цикл, но требует разрешения.

Для этого необходимо для каждого массива для всех уникальных комбинаций пар его измерений добавить фиктивную дугу между этими измерениями с очень большим весом (например, большим, чем сумма всех весов в графе) и повторить весь алгоритм поиска простых циклов и удаления конфликтов. Если мы нашли конфликтный цикл, то мы удаляем дугу. Из-за особенности алгоритма (применение сортировок и удаление дуг с минимальным совокупным весом) будет выбрана не фиктивная дуга, а существующая дуга в графе. Таким образом, будут разрешены все конфликты второго типа.

Стоит отметить, что после такой операции по данному графу массивов не всегда можно построить выравнивание всех массивов между собой, особенно если в графе было много конфликтов или были применены ограничения на поиск простых циклов, что является минусом данного алгоритма. Оценка сложности данного алгоритма является экспоненциальной в зависимости от количества вершин в графе, что накладывает ограничения на

его применимость на больших программах. Данный алгоритм может быть использован на сравнительно небольших программах, где количество узлов графа массивов не более 5.

Использование алгоритма максимального остовного дерева для поиска наиболее важных дуг в графе

Альтернативным алгоритмом поиска наиболее важных дуг является модифицированный алгоритм поиска минимального остовного дерева [15; 18]. Минимальное остовное дерево – такое дерево, которое является максимальным по включению ребер подграфом, не имеющее циклов, и в котором сумма весов ребер минимальна. Если исходный граф связный, то будет построено остовное дерево, если же в исходном графе несколько несвязных компонент, то результатом будет остовный лес.

В нашей задаче требуется найти набор дуг с наибольшим весом. Таким образом без изменения общности алгоритма поиска минимального остовного дерева можно искать максимальное остовное дерево – такое дерево, в котором сумма весов ребер максимальна. Была использована самая простая реализация – алгоритм Дейкстры-Прима. Недостатком данного алгоритма является тот факт, что решение получается не самое лучшее, как в случае полного перебора, так как не выполняется перебор всех цепочек дуг и их сравнение между собой. Достоинство же такого подхода заключается в линейной оценке сложности альтернативного алгоритма в зависимости от количества вершин в графе. Данное достоинство вместе с возможностью распараллелить данный алгоритм делает возможным его использование на любой программе с любым количеством массивов.

5.2.5 Создание вариантов распределения данных

После того, как был получен оптимальный граф, который связывает измерения массивов в соответствии с их использованием в циклах программы, можно получить варианты распределения данных. Для каждой области распараллеливания создаются свои варианты распределения данных по усеченному графу массивов (далее, граф массивов).

Затем в графе массивов ищутся все деревья, которые связывают массивы друг с другом (граф не обязательно является связным). После такого поиска мы знаем, сколько подграфов (деревьев) у нас есть и какие массивы в эти подграфы входят. Для каждого дерева создается свой шаблон в модели DVMH (DVMH TEMPLATE), который и будет распределяться с помощью директивы DISTRIBUTE и на который будут выровнены все массивы данного дерева. Шаблон представляет собой виртуальный массив, под который не отводится память в программе.

Шаблон создается по массиву с наибольшей размерностью, а среди одинаковых массивов одной размерности, выбирается тот, который занимает больше памяти. После того, как создан шаблон и найден массив, по которому строится этот шаблон, в граф добавляются дуги, связывающие измерения этого массива и измерения этого шаблона с атрибутами (1,0), весом 1.0 и типом связи $R - R$. Таким образом, в графе массивов появляется шаблон, до которого можно добраться по связям между массивами и узнать выравнивание на него.

Так как с шаблоном связан только один из массивов, необходимо уметь вычислять связь с шаблоном и для других массивов. Наибольшая сложность, которая может возникнуть при вычислении таких связей – не кратные коэффициенты при выравнивании на шаблон. Такие коэффициенты могут возникать в том случае, когда измерение одного массива требуется распределить, например, с развижкой $3 * i$, а измерение второго массива требуется распределить на то же измерение шаблона с развижкой $2 * i$. Таким образом, требуется вычислить наименьшее общее кратное и изменить соответствующие атрибуты в графе. Для поиска таких связей был реализован итерационный алгоритм, который постепенно увеличивал атрибут (N, 0) у дуги, которая соединяет измерение шаблона и измерение

массива до тех пор, пока не будут найдены корректные коэффициенты выравнивания на шаблон для всех массивов.

Варианты распределения данных создаются для каждого шаблона независимо, количество вариантов для каждого шаблона будет равно $2^{DIM(T_i)}$, а общее их количество – $\sum_{i=1}^{N} 2^{DIM(T_i)}$, где $DIM(T_i)$ – размерность шаблона (каждое измерение считается распределенным, либо размноженным), N – количество построенных шаблонов.

Модуль визуализации на текущий момент поддерживает два режима работы с вариантами распределения данных. Первый режим – создаются все возможные комбинации вариантов распределения данных, второй режим – пользователь сам выбирает какие измерения шаблона сделать распределенными, а какие – размножить, а также сколько версий параллельных схем создать. При использовании первого варианта, а также при применении алгоритма грубой оценки каждой полученной схемы, модуль визуализации позволяет пользователю отсортировать все полученные параллельные схемы по их прогнозируемой эффективности.

5.2.6 Создание директив выравнивания данных

Правила выравнивания массивов на шаблон в модели DVMH не зависят от выбранного в дальнейшем распределения этих шаблонов. Для каждого массива из подграфа, который не является шаблоном, выполняется поиск связи его измерений с шаблоном в этом подграфе. Поиск связи для конкретного измерения массива запускается в том случае, если это измерение присутствует в графе массивов. В результате поиска может быть не найдено связи с шаблоном по конкретному измерению. Данное измерение будет размножено (указана * в спецификации ALIGN). Если что-то найдено, проверяется, корректно ли найденное правило, и если не корректно – выдается ошибка и весь анализ прекращается, иначе – формируется связь с шаблоном.

В результате, после поиска будет получен массив, который надо выровнить, и правила выравнивания для каждого измерения массива на

шаблон. По этим правилам строится правило выравнивания, которое содержит в себе всю информацию о выравнивании двух массивов:

- массив, который выравнивается;
- массив, на который происходит выравнивание;
- правила связи левой части директивы выравнивания (ALIGN);
- правила связи правой части директивы выравнивание (ON).

По этим правилам выравнивания расширяется размер шаблона по каждому измерению, если правила выравнивания выводят за пределы текущего размера измерения шаблона. После того, как все правила были созданы без ошибок, есть возможность получить все директивы распределения и выравнивания данных для конкретного варианта распределения шаблона (шаблонов).

5.3 Построение распределения вычислений и доступа к удаленным данным

5.3.1 Создание директив распределения вычислений

Распределение вычислений происходит отдельно для каждого из вариантов распределения данных. В зависимости от того или иного варианта распределения данных распределение вычислений и организация доступа к удаленным данным могут быть совершенно разными. Так, например, в случае размножения всех измерений распределяемых шаблонов (стоит * во всех измерениях шаблонов в директивах DISTRIBUTE), нет необходимости организовывать теневые обмены между узлами кластера, так как все необходимые данные и так будут находиться на каждом узле в таком варианте программы.

Алгоритм распределения вычислений выделен в отдельный проход. Данный проход оперирует полученными данными после прохода распределения данных. Проход распределения вычислений, который состоит из двух этапов, использует ту же функцию для анализа операторов файла

программы, что и проход распределения данных, но с некоторыми отличиями. На первом этапе независимо создаются директивы распределения вычислений для каждого цикла из дерева циклов. На втором этапе происходит объединение созданных директив при каждом из циклов по некоторым правилам.

Первый этап создания директив распределения вычислений

Вся функциональность, описанная для сбора информации по операторам (см. Раздел 5.2 Главы 5), повторяется согласно алгоритму поиска распределения данных. Рассмотрим алгоритм создания директив распределения вычислений.

Выполняется проход по всем циклам, для каждого цикла проверяется, содержится ли он в какой-либо области распараллеливания, если нет – цикл не рассматривается и пропускается. После анализа операторов создается информация о записях и чтениях по массивам в этом цикле. На основе этой информации запускается процедура проверки возможности на распараллеливаемость данного цикла. Цикл не подлежит распараллеливанию, если:

– итерационная переменная цикла присутствует в индексных выражениях в нескольких измерениях массива, который используется в левой части присваивания (см. Листинг 5.3);

Листинг 5.3 — Пример цикла с конфликтом

```
1  DO K = 2, L-1
2     DO J = 2, L-1
3        A(J,J,K)= ...
4     ENDDO
5  ENDDO
```

– есть несколько различных обращений к массиву, индексные выражения которого принадлежат одному и тому же измерению массива и содержат одну и ту же итерационную переменную цикла в левой части присваивания, и этот массив не присутствует в директиве

!$SPF ANALYSIS(ACROSS(...)) (см. Листинг 5.4). В данном случае будет нарушено правило собственных вычислений в терминах DVMH-модели: каждый процессор должен выполнять присваивания только в свои элементы;

Листинг 5.4 — Пример цикла с конфликтом

```
1  DO K = 2, L-1
2    DO J = 2, L-1
3      DO I = 2, L-1
4        A(I,J,K)= ...
5        A(I + 1,J,K)= ...
6      ENDDO
7    ENDDO
8  ENDDO
```

– есть зависимость между чтением и записью по массиву и эта зависимость не объявлена в директиве !$SPF ANALYSIS (ACROSS(...)).

Если описанные выше проверки не пройдены, то директива распределения вычислений для этого цикла не создается. Иначе выполняется дальнейшая обработка данного цикла и попытка создать директиву.

Для создания директивы распределения вычислений необходимо найти массив из графа массивов, на который будет отображаться пространство витков данного цикла. Если в рассматриваемом цикле присутствует запись всего в один массив – он и будет выбран. Если массивов на запись несколько, то выбор происходит, прежде всего, среди массивов, которые участвуют в спецификации ACROSS, если таковая имеется. В модели DVMH требуется, чтобы цикл был отображен на массив, который участвует в спецификации ACROSS.

В результате алгоритма поиска оптимального выравнивания данных, который был описан выше, был получен усеченный граф массивов. Конфликтные ситуации первого и второго типов были разрешены с помощью удаления соответствующих ребер этого графа. В результате чего интересы тех или иных циклов могли быть не учтены, что может привести к тому, что нельзя создать директиву распределения вычислений для этого цикла. Для этого необходимо выполнить следующие проверки.

Выполняется обработка всех записей в массивы, на которые необходимо отобразить вычисления в данном цикле, и проверяются все комбинации

обращений массивов на запись в данном цикле. Критерий проверки – все комбинации пар обращений должны удовлетворять выбранному взаимному выравниванию, то есть иметь одинаковое выравнивание. Такое требование вызвано правилом собственных вычислений, которое было принято в модели DVMH (см. Глава 2, раздел 2.2.6). Для такой проверки берутся коэффициенты из обращения у двух массивов (D_A, E_A) и (D_B, E_B) и сравниваются с коэффициентами, которые задают правила выравнивания этих двух массивов между собой в соответствии с графом массивов. Если проверки не пройдены, то регистрируется конфликтная запись по этому циклу (что свидетельствует о нарушении правила собственных вычислений), и директива не создается для этого цикла. Иначе выбирается массив с наименьшей размерностью. Если в цикле отсутствуют записи в массивы, то выполняется поиск чтений по массивам, на которые будет сделано отображение вычислений.

В результате будет получен массив, на который будут отображаться вычисления рассматриваемого цикла. Если данный массив не содержится в спецификации ACROSS, то выбирается соответствующий этому массиву шаблон для отображения цикла, иначе используется найденный массив.

Если цикл не содержит ограничений на распараллеливание (считается потенциально параллельным), то запускается процедура создания директивы распределения вычислений. Если директива создана, то она привязывается к соответствующему циклу в дереве циклов. Для директивы заполняется информация о массиве, на который требуется отобразить вычисления, правила отображения, а также считывается информация из SPF директив перед данным циклом (приватные переменные, редукция, SHADOW, ACROSS, REMOTE_ACCESS). Также заполняется некоторая информация для дальнейшего вычисления SHADOW-граней (теневых-граней) и ACROSS-граней.

В результате выполнения данного этапа для всех потенциально параллельных циклов либо будет создана директива распределения вычислений, либо она будет отсутствовать по той или иной причине.

Второй этап создания директив распределения вычислений

Данный этап состоит в объединении полученных директив для каждого из циклов, если имеет место тесная вложенность. Если цикл не тесно-вложенный, то запускается преобразование, которое позволяет на основе информации, полученной от статического и динамического анализов, произвести объединение циклов и сделать их тесно-вложенными (внесение инварианта цикла). Данное преобразование выполняется на лету и позволяет устранить нетесную вложенность, если это возможно, без потери результатов анализа и построенных структур.

Для объединения результатов первого этапа необходимо выполнить проход по всем циклам и для тех пар циклов, которые являются тесно-вложенными, выполнить объединение директивы распределения вычисления от самого вложенного цикла до самого верхнего по следующему правилу: самый вложенный цикл «передает» свои данные его родителю, затем следующий цикл по вложенности «передает» уже частично объединенную информацию также своему родителю. Объединение информации происходит только в случае ее наличия.

В итоге, самый верхний цикл будет содержать объединенную информацию от всех тесно-вложенных циклов, следующий по уровню вложенности цикл будет содержать объединенную информацию от всех тесно-вложенных циклов, которые расположены ниже по уровню вложенности и т.д. Такой подход позволяет выбирать разные циклы для распараллеливания программы в зависимости от выбранного варианта распределения данных.

Выбор директив распределения данных для вставки в код программы

В зависимости от выбранного варианта распределения данных (шаблонов) будут выбраны соответствующие директивы распределения вычислений, а также созданы дополнительные директивы перераспределения данных и доступа к удаленным данным, если это потребуется.

Данный алгоритм также реализован в виде отдельного прохода, который по очереди обрабатывает все файлы проекта. Для каждого цикла в файле запускается рекурсивная процедура, которая выбирает, какую директиву необходимо будет вставить в исходный код программы. Так как при каждом цикле имеется агрегированная информация по всем тесно-вложенным циклам, которые расположены ниже рассматриваемого, остается только выбрать необходимый цикл. Приоритет отдается самому внешнему циклу, для которого выполнены следующие критерии:

- у цикла присутствует директива распределения вычислений;
- у цикла нет ограничений на параллельное выполнение (потенциально параллельный цикл);
- цикл находится в области распараллеливания.

