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»

Автореферат
диссертации на соискание учёной степени
кандидата физико-математических наук

Москва — 2020

Работа выполнена в Федеральном государственном учреждении «Федеральный исследовательский центр Институт прикладной математики им. М.В. Келдыша Российской академии наук» и в Федеральном государственном бюджетном образовательном учреждении высшего образования «Московский государственный университет им. М.В. Ломоносова» на кафедре системного программирования факультета вычислительной математики и кибернетики.

Научный руководитель:	доктор физико-математических наук, профессор, главный научный сотрудник Института прикладной математики им. М.В. Келдыша РАН **Крюков Виктор Алексеевич**
Официальные оппоненты:	**Галатенко Владимир Антонович,** доктор физико-математических наук, заведующий сектором автоматизации программирования НИИСИ РАН
	Волконский Владимир Юрьевич, кандидат технических наук, начальник отделения 3.5 «Системы программирования» ПАО ИНЭУМ им. И.С. Брука
Ведущая организация:	Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского»

Защита состоится 20 октября 2020 года в 11:00 часов на заседании диссертационного совета Д002.024.01 при ИПМ им. М.В. Келдыша РАН, 125047, Москва, Миусская пл., д. 4.

С диссертацией можно ознакомиться в библиотеке и на сайте ИПМ им. М.В. Келдыша РАН http://keldysh.ru.

Автореферат разослан « » _____ 2020 года.

Ученый секретарь
диссертационного совета
Д002.024.01,
кандидат физико-математических
наук М. Г. Широбоков

Общая характеристика работы

Актуальность темы. 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 — это 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 общей памяти 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 для многих программ практически невозможно потому, что при переходе от последовательной программы к параллельной в большинстве

случаев требуется ее серьезное преобразование или даже изменение алгоритма.

Для распараллеливания программы на системах с общей памятью 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, но и распределение данных, которое должно быть согласовано с распределением вычислений (то есть распределенные вычисления должны выполняться теми процессорами, на которые отображены соответствующие им данные). Помимо этого, необходимо обеспечить на каждом процессоре доступ к удаленным данным, таким данным, которые расположены на других процессорах.

С целью автоматизации распараллеливания программ создаются высокоуровневые языки параллельного программирования, такие как HPF, OpenMP-языки, DVM-языки, CoArray Fortran, UPC, Titanium, Chapel, X10, Fortress, XcalableMP, а также создаются системы автоматизации распараллеливания программ, такие как Paradigm, CAPTools/Parawise, FORGE Magic/DM, BERT77, ParalWare Trainer, Appolo, САПФОР, ДВОР, АРС, которые используют для отображения на параллельные вычислительные системы как языки низкого уровня, так и языки высокого уровня.

Но, несмотря на разнообразие систем автоматического или автоматизированного распараллеливания, используемые в них решения обладают общим недостатком — все они используют анализ и/или преобразование абсолютно всей исходной последовательной программы. В случае отображения на системы с общей памятью 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. И, зачастую, для частичного распараллеливания полный анализ кода не требуется, но на данный момент не существует средств, которые позволяют отобразить на гибридный кластер только часть большой последовательной программы.

Для решения этой проблемы 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.

Цели и задачи. 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 эффективное распараллеливание 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 механизма, который позволяет проводить поэтапное (инкрементальное) распараллеливание последовательных Фортран-программ для гетерогенных кластеров;
- разработка и проектирование новой системы SAPFOR 2 и исследование полученной системы на тестовых и реальных прикладных программах.

Научная новизна. В результате данной работы 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 о распараллеливаемом программном комплексе.

Спроектирован, разработан и реализован механизм областей распараллеливания, что существенно расширило класс задач, к которым

можно применить систему SAPFOR 2. Механизм областей, позволяющий выделить 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.

Была разработана, спроектирована и реализована новая система (SAPFOR 2) автоматизации отображения Фортран-программ на гетерогенный кластер с поддержкой инкрементального распараллеливания. Предложенная структура системы позволяет легче расширять ее — 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.