Если текущий цикл не удовлетворяет выше описанным критериям, выполняется рекурсивная обработка его потомков. Если найден цикл, который удовлетворяет условиям, выполняются проверка тесной вложенности его потомков:

- если потомки для выбранного цикла не являются тесно-вложенными с выбранным циклом, то добавляются директивы перераспределения данных до этого цикла и после завершения этого цикла. Правила перераспределения описаны ниже;
- если потомки для выбранного цикла являются тесно-вложенными с выбранным циклом, то выполняется проверка соответствия выбранного распределения данных с выбранным распределением вычислений (имеются ли распределенные измерения массивов, на которые нельзя отобразить вычисления в данном цикле). Если есть такие массивы, то для них выполняется проверка – распределены ли те измерения массива, на которые не будет выполнено

отображение вычислений в выбранной директиве распределения вычислений, и если да – добавляем директивы перераспределения до этого цикла и после выполнения этого цикла по описанным ниже правилам.

Данные проверки позволяют корректно расставить директивы распределения вычислений в соответствии с выбранным вариантом распределения данных с учетом требований модели DVMH. Перераспределение данных может возникать только для согласованного параллельного выполнения рассматриваемого цикла с учетом варианта распределения данных.

Правила вставки директив перераспределения данных

В текущей версии системы SAPFOR 2 перераспределение данных осуществляется только для шаблонов (перераспределяется все дерево выравнивания). Перераспределение данных делается для того, чтобы сохранить корректность выполнения данного цикла. Перераспределение до цикла приводит к размножению тех измерений массивов, которые являются распределенными и на которые нельзя отобразить вычисления в данном цикле. Перераспределение после цикла приводит к возвращению состояния распределения массивов, которое было до выполнения цикла.

Для того, чтобы осуществить перераспределение данных, необходимо найти те измерения шаблонов, которые являются распределенными и на которые нет отображения вычислений в выбранной директиве распределения вычислений. Все найденные распределенные измерения становятся размноженными до цикла, а после цикла все размноженные измерения становятся распределенными.

Таким образом, перераспределение данных позволяет выполнить цикл параллельно (для которого была найдена директива распределения вычислений), даже если распределение вычислений задано не для каждого распределенного измерения шаблона. Если для двух подряд идущих циклов необходимо одинаковое перераспределение данных, будут сгенерированы следующие директивы перераспределения данных (см. Листинг 5.5).

Перераспределение данных является сложной операцией, которая вызывает большое количество пересылок между узлами. Для того, чтобы не выполнять лишние действия по пересылке данных между узлами, была реализована оптимизация: две подряд идущие директивы перераспределения данных заменяются последней из них (система поддержки DVMH не будет выполнять перераспределение данных, если распределение массива уже находится в нужном состоянии).

Листинг 5.5 — Пример расстановки директив перераспределения данных

```
1  !DVM$ DISTRIBUTE A(BLOCK, BLOCK)
2  ....
3  !DVM$ REDISTRIBUTE A(BLOCK, *)
4  !DVM$ PARALLEL (K) on A(K, *)
5  DO K = 2, L-1
6    A(K, 1) = 5
7  ENDDO
8  !DVM$ REDISTRIBUTE A(BLOCK, BLOCK) ! Директва будет удалена
9
10 !DVM$ REDISTRIBUTE A(BLOCK, *)
11 !DVM$ PARALLEL (K) on A(K, *)
12 DO K = 2, L-1
13   A(K, 2) = 7
14 ENDDO
15 !DVM$ REDISTRIBUTE A(BLOCK, BLOCK)
```

5.3.2 Организация доступа к удаленным данным

Для корректной работы параллельных циклов требуется расстановка директив, которые организуют доступ к удаленным данным. Удаленные данные могут быть следующих типов:

- редукционные данные (REDUCTION). Данные такого типа используют редукционные операции, в которых участвуют все процессы (операция является коллективной);
- SHADOW-грани или теневые грани. Данные такого типа необходимы для доступа на чтение данных, расположенных на соседних

процессорах. Обращение к таким данным должно удовлетворять правилам, введенным в модели DVMH;

- ACROSS-грани. Данные такого типа необходимы для организации конвейерного режима выполнения циклов с регулярными зависимостями по данным;
- удаленные данные (REMOTE_ACCESS). Данные такого типа необходимы для доступа на чтение любых данных, расположенных на любом процессоре. Доступ к таким данным часто организуется в том случае, если нельзя использовать все предыдущие возможности.

В системе SAPFOR 2 поддерживаются редукционные операции по скаляру и массиву со следующими типами: сумма, произведение, максимум, минимум, MINLOC, MAXLOC, логические операции с типами AND, OR, EQV, NEQV над скалярными переменными типа BOOLEAN.

Для определения SHADOW-граней и ACROSS-граней используется один и тот же алгоритм, так как ACROSS-грани фактически являются расширением понятия SHADOW-граней. Для организации удаленного доступа типа ACROSS и SHADOW используется информация об отображении витков цикла на массив (шаблон) в директиве распределения вычислений, правила выравнивания массивов между собой, а также правила выравнивания массивов на шаблон. Если измерение массива, на которое отображаются витки параллельного цикла, является размноженным, то все удаленные ссылки аннулируются в качестве оптимизации.

Удаленные ссылки типа REMOTE_ACCESS могут быть вставлены как в DVMH-директиву параллельного цикла, так и для отдельно стоящего оператора. Последняя возможность необходима в случае доступа к элементам распределенных массивов вне параллельных циклов на чтение. В случае использования доступа к распределенным массивам вне параллельных циклов на запись компилятор DVMH сам вставляет проверки на то, что данное присваивание происходит на том процессоре, на котором находятся эти данные.

Удаленные ссылки типа REMOTE_ACCESS добавляются к DVMH-директиве параллельного цикла в том случае, если обращение к массиву на чтение не удовлетворяет правилам регулярного доступа $D * x + E$ хотя

бы по одному его измерению, либо обращение массива на чтение не соответствует правилам выбранного варианта распределения данных и правилам выравнивания массивов между собой.

Удаленные ссылки типа REMOTE_ACCESS добавляются для оператора, находящегося в области распараллеливания, если имеет место чтение из распределенного массива. Перед этим оператором необходимо вставить директиву REMOTE_ACCESS, для того чтобы все процессоры смогли прочитать информацию о текущем элементе массива.

Если рассматриваемый оператор является оператором присваивания, и левая и правая части этого оператора содержат одинаковые обращения к массивам (например, $A(i,j,k) = A(i,j,k) + 5$), то вставка директивы REMOTE_ACCESS не требуется, так как для данного оператора будет выполнено правило собственных вычислений – все обрабатываемые элементы на запись и чтение находятся на одном процессоре.

5.4 Расстановка вычислительных регионов и директив актуализации данных

В DVM-системе применяется двухуровневая балансировка распределения данных и вычислений между всеми устройствами. Первый уровень – распределение данных между узлами кластера. Второй уровень – распределение данных и вычислений между устройствами внутри узла, такими как многоядерные процессоры, ускорители разной архитектуры (например, ГПУ). Для того, чтобы задействовать устройства внутри узла, необходимо превратить DVM-программу в DVMH-программу путем вставки вычислительных регионов и директив актуализации данных между устройствами.

Выполнение DVMH-программы можно представить как выполнение последовательности вычислительных регионов и участков между ними, которые будем называть внерегионным пространством. Код во внерегионном пространстве выполняется на центральном процессоре, тогда как вычислительные регионы могут выполняться на разнородных вычислительных устройствах. Как внутри, так и вне регионов могут быть как параллельные циклы, так и последовательные участки программы.

Вычислительный регион выделяет часть программы (с одним входом и одним выходом) для возможного выполнения на одном или нескольких вычислительных устройствах. Регион задается парой директив, отмечающих начало и конец региона (!DVM$ REGION и !DVM$ END REGION). Содержимое региона – часть программы, содержащая произвольное количество параллельных циклов, возможно, разделенных последовательными группами операторов. Регион может быть пустым. DVMH-массивы распределяются между вычислителями, нераспределенные данные размножаются. Витки параллельных DVMH-циклов внутри региона делятся между вычислителями в соответствии с правилом отображения параллельного цикла, заданного в директиве параллельного цикла.

В SAPFOR 2 минимально возможной областью кода, которая может быть включена в регион, является параллельный цикл. Для того, чтобы параллельный цикл можно было включить в вычислительный регион, необходимо и достаточно проверить наличие нечистых процедур, которые вызываются из данного цикла. Нечистая процедура – такая процедура, которая содержит побочные эффекты, не объявлена ключевым словом PURE, не содержит спецификации INTENT, а также использует переменные из common-блоков на запись, использует операторы ввода/вывода.

На первом этапе каждый параллельный цикл, который проходит критерии на возможность включения в регион, окружается директивами начала и конца региона. Далее выполняется объединение регионов, если между концом текущего региона и началом следующего нет никаких операторов.

Вне вычислительных регионов управление перемещением данных между оперативной памятью ЦПУ и памятями ускорителей задается при помощи директив актуализации – !DVM$ GET_ACTUAL и !DVM$ ACTUAL. Отметим особенности реализации данных директив в DVM-системе.

Директива !DVM$ GET_ACTUAL делает все необходимые обновления для того, чтобы в памяти ЦПУ были актуальные (т.е. самые новые) значения данных в указанных в списке подмассивах и скалярах. В случае отсутствия у директивы параметров все данные в памяти ЦПУ становятся актуальными.

Директива !DVM$ ACTUAL объявляет тот факт, что указанные в списке подмассивы и скаляры имеют самые новые значения в памяти ЦПУ. Значения указанных переменных и элементов массивов, находящиеся в памяти ускорителей, считаются устаревшими и перед использованием будут при необходимости обновлены. В случае отсутствия параметров все данные считаются актуальными только в памяти ЦПУ. Параметры директив – список переменных, массивов и подмассивов – указываются в круглых скобках через запятую.

Данная реализация в компиляторе DVMH уже содержит в себе некоторые оптимизации по перемещению данных. Так, например, в следующем коде (см. Листинг 5.6), если в момент выполнения директивы в строке 3 актуальные данные находились на ГПУ, то будет выполнено копирование данных массива A из памяти ГПУ в память ЦПУ. Далее выполнение директивы в строке 5 приведет к изменению статуса актуальности массива A – актуальные данные находятся на ЦПУ. Следующая директива в строке 7 не будет выполнять копирование данных, так как актуальные данные и так находятся на ЦПУ. И соответственно директива в строке 9 опять обновит статус актуальности массива A.

Листинг 5.6 — Использование директив перемещения данных

```
1 !DVM$ DISTRIBUTE A(BLOCK, BLOCK)
2 ....
3 !DVM$ GET_ACTUAL(A)
4 ... ! внерегионное пространство
5 !DVM$ ACTUAL(A)
6 ... ! внерегионное пространство
7 !DVM$ GET_ACTUAL(A)
8 ... ! внерегионное пространство
9 !DVM$ ACTUAL(A)
```

Таким образом, если внерегионное пространство содержит в себе вызовы процедур и/или выполнение каких-либо регионов, то это будет учтено непосредственно там, где стоит директива актуализации. Поэтому для расстановки данных директив достаточно проанализировать внерегионное пространство и поставить директиву актуализации данных GET_ACTUAL перед использованием, а также поставить директиву актуализации данных ACTUAL после изменения распределенного массива в параллельных циклах и одиночных операторах вне регионов.

В качестве оптимизации выполняется объединение директивы актуализации данных для составных операторов (например, цикл WHILE, оператор IF, SWITCH-блок) для сокращения количества вызовов системы поддержки и минимизации количества пересылок. Для этих целей используется граф потока управления, который позволяет определить все пути использования, где массив мог быть использован и модифицирован.

5.5 Оценка и выбор схем распараллеливания

В результате успешного распараллеливания системой SAPFOR 2 последовательной версии программы будет получено несколько параллельных схем, которые отличаются количеством измерений DVM-шаблонов, которые были распределены. Например, если был построен всего один шаблон с N измерениями, то может быть получено 2^N параллельных схем. В случае двух и более шаблонов количество схем будет равно $\prod_{i=1}^{k} 2^{N_i}$, где k – количество DVM-шаблонов, N_i – размерность каждого i-го DVM-шаблона. Даже в случае одного трехмерного шаблона схем будет уже 8, и необходимо каким-либо образом выбрать наиболее эффективную схему среди предложенных.

По умолчанию пользователю доступны все параллельные схемы через модуль визуализации системы SAPFOR 2. Имеется возможность выбрать интересующую схему и выполнить ее оценку и построение. Наряду с этим имеется возможность оценить все доступные схемы и отсортировать их по значимости. Чем выше значимость схемы, тем она считается более эффективной по сравнению с остальными. Ставится задача оценки не каждой схемы по отдельности, а всех схем сразу, так как необходимо понять, насколько одна схема лучше другой и по какому критерию. Данное упрощение может быть введено по причине того, что схемы получаются из одной и той же программы с одними и теми же DVM-шаблонами и одним и тем же набором параллельных циклов. Схемы отличаются количеством обращений к удаленным данным из-за разного распределения DVM-шаблонов в каждой из схем.

В зависимости от необходимой точности оценки и наличия или отсутствия сведений о программе (например, количество витков циклов) может быть выбрана одна из стратегий:

– «поверхностное» оценивание на основе информации только от статического анализа;
– «упрощенное» оценивание на основе информации от статического и динамического анализов;

Рассмотрим каждую из стратегий. Стоит отметить, что в большинстве случаев статический анализ не способен определить необходимые для оценки параметры программы, например, такие, как количество витков циклов. Также статический анализ не может определить граф потока управления программы, так как переходы от одного оператора к другому могут быть параметризованы с помощью переменных, значения которых известны только на этапе выполнения. Напротив, динамический анализ способен определить значения переменных и граф потока управления, но для этого потребуется реальный запуск исходной последовательной программы, при этом полученные результаты анализа будут актуальны только для текущих входных данных.

«Поверхностное» оценивание основано только на информации от статического анализа. В данном случае предполагается, что количество витков цикла и граф потока управления не известны, но известно количество элементов массива, так как данная информация требуется для корректного распределения данных в модели DVMH. Данная оценка основана на подсчете негативных факторов, которые влияют на соответствующий вариант параллельной схемы. К негативным факторам относятся всевозможные коммуникации, возникающие в результате директив SHADOW_RENEW, REMOTE_ACCESS и REDISTRIBUTE/REALIGN. Для каждой директивы рассчитывается количество данных, которое необходимо передать по коммуникационной среде. Так как известны размеры массивов по каждому из измерений, то в соответствии с директивами рассчитывается количество байт, которое будет передано суммарно. В зависимости от директивы может быть передано как несколько плоскостей массива, так и весь массив целиком.

Соответственно, чем больше директив, приводящих к обменам, тем менее эффективной будет считаться схема. К недостаткам данной стратегии можно отнести тот факт, что не учитывается граф потока управления, то есть неизвестно, сколько раз будет выполняться данный цикл, в котором, допустим, присутствует доступ к удаленным данным с помощью REMOTE_ACCESS, требующий подгрузки всего массива. Данная операция может быть достаточно времязатратной, но может выполняться в программе только в начале ее выполнения.