Проведенные исследования применимости системы SAPFOR 2 для тестов и реальных приложений показали высокую эффективность предложенного подхода, позволяющего значительно упростить и ускорить разработку эффективных параллельных программ для гетерогенных кластеров, и 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.

Теоретическая и 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 функционал новыми возможностями благодаря использованию модульной организации алгоритмов анализа и преобразований.

Апробация работы. Основные результаты диссертационной 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». Основные положения и результаты диссертации были опубликованы в 8 научных изданиях, из них 4 включено в перечень рекомендованных ВАК РФ: 2 статьи в журналах, индексируемых в базах данных Scopus и/или Web of Science [2, 7], 2 статьи в журналах из списка ВАК [6, 8].

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, решающего задачу моделирования добычи залежей нефти и газа, разработанном в федеральном научном центре НИИСИ РАН.

Личный вклад. Содержание диссертации и основные положения, выносимые на защиту, отражают персональный вклад автора в проведенное исследование. Разработка, проектирование и реализация основных компонентов системы SAPFOR 2 и алгоритмов распределения данных, вычислений и отображения на ускорители проведены лично автором.

Содержание работы

Во **введении** обосновывается актуальность исследований, проводимых в рамках данной диссертационной работы, формулируется цель, ставятся задачи работы, излагается научная новизна и практическая значимость представляемой работы, приводится краткое описание каждой главы.

В **главе 1** описываются проблемы отображения на кластер и причины возникновения автоматизированных и автоматических систем распараллеливания для кластеров. Автоматизированное (а также и ручное) распараллеливание для кластера гораздо сложнее, чем на общую память 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 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.

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. Существующие 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 и вычислений. А ведь, зачастую, распараллеливание всей программы не требуется;
– сильное вовлечение программиста в принятие ответственных решений по распределению данных.

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

Далее приводится обзор экспериментальной версии системы САПФОР, разработанной в Институте прикладной математики им. М.В. Келдыша РАН. Некоторые из рассмотренных ограничений очень сильно сужают класс задач, которые можно распараллелить такой системой. Обозначим те ограничения, которые существенно влияют на процесс распараллеливания: обязательная подстановка процедур, использование языка

Фортран 77, отсутствие преобразований кода в системе, распараллеливание только полностью всей программы, массивы могут быть связаны между собой только «один к одному» (без сдвигов и раздвижки).

В **главе 2** описывается схема работы и структура организации системы SAPFOR 2. Новая система была призвана устранить описанные недостатки предыдущей системы автоматизации распараллеливания САПФОР и расширить класс программ, использующих структурные сетки, к которым можно было бы ее применить. 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 позволил реализовать полноценный межпроцедурный анализ 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, поиск зависимостей по данным.

Применение межпроцедурного анализа отменяет требование полной подстановки процедур во всей программе. Подставлять процедуры требуется лишь в некоторых случаях, например, когда внутри одного параллельного цикла есть вызов процедуры, в которой находится другой 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 позволяет выполнять более точный статический анализ кода программы.

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

Далее описывается предложенная в данной работе структура системы автоматизации распараллеливания программ SAPFOR 2. В основу были положены некоторые идеи архитектурных решений современных компиляторов, таких как LLVM или GCC. Все алгоритмы анализа и преобразования организованы в виде так называемых проходов. Проходом будем называть совокупность некоторых простых функций, выполняющих какой-либо анализ или преобразование над исходным кодом пользовательской

программы. Каждый проход состоит из двух фаз: независимая обработка всех файлов проекта и объединение результатов обработки по файлам (так называемая агрегирующая фаза). Наличие обеих фаз является не обязательным требованием к созданию проходов.

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.