Для устранения описанного недостатка была введена «упрощенная» стратегия оценивания. Методы оценки данной стратегии схожи с методами стратегии «поверхностного» оценивания, но с помощью динамического анализа можно узнать количество витков цикла и граф потока управления. Имея эту информацию, добавляется вес каждой операции обмена в соответствии с количеством ее выполнения в программе. К недостатку данной стратегии можно отнести тот факт, что программа должна быть запущена на каких-то входных данных (которые задает пользователь системы SAPFOR 2). Тем самым, при смене входных данных оценка соответствующих схем может поменяться.

Рассмотренные стратегии оценивания не зависят от количества процессоров, на котором будет запущена программа. Предполагается, что на каждое распределенное измерение DVM-шаблона будет отображено по крайней мере 2 MPI-процесса, иначе данное измерение можно считать размноженным. Учет количества процессоров, а также скорости коммуникаций и времен выполнения циклов позволил бы получить более точную оценку и выполнить подбор решетки процессоров, на которой предпочтительно запускать полученную DVMH-программу. Такая стратегия оценивания схемы достаточно трудоемкая и практически не реализуемая на практике, так как необходимо учитывать специфику работы многоядерных процессоров и графических ускорителей внутри узла кластера, а также поведение динамически настраиваемых функций библиотеки поддержки DVM-системы.

В зависимости от количества информации, которая может быть получена от статического и/или динамического анализов системой SAPFOR 2, выполняется оценка каждой схемы по первой или второй стратегии. Затем в модуле визуализации системы пользователю выводится соответствующий

рейтинг всех схем распараллеливания (либо какого-то их количества, которое может задать пользователь). По запросу пользователя для каждой выбранной схемы строится ее параллельная версия в модели DVMH.

Глава 6. Исследование алгоритмов автоматизированного распараллеливания

В данной главе описываются результаты экспериментальной проверки разработанной системы SAPFOR 2 на ряде тестовых программ и модельных задач. Также приводится сравнение возможностей распараллеливания новой системы SAPFOR 2 и предыдущей системы САПФОР.

6.1 Автоматическое распараллеливание небольших тестовых программ

Рассмотрим пример тестовой программы, для которой предыдущая система САПФОР работает неэффективно (см. Листинг 6.1). В данной программе итерационное пространство всех циклов одинаковое за исключением цикла в строке 13. Однако именно в этом цикле вычислений больше, чем в других циклах.

Листинг 6.1 — Пример тестовой программы

```
1  PROGRAM Simple
2  parameter (N=10000)
3  integer a(1:N), b(1:N), c(1:N)
4  DO i = 1,10000
5    a(i) = i
6  ENDDO
7  DO i = 1,10000
8    b(i) = i
9  ENDDO
10 DO i = 1,10000
11   c(i) = i
12 ENDDO
13 DO i = 1, 10000-25
14   a(i+1) = b(i)
15   a(i+1) = b(i-25) + b(i+25)
16 ENDDO
17 DO i = 1,10000
18   b(i) = c(i)
```

```
19 ENDDO
20 DO i = 1,10000
21   a(i+1) = c(i)
22 ENDDO
23 END
```

Циклы на строках 4, 7, 10 выполняют инициализацию данных. В данном примере инициализация всех массивов была намеренно сделана в разных циклах для того, чтобы не образовывались связи между всеми тремя массивами. В программе для эффективного выполнения таких циклов требуются следующие правила выравнивания массивов между собой:

* $B[i] => TEMPL[i]$
* $C[i] => TEMPL[i]$
* $A[i] => TEMPL[i-1]$

Первое и второе правила вытекают из цикла в строке 17. А последнее – из цикла в строке 13. Для последнего правила есть еще два варианта, соответствующие оператору в строке 15, когда $A(I+1)$ выровнен так же, как и $B(I-25)$ или $B(I+25)$. В зависимости от выбора правила выравнивания необходимо создавать теневую грань разной ширины. В случае выравнивания на $B(I-25)$ или $B(I+25)$ ширина теневой грани будет 50 элементов слева и 0 справа или 0 слева и 50 справа. В случае выбранного варианта ширина теневых граней будет по 25 слева и справа, что может быть эффективнее при наличии параллельных коммуникаций по следующим причинам. Во-первых, минимизируется ширина теневой грани для связи с соседним процессором, а, во-вторых, появляется возможность запустить теневые обмены параллельно с двумя соседями сразу.

Предыдущая система САПФОР позволяет связать массивы только по следующим правилам:

* $B[i] => TEMPL[i]$
* $C[i] => TEMPL[i]$
* $A[i] => TEMPL[i]$

Такого вида связи приведут к тому, что в цикле на строке 13 вместо теневых обменов по 25 элементов справа и слева необходимо выполнить подкачку всех частей массива B из-за того, что правила распределения данных не позволяют распределить данный цикл оптимальным образом (см. Листинг 6.2). Также можно заметить, что система САПФОР не смогла

связать массивы B и C с массивом A, что также приводит к неэффективным обменам.

Листинг 6.2 — Результат распараллеливания последних трех циклов системой САПФОР

```
!DVM$ DISTRIBUTE (BLOCK) :: a
!DVM$ DISTRIBUTE (BLOCK) :: b
!DVM$ ALIGN c(i) WITH b(i)

!DVM$ PARALLEL (i) ON a(i+1),REMOTE_ACCESS (b(:))
DO i = 1, 10000-25
  a(i+1) = b(i);
  a(i+1) = b(i-25) + b(i+25);
ENDDO

!DVM$ PARALLEL (i) ON b(i)
DO i = 1,10000
  b(i) = c(i);
ENDDO

!DVM$ PARALLEL (i) ON a(i+1),REMOTE_ACCESS (c(i))
DO i = 1,10000
  a(i+1) = c(i)
ENDDO
```

Благодаря использованию DVM-шаблонов и расширенному правилу выравнивания (правила выравнивания со сдвигом и раздвижкой), новая система SAPFOR 2 позволяет эффективно распараллеливать такие программы. В данной программе требуется выравнивание на DVM-шаблон, так как правила выравнивания для массива A выводят за пределы других массивов, то есть при использовании правила $A[i] => B[i-1]$ для отображения элемента $A(1)$ необходимо было бы иметь элемент массива $B(0)$. Результат распараллеливания программы системой SAPFOR 2 приведен в Листинге 6.3.

Листинг 6.3 — Результат распараллеливания последних трех циклов системой SAPFOR 2

```
!DVM$ ALIGN a(iEX1) WITH templ(iEX1 - 1)
!DVM$ ALIGN b(iEX1) WITH templ(iEX1)
!DVM$ ALIGN c(iEX1) WITH templ(iEX1)
!DVM$ TEMPLATE, COMMON :: templ(0:10000)
```

```
!DVM$ SHADOW b(25:25)

!DVM$ PARALLEL (i) ON templ(i), SHADOW_RENEW (b(25:25))
DO  i = 1,10000 - 25
  a(i + 1) = b(i)
  a(i + 1) = b(i - 25) + b(i + 25)
ENDDO
!DVM$ PARALLEL (i) ON templ(i)
DO  i = 1,10000
  b(i) = c(i)
ENDDO
!DVM$ PARALLEL (i) ON templ(i)
DO  i = 1,10000
  a(i + 1) = c(i)
ENDDO
```

Сравнивать времена запусков этих программ нет необходимости, так как из-за наличия REMOTE_ACCESS в программе на Листинге 6.2, время выполнения данной программы на нескольких процессорах, расположенных на разных узлах, только увеличится в силу того, что каждому процессору требуются целиком массивы B и C. Причем замедление будет пропорционально количеству используемых процессоров и узлов, так как суммарное количество коммуникаций будет также расти.

6.2 Результаты автоматизированного распараллеливания модельных задач

6.2.1 Сравнение систем автоматизации распараллеливания САПФОР и SAPFOR 2

Для проведения экспериментов по сравнению предыдущей и новой систем распараллеливания были выбраны следующие модельные программы: программы BT и LU из пакета тестов NAS[43] версии 2.3, а также CAVITY (каверна) и KONT (контейнер) – программы, разработанные в ИПМ им. М.В. Келдыша РАН [47]:

– программа CAVITY предназначена для моделирования циркуляционного течения в плоской квадратной каверне с движущейся верхней крышкой в двумерной постановке в широком диапазоне как параметров задачи, так и параметров численного метода;
– программа KONT предназначена для численного моделирования течения вязкой тяжелой жидкости под действием силы тяжести в прямоугольном контейнере с открытой верхней стенкой и отверстием в одной из боковых стенок в трехмерной постановке в широком диапазоне как параметров задачи, так и параметров численного метода.

Выбранные программы были распараллелены разработчиками DVM-системы с помощью модели DVMH. Таким образом эти программы уже были приведены к потенциально параллельному виду: необходимые преобразования кода для автоматизации распараллеливания были выполнены. Также была выполнена полная inline-подстановка процедур в данные программы, так как система САПФОР не справляется с процедурами.

Таблица 2 — Основные характеристики программ

	BT	LU	CAVITY	KONT
Количество строк	11300	3800	641	1025
Количество циклов	522	425	61	71
Количество массивов	32	33	18	37
Суммарное число измерений	70	66	33	97
Общий объем данных в параллельной программе	4 ГБ	4 ГБ	32 ГБ	0.9 ГБ
Объем данных на 1 процесс из 256 в параллельной программе	15 МБ	15 МБ	128 МБ	4 МБ

В Таблице 2 представлены основные характеристики распараллеливаемых программ. Количество строк кода считалось для фиксированного формата. Так как предыдущая система САПФОР не поддерживает распараллеливание программ с функциями, то после inline-подстановки программы BT и LU увеличились в объеме, в особенности BT (в 3.5 раза). Количество циклов – это количество DO-ENDDO циклов, которые анализировались и распараллеливались.

Таблица 3 — Время работы систем распараллеливания в секундах

	BT	LU	CAVITY	KONT
САПФОР	2820	1680	240	636
SAPFOR 2	20	10	7	7

В Таблице 3 представлено время работы систем распараллеливания. Можно отметить, что из-за использования базы данных время работы предыдущей системы распараллеливания в несколько раз больше.

Запуски производились на суперкомпьютере K10 [48], состоящем из 16 вычислительных узлов, в каждом из которых установлено два 8-ми ядерных процессора. Общее количество ядер равно 256. В Таблице 4 приведены времена запусков параллельных программ (для тестов NAS задавался класс C), полученных с помощью предыдущей системы САПФОР. В Таблице 5 приведены времена запусков параллельных программ, полученных с помощью разработанной системы SAPFOR 2. В скобках указано насколько время выполнения этих программ меньше времени выполнения программ, полученных с помощью системы САПФОР.

Таблица 4 — Времена (в секундах) запусков программ, распараллеленых системой САПФОР

# процессов	BT	LU	CAVITY	KONT
1 (1 узел)	2286	1543	15127	1585
2 (1 узел)	1126	780	7350	798
4 (1 узел)	578	398	3720	402
8 (1 узел)	310	214	2020	212
16 (1 узел)	173	123	1132	116
32 (2 узла)	94	67	567	61
64 (4 узла)	52	35	284	31
128 (8 узлов)	33	23	149	17
256 (16 узлов)	23	13	74	10

Параллельные программы системы SAPFOR 2 отличаются от полученных параллельных программ системой САПФОР, например. использованием DVM-шаблонов, но несмотря на это времена запусков практически совпадают.

Таблица 5 — Времена (в секундах) запусков программ, распараллеленых системой SAPFOR 2

# процессов	BT	LU	CAVITY	KONT
1 (1 узел)	2215 (+3.1%)	1551 (-0.5%)	15127 (+0%)	1585 (+0%)
2 (1 узел)	1095 (+2.8%)	784 (-0.5%)	7513 (-2.1%)	801 (-0.4%)
4 (1 узел)	554 (+4.3%)	402 (-0.9%)	3832 (-2.9%)	407 (-1.0%)
8 (1 узел)	302 (+2.7%)	220 (-2.6%)	2025 (-0.2%)	214 (-0.8%)
16 (1 узел)	172 (+0.1%)	127 (-2.9%)	1164 (-2.7%)	118 (-1.4%)
32 (2 узла)	96 (-1.8%)	68 (-1.4%)	573 (-1.0%)	61 (-0.5%)
64 (4 узла)	54 (-4.1%)	36 (-2.8%)	286 (-0.9%)	32 (-3.0%)
128 (8 узлов)	36 (-8.2%)	24 (-4.1%)	151 (-1.3%)	18 (-7.0%)
256 (16 узлов)	25 (-8.7%)	14 (-7.6%)	76 (-2.6%)	11 (-8.3%)

6.2.2 Распараллеливание тестов NAS версии 3.3

Версии тестов NAS 3.3 отличаются от версий 2.3 тем, что в версии 3.3 разработчиками тестов были выполнены оптимизации, которые позволили ускорить последовательную программу, но усложнили при этом процесс автоматизированного или автоматического распараллеливания.

За основу были взяты исходные программы, которые были написаны разработчиками этого пакета тестов. Системой SAPFOR 2 были распараллелены программы BT, LU, SP, EP, FT, CG и MG:

- программа BT (Block Tridiagonal Solver) – нахождение конечно-разностного решения 3-х мерной системы уравнений Навье-Стокса для сжимаемой жидкости или газа. Используется трехдиагональная схема, метод переменных направлений. В данной задаче в трех из основных вычислительных циклов есть зависимость по данным по одному измерению массива;
- программа LU (Lower-Upper Solver) – нахождение конечно-разностного решения 3-х мерной системы уравнений Навье-Стокса для сжимаемой жидкости или газа. Отличается от BT тем, что применяется метод LU-разложения с использованием алгоритма SSOR (метод верхней релаксации). В данной задаче в одном из основных

циклов есть зависимость по данным сразу по трем измерениям массива;

- программа SP (Scalar Pentadiagonal Solver) – нахождение конечно-разностного решения 3-х мерной системы уравнений Навье-Стокса для сжимаемой жидкости или газа. Программа структурно похожа на BT, но отличается тем, что применяется скалярная пятидиагональная схема. В данной задаче в трех из основных вычислительных циклов есть зависимость по данным по одному измерению массива;
- программа EP (Embarrassingly Parallel) – генерация независимых нормально распределённых случайных величин;
- программа FT (Fast Fourier Transform) – решение трехмерного уравнения в частных производных при помощи быстрого преобразования Фурье (FFT);
- программа CG (Conjugate Gradient) – приближение к наименьшему собственному значению большой разреженной симметричной положительно определенной матрицы с использованием inverse iteration вместе с методом сопряженных градиентов в качестве подпрограммы для решения СЛАУ;
- программа MG (MultiGrid) – аппроксимация решения трехмерного дискретного уравнения Пуассона при помощи V-циклового многосеточного метода.

В Таблице 6 представлены основные характеристики распараллеливаемых программ. Количество строк кода считалось для фиксированного формата. В системе SAPFOR 2 реализован межпроцедурный анализ свойств программы – поиск приватных переменных и связывание передаваемых в качестве параметров процедуры массивов. Данная возможность позволяет анализировать программы с процедурами и выполнять подстановку процедур только там, где это необходимо, например, в параллельном цикле. Отмена полной подстановки процедур делает программу более приближенной к исходной последовательной версии и ускоряет анализ кода системой SAPFOR 2.

Разработчиками тестов NAS предоставляются варианты параллельных программ с использованием MPI. С данными программами

Таблица 6 — Основные характеристики исходных программ NPB 3.3

Количество	**BT**	**LU**	**SP**	**EP**	**CG**	**FT**	**MG**
файлов	19	22	21	5	6	10	6
процедур	25	24	26	9	15	20	23
циклов	179	170	253	8	45	41	96
строк кода	3200	3000	2780	465	1120	1041	1714
массивов всего / распред.	47/9	68/13	40/10	6/0	44/18	32/14	110/71
SPF директив	18	5	15	1	0	3	11
времени работы SAPFOR 2, сек	10	10	10	2	10	10	10

производилось сравнение эффективности полученных параллельных DVMH-программ системой SAPFOR 2.

Для успешного автоматизированного распараллеливания необходимо, чтобы программа была приведена в потенциально параллельный вид – такой вид, при котором возможно автоматическое отображение последовательной программы в параллельную программу в модели DVMH. Если программа написана в потенциально параллельном виде (разработчик предполагал отображение своей программы на параллельные архитектуры), то никаких дополнительных модификаций исходного кода не требуется. Но, в основном, последовательные программы содержат разного рода зависимости, мешающие автоматическому отображению на параллельную архитектуру. Рассмотрим необходимые преобразования кода, которые были выполнены для приведения исходных тестов NAS к потенциально параллельному виду для отображения на кластер.

Для всех тестов, кроме теста CG, потребовалось указание свойств к потенциально параллельным циклам с использованием SPF-директив – приватизируемые массивы и редукция по массиву. Количество указаний, которые надо было вставить вручную через модуль визуализации системы SAPFOR 2, показано в Таблице 6. Приватизируемые массивы в циклах могут быть обнаружены только с помощью динамического анализа, а редукционные операции по массивам на данный момент не обнаруживаются даже динамическим анализом.

Для успешного отображения на гетерогенные узлы кластера, а также для увеличения параллелизма с целью задействования большего количества процессоров, необходимы следующие преобразования: расширение приватизируемых переменных, которое позволит устранить зависимость тесно-вложенного гнезда циклов, и разделение циклов, которое позволит увеличить количество распараллеливаемых тесно-вложенных циклов. Данные преобразования встроены в систему SAPFOR 2 и выполняются при расстановке соответствующих указаний с помошью SPF директив. Для запуска программ на небольших кластерах или на достаточно больших входных данных такие преобразования не потребовались.

Тест CG не требует модификации кода для приведения к потенциально параллельному виду для отображения только на узлы кластера.

Тесты BT и SP потребовали небольших изменений в коде (не более 20 строк) для того, чтобы обеспечить оптимальное выполнение полученной в результате распараллеливания системой SAPFOR 2 версии на кластере.

Тесты EP и FT потребовали небольших изменений в коде (не более 10 строк) для устранения конфликтов анализа по диагностикам системы SAPFOR 2. В основном изменения связаны с исключением возможности профилирования внутренних циклов – удалением вызовов процедур замера времени, которые не позволяют выполнять циклы параллельно.

Тест MG потребовал ручной модификации кода для приведения в потенциально параллельную форму, а именно устранение моделирования многомерных массивов на одномерном. Данная особенность в старых программах встречается достаточно часто в силу ограничений использования стандарта Фортрана 77. Для устранения данного конфликта были введены трехмерные массивы вместо одного одномерного, вручную было модифицировано около 400 строк кода. На основные вычислительные процедуры модификации не повлияли, так как эти процедуры принимали те же трехмерные массивы.

Тест LU потребовал наибольшее количество модификаций для приведения в потенциально параллельную форму, но в отличие от теста MG все модификации выполнялись автоматически с помощью преобразований, встроенных в систему SAPFOR 2. В данном тесте используется сложная трехмерная схема SSOR (Symmetric successive over-relaxation) с регулярными зависимостями по всем трем измерениям. Распараллеливание таких

циклов на кластер, многоядерные процессоры и ГПУ является достаточно трудоемкой задачей. Для приведения в потенциально параллельную форму данной программы требуются следующие преобразования кода: слияние и разделение циклов, расширение и сужение приватных переменных, подстановки процедур, внос инварианта. Все преобразования выполняются при расстановке соответствующих указаний с помошью SPF директив, а подстановка процедур может быть выполнена с помощью вызова соответствующего диалогового меню модуля визуализации системы.

Таблица 7 — Количество строк кода программ NPB 3.3

Количество	**BT**	**LU**	**SP**	**EP**	**CG**	**FT**	**MG**
Исходная	3200	3000	2780	465	1120	1041	1714
Пот.Паралл.	3225	3051	2795	472	1120	1044	2104
MPI	7672	6181	5773	1137	2615	2909	3042

В Таблице 7 показано насколько изменилось количество строк кода в результате приведения программ к потенциально параллельному виду. Для сравнения приведено количество строк кода MPI-программ, написанных разработчиками тестов NAS. В результате преобразований совокупное количество строк (13 811 строк) кода получается примерно в полтора раза меньше, чем суммарное количество строк MPI-версии тестов (21 657 строк), при этом потенциально параллельные версии в некоторых случаях могут быть автоматически отображены на кластеры с гетерогенными узлами в модели DVMH.

В данном случае по количеству строк кода можно оценить трудоемкость распараллеливания тестов NAS вручную и с помощью системы SAPFOR 2. Можно отметить, что MPI-версии, например, для тестов BT, SP, LU могут запускаться только на квадратных двумерных решетках вида NxN. Полученные параллельные версии в модели DVMH лишены данных особенностей и могут использовать любое количество процессов и графических ускорителей.

Производилось сравнение эффективности полученных системой SAPFOR 2 параллельных DVMH-программ с MPI-версиями. В Таблице 8 представлены результаты запусков исследуемых программ на 1 и на 16 узлах кластера K100 с использованием 1 процесса на узле. Версии

Таблица 8 — Времена (в секундах) запусков тестов NAS 3.3, распараллеленых с помощью системы SAPFOR 2 и MPI на кластере K100, класс C

# проц.	**BT**	**LU**	**SP**	**EP**	**CG**	**FT**	**MG**
1 SAPFOR 2	1024	770	888	317	227	257	141
1 MPI	1356	878	1067	411	307	242	135
16 SAPFOR 2	90.8	67.1	110.7	22.5	20.7	26.3	13.16
16 MPI	121.2	69.7	99.5	29.1	21.01	26.8	8.23

программ, полученные с помощью системы SAPFOR 2, не уступают, а в некоторых случаях выигрывают у MPI-версий, распараллеленых вручную.

Преимущество системы SAPFOR 2 перед ручным распараллеливанием заключается в том, что одна и та же программа может быть запущена на различных машинах с использованием не только MPI-процессов, а также многоядерных процессоров (нескольких нитей в одном процессе), но и графических ускорителей. Например, для использования в MPI-версии таких технологий, как OpenMP и CUDA, программисту придется дописать еще как минимум столько же строк кода, сколько есть в текущей MPI-версии. После этого необходимо будет отладить данный код, а для графического процессора потребуется его оптимизация.

Система DVM в данном случае поможет в отладке кода, что позволит сильно сократить время и потраченные программистом силы для получения приемлемой параллельной версии. Причем все модификации кода не обязательно проводить в параллельной версии, полученной при помощи системы SAPFOR 2, можно также выполнять их в исходной версии и повторять процесс распараллеливания заново.

Таблица 9 — Времена (в секундах) запусков тестов NAS 3.3, распараллеленых с помощью системы SAPFOR 2 и MPI на кластере K60, класс C

	BT	**LU**	**SP**	**EP**	**CG**	**FT**	**MG**
SERIAL	757	536	443	229	237	212	32.3
MPI	20.2	22.54	40.55	8.02	9.8	15.09	2.82
ЦПУ SAPFOR 2	32.54	20.05	70.2	5.85	10.35	25.6	3.55
ГПУ SAPFOR 2	26.5	15.4	–	0.15	16.89	–	2.36

В Таблице 9 приведены результаты запуска на одном узле с использованием двух 16-ти ядерных процессоров Intel Xeon Gold (всего 32 физических ядра и 64 потока) и одного графического ускорителя Tesla V100 следующих программ: исходных текстов (SERIAL), написанной на MPI, а также полученных системой SAPFOR 2. Для MPI-версий использовались 32 процесса (за исключением тестов BT и SP – 64 процесса), для ЦПУ SAPFOR 2 – 64 нити, для ГПУ SAPFOR 2 – один графический процессор. Прочерк означает невозможность запуска данной параллельной версии на графическом процессоре, так как требуются дополнительные преобразования и оптимизации кода, который вызывается из вычислительных регионов.

По результатам можно сделать вывод, что система SAPFOR 2 позволила получить параллельные программы тестов NPB 3.3 из непотенциально параллельных исходных последовательных версией не являющихся потенциально параллельными. Данные параллельные программы обладают следующими свойствами:

- были получены без существенной модификации кода, либо модификации кода с помощью реализованных внутри системы преобразований по указанию программиста;
- показывают эффективность, сравнимую с оптимизированными тестами, написанными с использованием MPI разработчиками пакета NPB 3.3;
- могут выполняться на ГПУ и многоядерных процессорах, а также на кластере, при этом максимально приближены к последовательной версии программы.

Также можно отметить, что для более эффективного выполнения на графическом процессоре требуются дополнительные модификации кода последовательной программы, которые пока что не могут быть выполнены системой SAPFOR 2 автоматически. Данные преобразования и оптимизации специфичны в основном для графических процессоров, но несмотря на это, программисту необходимо менять код последовательной программы, а работа по распараллеливанию по-прежнему будет лежать на системе SAPFOR 2.

6.2.3 Распараллеливание программы моделирования распространения упругих волн в средах со сложной 3D геометрией

Моделирование трехмерных упругих волн в средах различного строения является важным аспектом создания геофизических трехмерных моделей и изучения особенностей волновых полей. Решить обратную задачу геофизики (восстановление строения и параметров среды по экспериментально полученным записям сигналов) зачастую очень сложно, и одним из методов является решение набора прямых задач (моделирование сейсмополей в среде с заданными параметрами и строением) с варьированием значений параметров и геометрии среды при сравнении реальных данных с результатами моделирования.

Широко используемый метод для решения прямой задачи – метод конечных разностей. Отметим, что исследуемая область может иметь сложную геометрию трехмерной поверхности, поэтому важным отличительным моментом рассматриваемой задачи является построение криволинейной трехмерной сетки. Например, объектом исследования может быть магматический вулкан. Изучение строения среды и мониторинг подобного объекта является важной практической задачей, требующей больших вычислительных мощностей для достаточно быстрого получения результата.

Подобный подход к численному моделированию упругих волн подразумевает работу с большим количеством 3D данных. Учитывая масштабы области при решении реальных задач (сотни километров в каждом координатном направлении), задача численного моделирования становится не выполнима на персональной рабочей станции даже с установленным в ней графическим процессором [3].

При решении данной задачи используется построение криволинейной трехмерной сетки для расчетной области. Важнейшим моментом является ортогональность ребер ячеек возле свободной поверхности: все пересекающиеся ребра каждой криволинейной ячейки возле поверхности локально-ортогональны. Это означает, что в каждой точке поверхности вертикальные ребра ячеек перпендикулярны плоскости касательной к поверхности в этой точке. В этих же точках ортогональны и ребра, соответствующие горизонтальным направлениям.

Программа, реализующая описанный метод, состоит из 4 процедур, включая главную программу. Объем анализируемого кода составляет 3000 строк в фиксированном формате языка Фортран 95. Процедуры в данной программе вызываются с разными массивами, передаваемыми в качестве аргументов, что усложняет межпроцедурный анализ. В программе всего 198 массивов, которые могут быть распределены. В итоге, распределенными становятся только 51 из них. Программа содержит 95 циклов формата DO-ENDDO.

Данная программа разрабатывалась со всеми принципами ко-дизайна, важным моментом которого является учет дальнейшего отображения программы на имеющиеся параллельные архитектуры. Таким образом, последовательная программа уже находилась в потенциально параллельном виде, никаких дополнительных преобразований проводить не потребовалось.

Для распараллеливания данной программы системе SAPFOR 2 потребовалось примерно 20 секунд, а также расстановка трех директив для редукционных операций по массивам. В результате распараллеливания системой SAPFOR 2 было добавлено в программу 60 директив распределения данных и 15 директив распределения вычислений. Несмотря на такое малое количество директив распределения вычислений, в данной программе есть достаточно большие циклы. Например, один цикл занимает около 450 строк в фиксированном формате, а директива к данному циклу – 11 строк, так как в данном цикле большое количество приватных скалярных переменных (более 60). Распараллеливание данной программы вручную было бы достаточно трудоемким, а вероятность внесения ошибки очень высокой.

Автором программы была написана параллельная версия с использованием технологии MPI. Размер кода параллельной MPI-версии оценивается в 10000 строк. Трудоемкость написания такой версии достаточно высока, так как используются асинхронные пересылки. Для добавление OpenMP директив потребовалось бы еще несколько сотен строк кода, а также время на отладку. Подключение графических ускорителей потребовало бы как минимум двукратное увеличение строк кода, так как для каждого цикла необходимо создать его копию-ядро, которое будет выполняться на ГПУ. Также необходимо обеспечить выделение памяти, ее копирование и корректный запуск копий-ядер на ГПУ.

Оценка эффективности полученной DVMH-программы и параллельной программы с использованием технологии MPI выполнялась на суперкомпьютере K60 [49]. Раздел, на котором установлены графические ускорители NVidia Tesla V100, содержит 8 узлов, каждый из которых состоит из двух 16-ти ядерных процессоров Intel Xeon Gold 6142 v4 и четырех Tesla V100.

Была произведена оценка слабой масштабируемости. Для замера слабой масштабируемости был выбран такой размер данных, при котором на каждый процесс приходилось примерно по 12ГБ данных. Всего на один узел для использования четырех ГПУ отображались 4 процесса, таким образом в совокупности на один узел приходилось примерно 48ГБ данных. Результаты слабой масштабируемости представлены в Таблице 10.

Всего было задействовано 8 узлов кластера: 256 ядер ЦПУ и 32 ГПУ, размер задачи при этом составил примерно 384ГБ (исходя из 48 ГБ данных на один узел). Из Таблицы 10 видно, что DVMH-программа не уступает по скорости выполнения программе с ручным распараллеливанием, что подтверждает возможность эффективного автоматизированного распараллеливания системой SAPFOR 2 программ, приведенных к потенциально параллельному виду. При подключении графических ускорителей можно получить стократное ускорение по отношению к одному процессу или девятикратное по отношению ко всем ядрам ЦПУ.

Таблица 10 — Слабая масштабируемость (100 итераций, секунды)

# процессов (ГПУ)	**1(1)**	**32(4)**	**64(8)**	**128(16)**	**256(32)**
SAPFOR 2 (MPI)	211	28.7	29.1	29.0	28.8
SAPFOR 2 (MPI+ГПУ)	1.8	3.2	3.2	3.5	3.3
MPI	210	27.8	27.9	27.5	27.8

Разница между ручным распараллеливанием и DVMH-программой при использовании только ядер центрального процессора практически незаметна. Подключение графических ускорителей дает существенное ускорение программы. Для получения такой параллельной версии программы системе SAPFOR 2 в данном случае не понадобились какие-либо специальные указания от пользователя. Напротив, трудоемкость написания такой программы вручную достаточно высокая из-за большого количества кода и сложности отладки.

6.3 Опыт применения механизма областей распараллеливания

6.3.1 Инкрементальное распараллеливание теста NAS BT

Описание задачи

Использование механизма областей распараллеливания хорошо подходит в тех случаях, когда программист сталкивается с незнакомой ему программой. Чем больше рассматриваемая программа, тем сложнее процесс ее распараллеливания на кластер, так как необходимо представлять структуру программы, знать ее поведение при различных входных данных. Рассмотрим процесс распараллеливания последовательной исходной программы на языке Фортран с применением областей распараллеливания. Конечная цель – отображение всей программы на кластер с графическими процессорами. Будем действовать поэтапно: сначала отобразим программу на кластер, далее на кластер с многоядерными процессорами и ГПУ. Все описанные результаты запусков в данном разделе были получены на суперкомпьютере K60 [49] (раздел с ГПУ) с использованием компиляторов Intel версии 18 и CUDA версии 10.0. На разделе с ГПУ K60 в каждом узле установлены два 16-ти ядерных ЦПУ Intel Xeon Gold 6142 и четыре ГПУ Tesla V100.

В качестве демонстрационной программы будем использовать Block Tridiagonal Solver (тест BT из пакета тестов NAS версии 3.3), решающей 3-х мерную систему уравнений Навье-Стокса для сжимаемой жидкости или газа. В данной программе используется трехдиагональная схема и метод переменных направлений. Исходная версия данной программы содержит 3200 строк кода в фиксированном формате Фортран, 17 файлов и 19 процедур. Рассматриваемая программа является упрощенной версией используемой в инженерных расчетах программы, поэтому этапы загрузки и сохранения результатов счета были упрощены разработчиками данного пакета тестов.

Профилирование программы

Для того, чтобы понимать важность каждой процедуры, необходимо выполнить профилирование программы. Разработчики программы BT встроили профилирование программы на основе замеров времени выполнения вычислительных частей. Данная возможность может быть отключена без перекомпиляции программы. Основное время занимает вычислительная процедура ADI, которая в свою очередь вызывает пять процедур: три процедуры X_SOLVE, Y_SOLVE, Z_SOLVE, которые реализуют метод переменных направлений, а также две вычислительные процедуры $COMPUTE\_RHS$ и ADD. Времена выполнения последовательной программы и перечисленных выше процедур представлены в Таблице 11. Из Таблицы 11 видно, что 40% кода выполняются более 95% времени.

Таблица 11 — Профилирование последовательной программы BT

Название процедуры	Время выполнения, сек	Количество строк
Вся программа	636 (100%)	3200 (100%)
X_SOLVE	156 (24%)	320 (10%)
Y_SOLVE	196 (30%)	320 (10%)
Z_SOLVE	208 (32%)	320 (10%)
COMPUTE_RHS	70 (11%)	306 (9.5%)

Определить основную цепочку вызовов процедур можно несколькими способами. Первый способ – использовать модуль визуализации процесса распараллеливания системы SAPFOR 2. С помощью визуализатора можно построить граф вызовов процедур и выделить следующую цепочку вызовов: BT (главная программа) -> ADI (в итерационном цикле) -> Процедуры $X/Y/Z\_SOLVE$ и $COMPUTE\_RHS$. Второй способ – использовать всевозможные профилировщики, например, Intel Composer, GNU Profiler, Google Profiler и т.д. С помощью данных профилировщиков можно определить самые времяемкие процедуры в последовательной программе, а также цепочку их вызовов. И третий вариант – анализ кода и получение времен от встроенного профилирования, реализованного в программе. Совокупность данных методов позволяет получить детальное представление о программе, если рассматривать ее как «черный ящик».

Частичное распараллеливание программы с помощью областей

Проанализировав структуру программы, можно попытаться ее распараллелить снизу вверх, идя от самых внутренних процедур до самой внешней. Для того, чтобы локализовать действие системы SAPFOR 2 только на рассматриваемых процедурах, необходимо каким-то образом расставить области распараллеливания. Расставим 4 области распараллеливания – каждую времяемкую процедуру поместим в свою область.

Стоит отметить, что процедуры $X/Y/Z\_SOLVE$ являются похожими, так как реализуют метод переменных направлений. Суть данного метода заключается в том, что в каждой из процедур происходит прямая и обратная прогонки по соответствующему измерению массива: по X, Y и Z. Соответственно, по данным координатам присутствует зависимость в соответствующей процедуре. Далее, после рассмотрения только одной из них (например, X_SOLVE), будет подразумеваться, что аналогичные действия необходимо проделать и для остальных.

Для дальнейшего распараллеливания требуется выполнять анализ с помощью системы SAPFOR 2 и устранять возникающие в циклах проблемы, которые мешают их распараллеливанию. В данном случае потребовалась расстановка трех директив для приватизации массивов $FJAC$, $NJAC$, LHS перед циклами в процедурах $X/Y/Z\_SOLVE$ (по одной директиве на каждую процедуру), а также восьми директив, обозначающих начало и конец области распараллеливания. Процесс распараллеливания занял всего 1 минуту времени системы SAPFOR 2 (с учетом получения параллельной версии) и не более 10 минут времени программиста на получение и анализ данных от системы SAPFOR 2. В результате распараллеливания были получены следующие требования выравнивания данных:

- **область X_SOLVE:**
  $ALIGN(i,j,k)\ to\ TEMPL_0(*,j,k) :: QS, SQUARE$
  $ALIGN(*,i,j,k)\ to\ TEMPL_0(*,j,k) :: U$
  $ALIGN(*,i,j,k)\ to\ TEMPL_0(i,j,k) :: RSH$

- **область Y_SOLVE:**
  $ALIGN(i,j,k)\ to\ TEMPL_1(i,*,k) :: QS, SQUARE, RHO\_I$

$ALIGN(*,i,j,k)\ to\ TEMPL_1(i,*,k) :: U$
$ALIGN(*,i,j,k)\ to\ TEMPL_1(i,j,k) :: RSH$

– **область Z_SOLVE:**
$ALIGN(i,j,k)\ to\ TEMPL_2(i,j,*) :: QS, SQUARE, RHO\_I$
$ALIGN(*,i,j,k)\ to\ TEMPL_2(i,j,*) :: U$
$ALIGN(*,i,j,k)\ to\ TEMPL_2(i,j,k) :: RSH$

– **область COMPUTE:**
$ALIGN(i,j,k)\ to\ TEMPL_3(i,j,k) :: US,WS,QS,VS$
$ALIGN(i,j,k)\ to\ TEMPL_3(i,j,k) :: SQUARE,RHO\_I$
$ALIGN(*,i,j,k)\ to\ TEMPL_3(i,j,k) :: RSH,U,FORCING$

Для каждой области был построен свой вариант распределения данных и вычислений со своим DVM-шаблоном. Можно заметить, что в данном случае нельзя построить общего решения для всех областей сразу: так или иначе, любое выбранное отображение на шаблон будет конфликтовать с отображением в других процедурах. Данная проблема связана с тем, что программа не приведена к потенциально параллельному виду, и для получения эффективной параллельной программы требуются преобразования кода, такие как разделение циклов и расширение приватизируемых переменных. Эти преобразования реализованы в системе SAPFOR 2 и будут применены далее.

Выберем одну из 32-х параллельных схем с лучшей оценкой от системы SAPFOR 2 и запустим ее на выполнение на двух узлах на 64-х процессах. Полученная параллельная программа в модели DVMH содержит в себе 285 DVM-директив распределения данных и вычислений. В дальнейшем, когда будет произведено объединение областей, количество DVM-директив сократится.

В Таблице 12 приведены результаты запуска полученной параллельной программы, в скобках у параллельной версии указано «чистое» время выполнения области, а общее время содержит в себе также накладные расходы на вход и выход из области. По результатам видно, что распараллеленная системой SAPFOR 2 программа хорошо ускоряется и показывает до 76% эффективности при использовании 64 процессов. Но это «чистое»

Таблица 12 — Профилирование первой параллельной версии программы BT с областями (время в секундах)

Название процедуры	Исходная	Параллельная	«Чистое» ускорение
Вся программа	636	2960	
X_SOLVE	156	700 (3.6)	43.3 раз
Y_SOLVE	196	320 (4.8)	40.8 раз
Z_SOLVE	208	580 (4.2)	49.5 раз
COMPUTE_RHS	70	1120 (2.0)	35 раз

ускорение, которое может быть получено в пределе при нулевых накладных расходах на коммуникации при использовании нескольких узлов кластера. Если учитывать общее время, то получаем пятикратное замедление программы относительно исходной последовательной по причине того, что входы в области распараллеливания и выходы из них содержатся в итерационном цикле и выполняются на каждой итерации, а каждый вход и выход содержит в себе копирование массивов для сохранения корректного выполнения оставшейся части программы.

Объединение областей распараллеливания

Рассмотрим варианты объединения областей. В случае бесконфликтного объединения размноженное измерение, обозначенное «*», поглощает распределенное. Каждая область содержит в себе свой DVM-шаблон, который может быть распределен по следующим измерениям:

- **область X_SOLVE:**
  $TEMPL_0(*, j, k)$
- **область Y_SOLVE:**
  $TEMPL_1(i, *, k)$
- **область Z_SOLVE:**
  $TEMPL_2(i, j, *)$
- **область COMPUTE:**
  $TEMPL_3(i, j, k)$

Получается 3 варианта объединения:

– COMPUTE+Z_SOLVE+Y_SOLVE с отказом распределения по 2-му и 3-му измерениям;
– COMPUTE+X_SOLVE+Y_SOLVE с отказом распределения по 1-му и 2-му измерениям;
– COMPUTE+X_SOLVE+Z_SOLVE с отказом распределения по 1-му и 3-му измерениям.

Выполним объединение, например, COMPUTE+X_SOLVE+Y_SOLVE и получим следующее распределение данных:

– **область Z_SOLVE:**
$ALIGN(i,j,k)\ to\ TEMPL_2(i,j,*) :: QS, SQUARE, RHO\_I$
$ALIGN(*,i,j,k)\ to\ TEMPL_2(i,j,*) :: U$
$ALIGN(*,i,j,k)\ to\ TEMPL_2(i,j,k) :: RSH$

– **область COMPUTE:**
$ALIGN(i,j,k)\ to\ TEMPL_3(i,j,k) :: US,WS,QS,VS$
$ALIGN(i,j,k)\ to\ TEMPL_3(i,j,k) :: SQUARE,RHO\_I$
$ALIGN(*,i,j,k)\ to\ TEMPL_3(i,j,k) :: RSH,U,FORCING$

Выберем одну из 16-ти параллельных схем с лучшей оценкой от системы SAPFOR 2 и запустим ее на выполнение на двух узлах на 64-х процессах. Полученная параллельная программа в модели DVMH содержит в себе 244 DVM-директив распределения данных и вычислений (против 285 директив в первой версии). В Таблице 13 приведены результаты запуска полученной параллельной программы.

Таблица 13 — Профилирование второй параллельной версии программы BT с областями (время в секундах)

Название процедуры	Исходная	Параллельная	«Чистое» ускорение
Вся программа	636	1620	
X_SOLVE	156	3.6 (3.6)	43.3 раз
Y_SOLVE	196	4.8 (4.8)	40.8 раз
Z_SOLVE	208	580 (4.2)	49.5 раз
COMPUTE_RHS	70	1002 (2.0)	35 раз

Таким образом, вторая версия параллельной программы ускорилась почти в два раза по сравнению с первой версией параллельной программы,

но по-прежнему наблюдается замедление по сравнению с исходной последовательной версией программы. Далее невозможно выполнять объединение с учетом интересов всех областей, поэтому выполним объединение в одну область и вынесем данную область за пределы итерационного цикла. В результате распараллеливания с одной областью были получены следующие требования выравнивания данных:

– область BT:
  $ALIGN(i,j,k)\ to\ TEMPL(*,i,j,k) :: US,WS,QS,VS$
  $ALIGN(i,j,k)\ to\ TEMPL(*,i,j,k) :: SQUARE,RHO\_I$
  $ALIGN(m,i,j,k)\ to\ TEMPL(m,i,j,k) :: RSH,U,FORCING$

Для того, чтобы все измерения массивов были эффективно распределены на узлы кластера, необходимо не распределять первое измерение шаблона, а остальные – по желанию. В такой постановке есть всего 8 вариантов распределения шаблона: когда распределено только одно из измерений (3 варианта), когда распределены только два измерения (3 варианта) и когда распределены все три измерения или все три размножены.

В результате системой SAPFOR 2 будет получена третья версия параллельной программы. Исходя из того, что в процедурах $X/Y/Z\_SOLVE$ не было произведено никаких преобразований исходного кода, при выборе какого-либо распределения данных возникнет конфликт при распределении вычислений. После выполнения оценивания полученных 8-ми схем вариант с распределением только последнего измерения шаблона оказывается наилучшим. Полученная версия параллельной программы в модели DVMH содержит в себе 211 DVM-директив распределения данных и вычислений (против 244 директив во второй версии). В Таблице 14 отражены результаты запуска третьей версии параллельной программы. Из-за конфликтов распределения вычислений процедура Z_SOLVE плохо ускоряется, так как требуются перераспределения данных, которые порождают «лишние» коммуникационные обмены.

В данной версии в итерационном цикле отсутствуют накладные расходы на вход и выход из области распараллеливания, тем самым получается ускорить программу в 14 раз по сравнению с исходной версией. Но программа по-прежнему распараллелена не полностью. Для полного распараллеливания распространим найденные решения на всю программу.

Таблица 14 — Профилирование третьей параллельной версии программы BT с областями (время в секундах)

Название процедуры	Исходная	Параллельная	Ускорение
Вся программа	636	45	**14 раз**
X_SOLVE	156	3.6	43.3 раз
Y_SOLVE	196	4.8	40.8 раз
Z_SOLVE	208	30	6.9 раз
COMPUTE_RHS	70	6.0	11.6 раз

В итоге будет получена та же самая программа с таким же распределением данных, но она будет содержать уже 125 DVM-директив распределения данных и вычислений (против 211 директив в третьей версии). Времена запусков не будут отличаться, так как все вспомогательные процедуры находятся вне итерационного цикла.

Можно отметить, что все манипуляции над исходным кодом программы проводились с помощью модуля визуализации системы SAPFOR 2. Выполнялось указание свойств программы с помощью SPF-директив для приватизации переменных, а также расстановка областей распараллеливания. Далее запускались алгоритмы устранения конфликтов областей и построения распределения данных и параллельных схем. То есть не было необходимости изменять исходный код вручную, и программа оставалась наиболее близкой к исходной.

Преобразование программы с помощью системы SAPFOR 2

Для дальнейшего распараллеливания программы требуется ее преобразование, так как для использования графического ускорителя и большого количества процессов и нитей необходим больший ресурс параллелизма, чем есть в существующей программе. Исходная версия программы была оптимизирована для последовательного выполнения, поэтому в в X/Y/Z_SOLVE процедурах прямая и обратная прогонки выполняются в одном цикле. Для увеличения ресурса параллелизма необходимо разделить циклы, чтобы образовать тесно-гнездовые независимые между собой

циклы. А для этих целей необходимо выполнить преобразование расширение приватизируемых переменных для того, чтобы каждый виток цикла работал со своей «копией» приватизируемого массива.

Как уже было отмечено выше, расширение приватизируемых переменных и разделение циклов выполняется системой SAPFOR 2 по указанию директивы пользователя в коде программы и запуска соответствующего преобразования в модуле визуализации. В процедуру X_SOLVE для выполнения соответствующих преобразований необходимо вставить одну директиву для расширения приватизируемых переменных и одну директиву для разделения циклов. Аналогично для Y/Z_SOLVE. В процедуре COMPUTE_RHS необходимо также выполнить разделение циклов для образования трехмерных тесно-гнездовых циклов. Для этого необходимо добавить еще 4 директивы для разделения циклов.

Таким образом, необходимо расставить 10 директив в коде программы и запустить два преобразования через модуль визуализации системы SAPFOR 2, чтобы получить параллельную версию, которая будет лучше масштабироваться на большое количество процессов, а также сможет выполняться на графическом ускорителе.

Преобразование расширение приватизируемых переменных происходит таким образом, чтобы для каждого витка цикла данная переменная (или массив) были собственными. Для этих целей вводится новый массив и к нему добавляется необходимое количество измерений, равное количеству тесно-гнездовых циклов, для которого указано данное преобразование. В данной программе массивы FJAC, NJAC были трехмерными с формой (5,5,N), а LHS – четырехмерный с формой (5,5,3,N), где N – размерность решаемой задачи. После преобразования размерность массивов увеличилась на два, размерность которых также равна N: (5,5,N,N,N) и (5,5,3,N,N,N) соответственно. В следствие чего увеличилась и занимаемая ими память: для класса C (N = 162) массивы FJAC, NJAC стали занимать в памяти вместо 31 КБ – 850 МБ, для класса D (N = 408) массивы FJAC, NJAC стали занимать в памяти вместо 80 КБ – 13.5 ГБ. Соответственно массив LHS занимает еще в три раза больше памяти, чем FJAC и NJAC.

Полученная программа не является оптимальной и здесь есть простор для оптимизаций – массивы FJAC и NJAC можно «удалить», если вручную подставить все вычисления в вычислительном цикле, чтобы не

выполнять их расширение. Также можно сократить количество измерений массива LHS. Данные оптимизации выходят за рамки описываемой работы, так как считаются не формализуемыми и нетривиальными для автоматического выполнения.

Таблица 15 — Профилирование исходной и преобразованной последовательных программ BT

Название процедуры	Исходная	Преобразованная
Вся программа	636 сек (3200 стр.)	1494 сек (3510 стр.)
X_SOLVE	156 сек (320 стр.)	440 сек (416 стр.)
Y_SOLVE	196 сек (320 стр.)	480 сек (413 стр.)
Z_SOLVE	208 сек (320 стр.)	494 сек (424 стр.)
COMPUTE_RHS	70 сек (306 стр.)	78 сек (337 стр.)

В Таблице 15 представлены результаты запусков и количество строк кода полученной последовательной программы после преобразований. За счет увеличения количества обрабатываемой памяти после расширения приватных массивов наблюдается замедление программы почти в 2.3 раза, хотя общее количество строк кода увеличилось незначительно. Корректность программы проверяется «автоматически» с помощью встроенной функции в сам тест BT по завершении его выполнения.

После применения преобразований полученная параллельная версия в модели DVMH содержит 198 DVM-директив за счет того, что количество циклов, которое надо распараллелить, увеличилось. Также изменился и сам код программы. В Таблице 16 представлены времена запусков новой версии программы в модели DVMH, полученной с помощью SAPFOR 2 на тех же двух узлах и 64-х процессах. По сравнению с последовательной новая параллельная версия программы ускорилась в 25 раз или в 10.6 раз по сравнению с исходной версией.

Но несмотря на это, данная программа может быть выполнена на гибридном кластере с задействованием нитей и графических ускорителей. Например, данная программа может быть выполнена на двух 16-ти ядерных ЦПУ за 93.3 секунды или на одном графическом процессоре за 50 секунд. Полученное время для графического процессора можно оценить с точки зрения производительности на потраченный ватт энергии. Так, графический процессор при полной нагрузке тратит 300 BT, в то время как

Таблица 16 — Профилирование преобразованной параллельной версии программы BT (время в секундах)

Название процедуры	Исходная	Параллельная	Ускорение
Вся программа	1494 (636)	60	25 (10.6) раз
X_SOLVE	440 (156)	18	24 (8.6) раз
Y_SOLVE	480 (196)	19	25 (10.3) раз
Z_SOLVE	494 (208)	20	25 (10.4) раз
COMPUTE_RHS	78 (70)	2.0	39 (35) раз

четыре ЦПУ в общей сложности более 600 BT, что дает в 2.25 раз большую энерго-эффективность при запуске на ГПУ, чем при использовании ЦПУ.

Полученная параллельная программа может также быть запущена на нескольких графических ускорителях, но сложности реализации ACROSS в DVM-системе для графических процессоров не позволяет получить дополнительный выигрыш в таких конфигурациях, и программа выполняется медленнее, чем на одном ГПУ. Исследование реализации и потери производительности в DVM-системе выходит за рамки данной работы. Для эффективного выполнения программы на нескольких ГПУ требуется ее дальнейшая оптимизация и адаптация.

Для получения выигрыша на одном ГПУ можно попытаться применить преобразование не ко всей программе, а только для Z_SOLVE и COMPUTE_RHS. Судя по временам из Таблицы 15 получается примерно 1.5 кратное замедление последовательной программы вместо 2.3-х кратного после выполнения частичного преобразования. В результате такого объединения будут получены следующие времена, указанные в Таблице 17.

Таблица 17 — Времена запусков частично преобразованной параллельной версия программы BT на ГПУ (секундах)

	Исходная	Параллельная	Ускорение
Вся программа	636	38.4	16.5 раз

Таким образом получается лучшее ускорение на одном графическом ускорителе, чем было получено на двух узлах кластера при задействовании 64х процессов (см. Таблицу 14).

Ручная оптимизация и адаптация программы для кластера с несколькими ГПУ

Как было отмечено выше, для эффективного выполнения вычислительных регионов на нескольких ГПУ требуется дополнительная оптимизация, которая позволит получить еще больший выигрыш при использовании гетерогенного кластера. Большинство оптимизаций для ГПУ выполняются с целью минимизации количества чтений и записей в глобальную (общую) память и вынесения как можно большего количества вычислений на регистры. На современных архитектурах ГПУ скорость доступа к кэшу и регистровой памяти в два/три раза выше, чем к глобальной памяти.

Известно, что все скалярные переменные и массивы с константными размерами могут быть помещены на регистры. Таким образом, необходимо перенести выполнение всех важных вычислений на скалярные переменные и приватизируемые массивы. Данные оптимизации необходимо выполнить для процедур X/Y/Z_SOLVE, так как они получились наиболее времязатратными после отображение последовательной версии на ГПУ.

Для инициализации временного массива LHS используются два дополнительных массива FJAC и NJAC. Общее правило инициализации можно описать следующим формулами (для всех $i = 1..N$, $m1 = 1..5$, $m2 = 1..5$):

$$LHS(m1,m2,1,i) = FJAC(m1,m2,i-1) + NJAC(m1,m2,i-1)$$

$$LHS(m1,m2,2,i) = FJAC(m1,m2,i) + NJAC(m1,m2,i)$$

$$LHS(m1,m2,3,i) = FJAC(m1,m2,i+1) + NJAC(m1,m2,i+1)$$

Видно, что для инициализации i-го элемента требуется $i-1,\ i,\ i+1$ элементы двух других массивов, то есть необходимо хранить и вычислять всего три элемента массивов FJAC и NJAC вместо N. Тем самым можно снизить количество данных, которые нужно хранить в глобальной памяти ГПУ в несколько раз и сделать данные массивы приватизируемыми. Данная оптимизация увеличит количество вычислений в три раза, так как нам необходимо вычислить три элемента для каждого i-го элемента массива LHS вместо того, чтобы вычислить все N элементов заранее. Но так

как вычисления занимают порядка 2-16 тактов, а запрос к памяти порядка 300-500 тактов, то такие накладные расходы будут «покрыты».

Для выполнения данной оптимизации требуется подставить непосредственно все вычисления FJAC и NJAC в инициализацию массива LHS, а также удалить ненужные инициализации массивов FJAC/NJAC. Также можно вынести все вычисления на регистры для массива LHS – завести приватный массив LHS_P(5,5,3) и заменить использование LHS на введенный приватный массив. Для сохранения корректного счета в приватный массив требуется загрузить данные до начала вычислений и сохранить необходимые данные после.

Детально изучив код процедуры, например, X_SOLVE, состоящий примерно из 350 строк, можно заметить, что в прямой и обратной прогонке участвует массив размерностью LHS(5,5,1,N) вместо LHS(5,5,3,N), остальные элементы используются в промежуточных вычислениях и могут быть вынесены на регистры, что позволит в три раза сократить количество сохраняемых данных на ГПУ в глобальной памяти.

После выполнения описанных выше оптимизаций над последовательной программой получится эффективная параллельная программа для ГПУ в модели DVHM как по памяти, так и по вычислительной нагрузке. Описанная работа требует некоторых знаний об устройстве параллельной архитектуры ГПУ, а также требует экспериментального подхода, так как не всегда описанные действия могут привести к желаемому эффекту, а оценить то или иное преобразование с точки зрения эффективности (особенно для ГПУ) достаточно тяжело или же невозможно.

Сравнивать результаты работы полученной программы будем с той же MPI-программой на классе D. В результате выполненных преобразований DVMH-программа сможет работать на одном ГПУ при таком наборе данных, а также хорошо масштабироваться на несколько ГПУ. В Таблице 18 представлены результаты работы параллельных программ на кластере K60. Программа MPI была запущена на 256 процессах или на 16 узлах кластера, то есть были задействованы все ресурсы ЦПУ. Для получения того же ускорения DVMH-программу необходимо запустить на двух ГПУ. Полученная параллельная программа в модели DVMH после дополнительных ручных преобразований стала лучше выполняться на ГПУ и

Таблица 18 — Времена запусков оптимизированной параллельной версии программы BT, класс D

	# Устройств	Время (сек)	Ускорение (раз)
Последовательная	1 ЦПУ	13206	
MPI	256 ЦПУ	83	159
SAPFOR 2	1 ГПУ	108	122
SAPFOR 2	2 ГПУ	78	169
SAPFOR 2	4 ГПУ	60	220

хорошо масштабироваться за счет учета специфики реализации ACROSS и баланса между интенсивностью вычислений и доступом к памяти.

В заключение можно отметить следующие преимущества полученной параллельной версии BT в модели DVMH над MPI-программой: разработка, преобразование и поддержка программы ведутся в последовательной версии; программа может задействовать многоядерные процессоры и графические ускорители, что позволит тратить меньше вычислительных ресурсов кластера с большей эффективностью.

6.3.2 Инкрементальное распараллеливание программного комплекса COMPOSIT

Области распараллеливания были применены для распараллеливания программного комплекса COMPOSIT [50], который реализует моделирование многокомпонентной фильтрации при разработке месторождений нефти и газа. Композиционные модели фильтрации используются при подробном моделировании залежей, содержащих легкие углеводороды (конденсат и газ), а также когда необходимо более точно описывать массообмен между фазами, например, при изучении методов увеличения нефтеотдачи при закачке азота, углекислого газа, газов высокого давления.

Данный вычислительный комплекс содержит разные функции-решатели, функции ввода-вывода и функции для работы с контрольными точками. Для каждого решателя требуется свое распределение данных.

При определенном наборе параметров вызываются определенные решатели, поэтому рассмотрение системой SAPFOR 2 всего программного комплекса затруднительно. Комплекс написан на языке Фортран77 с применением механизма моделирования многомерных массивов на одномерном – используется один большой одномерный массив, который передается в функции как трехмерный.

Общее количество строк кода в программном комплексе составляет порядка 15 000 в фиксированном формате языка Фортран. Общее количество циклов в программе – 1018, общее количество объявленных массивов – 1533, а функций – 195. Система SAPFOR 2 не справляется со всем комплексом целиком из-за того, что достаточно большое количество функций принимают многомерные массивы как одномерные, а также есть часть функций, которые написаны на языке Си. Но на входных данных, на которых запускалось моделирование, такого рода функции ни разу не вызывались, а функции на языке Си осуществляют только ввод и вывод данных, которые не требуют распараллеливания. Для того, чтобы исключить неисполняемые операторы кода на конкретных входных данных, было получено профилирование с помощью GCov на маленьких тестовых данных. Эта информация позволила системе SAPFOR 2 игнорировать те операторы, которые ни разу не выполнялись. Во время создания параллельной версии программы система вставит предупреждающие печати и операторы останова в игнорируемых блоках программы на случай ее запуска на других отличающихся входных данных.

Алгоритм действия при исследовании данного комплекса был примерно таким же, как и при распараллеливании программы BT в разделе 6.3.1. Сначала было выполнено профилирование и найдены самые времязатратные процедуры. Затем был выполнен анализ тех процедур, которые выполняются дольше всего – процедуры обработки скважин. Они содержат в себе по одному большому циклу с вызовом большого количества более маленьких процедур. Таким образом распараллеливание снизу вверх начиналось с двух областей распараллеливания. В дальнейшем эти области были расширены и объединены в одну область вокруг итерационного цикла, тем самым сократив количество необходимого кода для анализа и распараллеливания до 4100 строк. Для исключения проблем анализа

система SAPFOR 2 дополнительно потребовала указать приватизируемые переменные-массивы для потенциально параллельных циклов. Было добавлено 6 директив системы SAPFOR 2, которые содержали 66 приватизируемых массивов.

Для одной из процедур потребовалось выполнить следующие два преобразования кода: расширение приватных массивов и расщепление циклов. Первое из них преобразует для тесного гнезда циклов ранга N приватный массив размерностью M в массив размерностью $M + N$, то есть происходит расширение массива или его «расприватизация» для данного гнезда циклов. Второе преобразование выполняет расщепление циклов для того, чтобы получить два тесно-гнездовых цикла без зависимостей по данным. Для данного преобразования как раз необходимо выполнить первое преобразование для устранения зависимостей между витками тесно-гнездовых циклов. Еще одно из типичных преобразований – внос инварианта цикла. Данное преобразование позволяет сделать циклы тесно-гнездовыми, что дает возможность системе SAPFOR 2 выполнять распараллеливание всего гнезда циклов. Все описанные преобразования выполняются автоматически системой SAPFOR 2 после расстановки программистом соответствующих SPF-директив системе через модуль визуализации.

В данном комплексе есть четыре процедуры, в которых рассчитываются некоторые характеристики по скважинам. Соответственно, циклы в таких процедурах организованы по скважинам, а не по сеточным элементам. В данном случае система SAPFOR 2 вставляет директивы доступа к удаленным данным для каждого оператора цикла, где используется доступ к нелокальному элементу распределенного массива. Этого можно избежать, если использовать специальную директиву DVM (!DVM$ ON), которая позволяет выполнять окруженный блок кода на том процессоре, где находятся распределенные данные, и без выполнения доступа к удаленным данным. Данное преобразование было выполнено вручную, так как на данный момент анализ для расстановки такой директивы для системы SAPFOR 2 достаточно трудоемкий.

Для получения результатов использовался суперкомпьютер K10, состоящий из 16 узлов. С целью оценки затрат времени на выполнение отдельных частей программы рассмотрен вариант, соответствующий разработке нефтяной залежи системой добывающих и нагнетательных скважин

(пятиточечная система с плотностью 50 га/скв) при закачке в пласт газа, обогащенного промежуточными фракциями. Использовано девятикомпонентное представление углеводородной системы, начиная с метана, этана и кончая псевдокомпонентами, соответствующими наиболее тяжелым фракциям нефти. При закачке жирного газа возможно образование закритических составов, которые нужно идентифицировать и для которых надо сохранять хорошее приближение, чтобы использовать в дальнейшем при возможном возвращении в докритическую область.

Таблица 19 — Времена в секундах выполнения программы на различных расчетных сетках

# проц.	**1**	**16**	**32**	**64**	**128**	**256**
21x21x6	37	6.1	4	3	2.9	3.61
201x201x6	3477	282	154	83.5	49.7	31.6
501x501x6	21690	1714	913	491	284	171

В Таблице 19 представлены времена выполнения программы. Из приведенных данных видно, что параллельная программа, полученная с помощью системы SAPFOR 2, показывает приемлемую эффективность при использовании 256 процессов на 16 узлах кластера K10 (ускорение в 127 раз). Наибольшая эффективность достигается при расчетах большего количества сеточных элементов и на 16-ти узлах составляет 50% по отношению к последовательной версии программы и 62% по отношению к одному вычислительному узлу. При этом уменьшение трудоемкости распараллеливания за счет применения областей и частичного распараллеливания можно оценить, по меньшей мере, в 15000/4100 раз, т.е. более, чем в 3.5 раза.

Заключение

Реализованная система SAPFOR 2 на текущий момент состоит из порядка 150 000 строк кода, написанных с использованием стандарта C++11 и языка C#: 55 000 строк кода разработаны лично автором, 45 000 строк кода были интегрированы, модифицированы и поддерживаются, 50 000 строк кода написаны на C# для модуля визуализации. Реализованные алгоритмы системы SAPFOR 2 были опробованы на более чем 200 000 строках кода в фиксированном формате языка Фортран 95 и показали эффективность предложенного подхода инкрементального распараллеливания на кластер.

Безусловно, использование новой архитектуры для системы SAPFOR 2, а также новых введенных возможностей: областей распараллеливания, автоматизации преобразований и новых алгоритмов распределения данных и вычислений, позволили существенно расширить класс программ, использующих структурные сетки, которые поддаются автоматизированному распараллеливанию. Несмотря на это, система SAPFOR 2 по-прежнему наталкивается на существенные трудности при распараллеливании исходных последовательных программ, которые были оптимизированы, прежде всего, для эффективного их выполнения на одном ядре.

Основной проблемой, которая препятствует отображению таких программ в параллельные версии, является их приведение к потенциально параллельному виду, такому виду, в котором не возникает существенных трудностей и конфликтов при распределении вычислений и данных на узлы кластера. Для устранения таких проблем требуется существенное изменение кода исходной программы, которое может быть выполнено путем автоматического выбора и выполнения необходимых преобразований программы. Дальнейшая работа по совершенствованию алгоритмов отображения последовательной программы в параллельную с помощью системы SAPFOR 2 будет направлена на преодоление описанных ограничений – создание типовых автоматических преобразований кода.

Основные результаты работы

- разработаны новые алгоритмы распараллеливания для кластеров с ускорителями (графическими процессорами) последовательных Фортран-программ научно-технического характера, использующих преимущественно регулярные вычисления:
  - построения наиболее перспективных вариантов распределения и перераспределения данных;
  - распределения вычислений и организации коммуникаций для каждого варианта распределения данных;
  - определения вычислительных регионов – фрагментов программы, которые целесообразно выполнять на ускорителях;
  - анализа входных/выходных данных вычислительных регионов и организации их актуализации;
  - выбора эффективных схем распараллеливания программы посредством грубого подсчета характеристик ее параллельного выполнения для каждой схемы.
- разработан новый метод инкрементального (пошагового) распараллеливания, который позволяет анализировать программы большого размера и выделять в программе области распараллеливания и задавать режим их анализа;
- реализованы новые алгоритмы построения схем распараллеливания в системе автоматизации распараллеливания SAPFOR 2 (в составе автоматически распараллеливающего компилятора), которая позволяет программисту участвовать в процессе распараллеливания программы (задавать свойства программы, выполнять ее преобразования, корректировать результаты алгоритмов автоматического распараллеливания);
- система автоматизации распараллеливания SAPFOR 2 была апробирована при распараллеливании тестов NAS NPB 3.3 и реальных практических Фортран-программ, созданных в ИПМ им. М.В. Келдыша РАН и других организациях. Результаты апробации подтвердили качество разработанных алгоритмов, продемонстрировав

эффективность получаемых параллельных программ, а также значительное ускорение и упрощение их разработки.

Автор выражает благодарность всему коллективу разработчиков DVM-системы, в совместной работе с которым были получены результаты распараллеливания тестовых и модельных программ, а также выявлены основные направления дальнейшего развития системы SAPFOR 2. Автор выражает отдельную благодарность Кузнецову Михаилу Юрьевичу за реализацию модуля визуализации для системы SAPFOR 2. Работа выполнена под руководством доктора физико-математических наук, Крюкова Виктора Алексеевича, которому автор выражает искреннюю признательность.

Автор выражает благодарность коллективу разработчиков системы V-RAY [51] под руководством Воеводина Валентина Васильевича и Воеводина Владимира Валентиновича, по инициативе которых была развернута работа по созданию системы автоматизации распараллеливания Фортран-программ для кластеров САПФОР [12]. Автор благодарен Барановой Татьяне Петровне, Вершубскому Валентину Юрьевичу и Ефимкину Кириллу Николаевичу – разработчикам статического анализатора ВерБа [52], на котором базировались первые версии системы САПФОР [12].

Список литературы

1. Сайт DVM-системы [Электронный ресурс]. — URL: http://www.dvm-system.org (дата обр. 01.01.2019).
2. Fortran-DVM – язык разработки мобильных параллельных программ / Н. А. Коновалов [и др.] // Журнал Программирование. — 1995. — № 1. — С. 49—54.
3. *Колганов, А. С.* Автоматизированное распараллеливание задачи моделирования распространения упругих волн в средах со сложной 3D геометрией поверхности на кластеры разной архитектуры / А. С. Колганов, Н. А. Катаев, П. А. Титов // Труды Международной научной конференции "Параллельные вычислительные технологии". — 2017. — С. 341—355.
4. *Колганов, А. С.* Автоматизированное распараллеливание задачи моделирования распространения упругих волн в средах со сложной 3D геометрией поверхности на кластеры разной архитектуры / А. С. Колганов, Н. А. Катаев, П. А. Титов // Журнал Вестник УГАТУ "Серия управление, вычислительная техника и информатика". — 2017. — Т. 21, № 3. — С. 87—96.
5. *Kolganov, A. S.* Automated Parallelization of a Simulation Method of Elastic Wave Propagation in Media with Complex 3D Geometry Surface on High-Performance Heterogeneous Clusters / A. S. Kolganov, N. A. Kataev, P. A. Titov // Journal Springer International Publishing Parallel Computing Technologies. — 2017. — No. 10421. — P. 32—41. — DOI: 10.1007/978-3-319-62932-2_3.
6. Productivity and Performance of Global-View Programming with XcalableMP PGAS Language / N. Masahiro [et al.] // Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. — 2012. — P. 402—409. — DOI: 10.1109/CCGrid.2012.118.
7. *Клинов, М. А.* Автоматическое распараллеливание Фортран-программ. Отображение на кластер / М. А. Клинов, В. Крюков //

Вестник Нижегородского государственного университета им. Н.И. Лобачевского. — 2009. — № 2. — С. 128—134.

8. *Zotov, S.* An Overview of the APC Compiler for Distributed Memory Machines / S. Zotov // Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications. — 2012. — P. 1939—1945.

9. Автоматизация распараллеливания программных комплексов / А. С. Колганов [и др.] // Труды Всероссийской научной конференции "Научный сервис в сети Интернет". — 2016. — С. 76—85. — DOI: 10.20948/abrau-2016.

10. Инкрементальное распараллеливание для кластеров в системе САПФОР / А. С. Колганов [и др.] // Труды Всероссийской научной конференции "Научный сервис в сети Интернет". — 2017. — С. 48—52. — DOI: 10.20948/abrau-2017.

11. Описание модели DVMH [Электронный ресурс]. — URL: http://dvm-system.org/static_data/docs/FDVMH-user-guide-ru.pdf (дата обр. 01.01.2019).

12. *Клинов, М. С.* Автоматическое распараллеливание некоторого класса фортран-программ. Отображение на кластер: автореф. дисс. канд. физ.-мат. наук / М. С. Клинов. — 2009.

13. Стандарт OpenMP 4.5 [Электронный ресурс]. — URL: https://www.openmp.org/wp-content/uploads/openmp-4.5.pdf (дата обр. 01.01.2019).

14. *Колганов, А. С.* Использование графических ускорителей для решения задач класса Data Intensive / А. С. Колганов // Труды Международной суперкомпьютерной конференции Научный сервис в сети Интернет: многообразие суперкомпьютерных миров. — 2014. — С. 79—88.

15. *Колганов, А. С.* Параллельная реализация алгоритма поиска минимальных остовных деревьев с использованием центрального и графического процессоров / А. С. Колганов // Труды Международной суперкомпьютерной конференции Параллельные вычислительные технологии. — 2016. — С. 530—543.

16. *Колганов, А. С.* Самая быстрая и энергоэффективная реализация алгоритма поиска в ширину на одноузловых различных параллельных архитектурах согласно рейтингу Graph500 / А. С. Колганов // Труды Международной суперкомпьютерной конференции Параллельные вычислительные технологии. — 2018. — С. 273—285.

17. *Колганов, А. С.* Использование графических ускорителей для решения задач класса Data Intensive / А. С. Колганов // Журнал Вестник УГАТУ "Серия управление, вычислительная техника и информатика". — 2014. — Т. 18, № 4. — С. 198—205.

18. *Колганов, А. С.* Параллельная реализация алгоритма минимальных остовных с использованием центрального и графического процессоров / А. С. Колганов // Журнал ЮУрГУ "Серия Вычислительная математика и информатика". — 2016. — Т. 5, № 4. — С. 5—19. — DOI: 10.14529/cmse160301.

19. *Kolganov, A. S.* Parallel implementation of minimum spanning tree algorithm on CPU and GPU / A. S. Kolganov // CEUR Workshop Proceedings, Germany). — 2016. — No. 1576. — P. 530—543.

20. *Колганов, А. С.* Поддержка интерактивности в системе САПФОР / А. С. Колганов, Н. А. Катаев, А. А. Смирнов // Труды Всероссийской научной конференции "Научный сервис в сети Интернет". — 2017. — С. 243—249. — DOI: 10.20948/abrau-2017.

21. *Колганов, А. С.* Оптимизация обработки изображений с ГПУ / А. С. Колганов // Труды Международной суперкомпьютерной конференции Параллельные вычислительные технологии. — 2018. — С. 266—272.

22. *Колганов, А. С.* Статический анализ приватных переменных в системе автоматизированного распараллеливания Фортран-программ / А. С. Колганов, Н. Н. Королев // Труды Международной суперкомпьютерной конференции Параллельные вычислительные технологии. — 2018. — С. 286—294.

23. An extension of the DVM system to solve problems with intensive irregular memory access / A. S. Kolganov [et al.] // Proceedings of the 4th GraphHPC conference on large-scale graph processing using HPC systems. — 2017. — No. 1981. — P. 25—30.

24. Опыт решения прикладных задач с использованием DVM-системы / А. Колганов [и др.] // Труды международной суперкомпьютерной конференции Russian supercomputing days. — 2017. — С. 650—661.

25. Расширение возможностей DVM-системы для решения задач, использующих нерегулярные сетки / А. Колганов [и др.] // Труды международной суперкомпьютерной конференции Russian supercomputing days. — 2016. — С. 596—603.

26. Автоматическое отображение Фортран-программ на кластеры с ускорителями / А. Колганов [и др.] // Труды Международной суперкомпьютерной конференции Научный сервис в сети Интернет. — 2014. — С. 17—22.

27. Dynamic tuning methods of dvmh-programs for clusters with accelerators / A. S. Kolganov [et al.] // CEUR Workshop Proceedings. — 2015. — No. 1482. — P. 257—268.

28. Методы динамической настройки DVMH-программ на кластеры с ускорителями / А. Колганов [и др.] // Труды Международной суперкомпьютерной конференции Russian supercomputing days. — 2015. — С. 257—268.

29. Использование Интернета для обучения параллельному программированию / А. Колганов [и др.] // Труды Международной суперкомпьютерной конференции Научный сервис в сети Интернет. — 2015. — С. 26—32.

30. Распараллеливание на языке Fortran-DVMH для сопроцессора Intel Xeon Phi тестов NAS NPB 3.3.1 / А. Колганов [и др.] // Труды Международной суперкомпьютерной конференции Параллельные вычислительные технологии. — 2015. — С. 19—30.

31. Распараллеливание тестов NAS NPB для сопроцессора Intel Xeon Phi на языке Fortran-DVMH / А. Колганов [и др.] // Журнал ЮУрГУ "Серия Вычислительная математика и информатика". — 2015. — Т. 4, № 4. — С. 48—62. — DOI: 10.14529/cmse150403.

32. Распараллеливание на графические процессоры тестов NAS NPB 3.3.1 на языке Fortran-DVMH / А. Колганов [и др.] // Журнал Вестник УГАТУ "Серия управление, вычислительная техника и информатика". — 2015. — Т. 19, № 1. — С. 240—250.

33. Распараллеливание на графические процессоры тестов NAS NPB 3.3.1 на языке Fortran-DVMH / А. Колганов [и др.] // Труды Международной суперкомпьютерной конференции Russian supercomputing days. — 2014. — С. 30—41.

34. Отображение на кластеры с графическими процессорами циклов с зависимостями по данным в DVMH-программах / А. Колганов [и др.] // Труды Всероссийской научной конференции "Научный сервис в сети Интернет". — 2013. — С. 250—257.

35. Отображение на кластеры с графическими процессорами DVMH-программ с регулярными зависимостями по данным / А. Колганов [и др.] // Журнал ЮУрГУ "Серия Вычислительная математика и информатика". — 2013. — Т. 2, № 4. — С. 44—56. — DOI: 10.14529/cmse130404.

36. Рейтинг TOP500 на ноябрь 2017 года [Электронный ресурс]. — URL: https://www.top500.org/list/2017/11/ (дата обр. 01.01.2019).

37. Система автоматизированного распараллеливания Paradigm [Электронный ресурс]. — URL: http://www.ece.northwestern.edu/cpdc/Paradigm/Paradigm.html (дата обр. 01.01.2019).

38. Система автоматизированного распараллеливания OlyMPIx [Электронный ресурс]. — URL: http://www.cs.cornell.edu/~kvikram/papers/pdcn.ps (дата обр. 01.01.2019).

39. Открытая распараллеливающая система [Электронный ресурс]. — URL: http://www.ops.rsu.ru (дата обр. 01.01.2019).

40. Система BERT77: Automatic and Efficient Parallelizer for FORTRAN [Электронный ресурс]. — URL: http://www.sai.msu.su/sal/C/3/BERT_77.html (дата обр. 01.01.2019).

41. Система ParaWise [Электронный ресурс]. — URL: http://www.parallelsp.com (дата обр. 01.01.2019).

42. Applied Parallel Research. FORGE Magic/DM [Электронный ресурс]. — URL: http://wotug.ukc.ac.uk/parallel/vendors/apr/ProductInfo/dpf_datasheet.txt (дата обр. 01.01.2019).

43. NAS Parallel Benchmarks [Электронный ресурс]. — URL: https://www.nas.nasa.gov/publications/npb.html (дата обр. 01.01.2019).

44. Библиотека Sage++ [Электронный ресурс]. — URL: http://www.extreme.indiana.edu/sage/ (дата обр. 01.01.2019).

45. *Niedzielski, D.* An Analytical Comparison of the I-Test and Omega Test / D. Niedzielski, K. Psarris // Proceedings of the Twelfth International Workshop on Languages and Compilers for Parallel Computing. — 2000. — P. 251—270.

46. *Pugh, W.* The Omega test: a fast and practical integer programming algorithm for dependence analysis / W. Pugh // Comm. of the ACM. — 1992. — Vol. 35, no. 8. — P. 102—114.

47. Использование языка Fortran DVMH для решения задач гидродинамики на высокопроизводительных гибридных вычислительных системах / В. Бахтин [и др.] // Журнал ЮУрГУ "Серия Вычислительная математика и информатика". — 2013. — Т. 2, № 3. — С. 106—120.

48. Гибридный вычислительный кластер К-10 [Электронный ресурс]. — URL: http://www.kiam.ru/MVS/resourses/k10.html (дата обр. 01.01.2019).

49. Гибридный вычислительный кластер К-60 [Электронный ресурс]. — URL: http://www.kiam.ru/MVS/resourses/k60.html (дата обр. 01.01.2019).

50. *Бахтин, В. А.* Использование параллельных вычислений для моделирования многокомпонентной фильтрации при разработке месторождений нефти и газа / В. А. Бахтин, А. В. Королев, Н. В. Поддерюгина // Тезисы международной конференции "Математика и информационные технологии в нефтегазовом комплексе". — 2016. — С. 164—166.

51. *Воеводин, В. В.* Параллельные вычисления / В. В. Воеводин, Вл. В. Воеводин. — Санкт-Петербург : Издательство БХВ-Петербург, 2002. — 608 с.

52. Развитие возможностей анализатора последовательных программ / Т. Баранова [и др.] // Труды Всероссийской научной конференции "Научный сервис в сети Интернет: решение больших задач". — 2008. — С. 38—42.

Приложение А

Описание внутреннего представления кода в системе SAPFOR 2 в библиотеке SAGE++

Внутреннее представление исходного кода программы на языке Фортран представляет собой AST (абстрактное синтаксическое дерево). Библиотека Sage++ [44] представляет собой реализацию данного дерева разбора. Все высокоуровневые функции, структуры и классы в Sage++ начинаются с префикса Sg, за исключением некоторых функций, которые проверяют некоторые свойства, например, $isSgForStmt(SgStatement^*)$ — функция, проверяющая, является ли выбранный узел циклом или нет.

Внутреннее представление создается с помощью парсера. Парсер использует грамматику для синтаксического анализа. Данная грамматика расширена для поддержки директив (спецкомментариев) DVMH и SAPFOR 2. Для того, чтобы начать работать с внутренним представлением, необходимо загрузить в память все файлы проекта, созданные парсером. Это делается с помощью класса $SgProject$, в конструктор которому передается имя текстового файла, в котором перечислен список $*.dep$ файлов. После создания проекта можно использовать следующий уровень абстракции — файл.

Файл является единицей трансляции, поэтому для того, чтобы переключиться на конкретный файл, необходимо использовать следующий вызов: $SgFile\ ^*file = \&(project.file(0))$. Данная функция получает файл с индексом 0. Также, класс $SgProject$ содержит в себе всю необходимую функциональность для работы с проектом, например, количество файлов в проекте ($project.numberOfFiles()$), или имя каждого файла в проекте ($project.fileName(0)$). Таким образом, стандартная фаза компиляции представляет собой проход по всем файлам для анализа кода программы пользователя.

Каждый файл проекта представляет собой набор связанных между собой операторов — $SgStatement$. Данный класс является абстрактным представлением всех операторов (базовым классом), от которого наследуются остальные классы для реализации специальных возможностей,

присущих отдельно взятому оператору. Например, *SgForStmt* является производным классом от *SgStatement* и содержит весь необходимый функционал для работы с оператором цикла. Согласно правилам языка C++ любой производный класс может быть приведен к базовому классу, что позволяет обрабатывать все операторы в дереве разбора как *SgStatement*, если нет необходимости исследовать их специфичные свойства.

В дереве разбора программист «видит» только *SgStatement*. К примеру, заголовок функции, оператор присваивания или оператор цикла — все это содержит в себе *SgStatement*. У каждого такого оператора есть вариант (SgStatement::variant()), который и задает вид конкретного оператора. Узнав вариант рассматриваемого оператора, можно использовать производное представление данного класса, преобразовав базовый класс к производному. Гарантируется, что данный оператор с правильным вариантом содержит всю необходимую информацию для производного класса. Например, чтобы узнать, является ли данный оператор оператором цикла, можно использовать код, представленный в Листинге A.1 или в Листинге A.2.

Листинг A.1 — Первый вариант получения оператора цикла

```
1 SgStatement *st = currSt;
2 if (st->variant() == FOR_NODE)
3     SgForStmt *forSt = (SgForStmt*) st;
```

Листинг A.2 — Второй вариант получения оператора цикла

```
1 SgStatement *st = currSt;
2 SgForStmt *forSt = isSgForStmt(st);
3 if (forSt)
4     DoSomth();
```

Помимо класса *SgStatement* есть класс *SgExpression*, представляющий собой выражения. Данный класс реализует выражения, которые есть у операторов. Например, следующий оператор присваивания $A[i] = B[i] + C[i]$ содержит в себе два выражения — то, что находится слева от оператора присваивания, и то, что находится справа. Для того, чтобы получить доступ к этим выражениям, необходимо использовать соответствующие функции класса *SgStatement*. Например, $expr(N)$ позволяет получить выражение N. Если выражения с номером N не существует, то

вернется пустой указатель ($NULL$). Всего оператор может содержать не более трех выражений (то есть $N = 0, 1, 2$). Данные выражения представляют собой $SgExpression$, которыми и наполняется оператор.

Класс $SgExpression$ является базовым классом для представления выражений. У данного класса есть такая же функция для взятия варианта (SgExpression::variant()). Используя данную функцию, можно узнать, с каким именно выражением необходимо работать и выполнить соответствующее преобразование к производному классу. Способ преобразования и проверки для выражений такой же, как и для операторов.

Все операторы файла ($SgStatement$) связаны между собой, есть понятие следующего оператора за данным в лексическом порядке и предыдущего оператора перед данным в лексическом порядке. Также у каждого оператора есть родительский оператор, который задает уровни вложенности операторов. Таким образом, следующий оператор, который идет за данным, не обязательно должен принадлежать текущей области вложенности (иметь одного и того же родителя). Например, в коде (см. Листинг A.3) за оператором $op1$ в строке 2, лексически следует оператор $op2$ в строке 3, за $op2$ — конец IF блока в строке 4, а за концом IF блока — $op3$ в строке 5.

Листинг A.3 — Пример условного оператора в языке Фортран

```
1 if (condition) then
2   op1
3   op2
4 endif
5 op3
```

Узнать, какой оператор является родителем для данного оператора, можно с помощью функции SgStatement::controlParent().

Стоит отметить оператор с вариантом $CONTROL\_END$. Данный оператор определяет конец блока операторов языка Фортран, например, $ENDIF$, $ENDDO$, $ENDFUNCTION$ и т.д. Для определения родителя для данного оператора нужно использовать также функцию SgStatement::controlParent(). У каждого оператора есть функция определения последнего оператора для данного — SgStatement::lastNodeOfStatement(). Если оператор является составным, например, $IF - ENDIF$, то последним оператором будет $ENDIF$ с вариантом $CONTROL\_END$.

В отличие от операторов выражения связаны в правое рекурсивное двоичное дерево. У каждого узла есть левый потомок (SgExpression::lhs()) и/или правый потомок (SgExpression::rhs()). Каждый потомок также является *SgExpression*. Каждый узел может иметь только левого потомка, либо только правого, либо вообще может не иметь потомков (в данном случае соответствующие функции вернут пустой указатель). Рекурсивно обойти такое дерево из выражений можно, например, так, как приведено в Листинге A.4.

Листинг A.4 — Обход дерева выражений

```
1  static void recExpression(SgExpression *exp, const int lvl)
2  {
3    if (exp)
4    {
5      SgExpression *lhs = exp->lhs();
6      SgExpression *rhs = exp->rhs();
7
8      doSmth()
9
10     recExpression(lhs, lvl + 1);
11     recExpression(rhs, lvl + 1);
12   }
13 }
```

У каждого оператора и выражения есть возможность получения его исходного кода на языке Фортран, то есть можно выполнить генерацию кода отдельного взятого оператора и выражения. Соответствующая функция называется *unparsestdout*(). Данная функция позволяет выполнить генерацию кода в консоль. Она служит в основном для отладки. Стоит заметить, что если вызвать данную функцию для оператора «Функция» или «IF блок», то вместе с этим оператором будут сгенерированы все вложенные в данный операторы (или все те операторы, у которых родитель — данный оператор). Аналогично и для выражений — будет сгенерирован код для всего бинарного дерева, начиная от текущего узла и ниже.

Для удобства отладки в SAPFOR 2 была реализована функция *recExpressionPrint*(*SgExpression *exp*), которая позволяет получить наглядное представление бинарного дерева разбора для выражений в формате GraphViz. Именуются узлы графа по такому правилу:

NODENUM_LVL_LR_TAGNAME_VALUE, где NODENUM – номер узла, LVL – глубина узла в дереве, LR – левое или правое это поддерево, TAGNAME – имя варианта, VALUE – значение, которое было в исходном коде, если оно доступно (например, имя символа, функции или операции). Данная функция на начальном этапе может существенно упростить процесс отладки и понимание того, как устроено внутреннее представление.

Помимо операторов и выражений есть таблицы символов ($SgSymbol$) и типов ($SgType$), которые свои для каждого файла и общие для всех операторов и выражений в данном файле. Символы представляют собой наполнение выражений и операторов.

Например, в приведенном выше выражении, есть символы A, B, C, которые являются символами следующего типа: одномерный массив из double (базовый тип double, производный – массив из одного измерения), а символ I является символом с типом integer. В данном выражении для всех обращений к I будет ссылка на таблицу символов к единственному экземпляру I. Таблицы символов и типов доступны по соответствующей функции класса $SgFile$. На базе встроенных типов можно строить производные типы. Таблицы типов и символов доступны на уровне файла.

Приложение Б

Описание спецкомментариев в системе SAPFOR 2

Листинг Б.1 — БНФ формы спецкомментариев системы SAPFOR 2

```
<SPF директива> ::= !$SPF <тип>
<тип>           ::= ANALYSIS   (<спец1> [, <спец1>]) |
PARALLEL   (<спец2> [, <спец2>]) |
TRANSFORM  (<спец3>) |
PARALLEL_REG <идент> | END PARALLEL_REG

<спец1> ::= <редукция> | <приватные>
<спец2> ::= <теневые грани> | <рег. зависимости> |
<удаленная ссылка>
<спец3> ::= NOINLINE

<редукция>      ::= REDUCTION (<ред. лист> [, <ред. лист>])
<ред. лист>     ::= <операция> (<идент>) | <операция_loc>
(<loc_ident>)
<операция>      ::= max | min | sum |  prod | and | or | eqv |
neqv
<операция_loc> ::= minloc | maxloc
<loc_ident>     ::= (<идент>, <идент>, <Константа>)

<приватные>      ::= PRIVATE (<идент> [, <идент>])
<теневые грани> ::= SHADOW (<описание массива>
[, <описание массива>])

<рег. зависимости> ::= ACROSS (<описание массива>
[,<описание массива>])
<описание массива> ::= (<идент> (<Константа>:<Константа>
[, <Константа>:<Константа>])
<удаленная ссылка> ::= REMOTE_ACCESS (<access_list>
[,<access_list>] )

<access_list> ::= <идент>(<expr>)
<expr>         ::= <идент> [<op> <expr>] |  <Константа>
[<op> <expr>] |  <идент> [<op> (<expr>)] |
<Константа> [<op>( <expr>)]
```

```
<ор>          ::= * | + | − | /
<Буква>       ::= [a−z] | [A−Z]
<Цифра>       ::= [0 −9]
<Константа>   ::= <Цифра>[ <Константа>]
<идент>       ::= <Буква> { <Буква> | <Цифра>}
```