В **главе 3** описываются основные объекты анализа программы — циклы и функции. Рассматриваются алгоритмы заполнения дерева циклов и графа вызовов функций. Для построения графа циклов необходимо выполнить сначала процедурный анализ, заполнив всю информацию о цикла в каждой функции, а затем объединить полученные свойства циклов с учетом вызовов функций. Структура каждого цикла содержит в себе свойства, которые описывают наличие ограничений на распараллеливание данного цикла, наличие в нем зависимостей, информацию об обращениях к массиву, информацию о полученной DVMH-директиве к данному циклу.

Среди всех типов циклов, которые можно использовать в языке Фортран (в том числе циклы, определяемые через *GOTO*), преобразуются в описанную структуру только циклы $DO - ENDDO$. Данное требование обусловлено тем, что с точки зрения DVMH-модели параллельными могут быть именно такие циклы (за исключением бесконечных 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.

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 (такого цикла, перед которым стоит DVM-директива распределения вычислений), а также использование таких циклов внутри пользовательских процедур или функций. Данное ограничение, прежде всего, связано с организацией работы с массивами в программе. Рассмотрим данные ограничения подробнее.

В модели DVMH существуют понятия *распределенный* массив и *нераспределенный* массив. Распределенным массивом будем называть такой массив, который был распределен между процессорами средствами DVM с помощью директив $DISTRIBUTE$ или $ALIGN$. Нераспределенным будем называть 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 вызовов функций системы поддержки DVMH, распределенные и нераспределенные массивы будут представлены по-разному. Таким образом, использование одного «экземпляра» процедуры, 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 их дублирование.

В **глава 4** дается описание механизма областей распараллеливания. Очень часто бывает, что распараллелить существующие большие программные комплексы полностью невозможно. Но зачастую в этом нет необходимости. Достаточно определить наиболее времяёмкие места в программе и постараться сначала распараллелить их. В данной работе рассматривается реализация метода инкрементального, или частичного, распараллеливания программ на гетерогенный кластер. Его идея заключается в том, что распараллеливанию подвергается не вся программа целиком, а её части (области распараллеливания), в которых заводятся дополнительные экземпляры требуемых данных, а также выполняется дублирование конфликтных функций, и в дальнейшем производится распределение этих данных и соответствующих им вычислений.

Область распараллеливания — это 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. Именно область распараллеливания будет анализироваться и распараллеливаться системой. Изначально вся программа является областью распараллеливания по умолчанию (*DEFAULT*).

Каждая такая область имеет свой уникальный идентификатор и строится путем добавления в нее или исключения из нее одного или нескольких фрагментов, а также анализируется системой SAPFOR 2 независимо от других областей. При этом запрещено иметь вложенные области. Если явные области не заданы в коде программы, то система по умолчанию будет рассматривать и пытаться распараллелить всю программу целиком, которая включена в область по умолчанию. Несмотря на то, что системой запрещается иметь вложенные области распараллеливания, это не гарантирует, что области после объявления их в коде программы не пересекаются ни по данным, ни по функциям. Например, вполне вероятно, что две области вызывают одну и ту же функцию. В этом случае строки кода такой

функции входят неявно в обе области. Другими словами, в областях всё равно могут иметь место конфликты. Поэтому система SAPFOR 2 на этапе анализа проводит дополнительную проверку выделенных областей и разрешает имеющиеся конфликты, если такие есть.

Алгоритм разрешения конфликтов принимает на вход дерево циклов и граф вызовов функций. Далее производится межпроцедурный анализ программы на предмет наличия прямых или косвенных конфликтов по массивам и функциям. Формально, конфликтные ситуации можно определить следующим образом:

- функция F вызывает функцию G явно, если в теле этой функции присутствует вызов функции G;
- функция G вызывает функцию H косвенно, если в этой функции нет вызова функции H, но присутствует вызов функции F, которая, в свою очередь, вызывает функцию H явно или косвенно;
- две области распараллеливания I и J являются конфликтными, если пересечение множеств вызываемых явно или косвенно функций из областей I и J не пусто. Такие функции будем называть конфликтными;
- массив A является конфликтным, если он используется в более, чем одной области распараллеливания.

После нахождения конфликтов, составляются множества конфликтных массивов и функций. Для каждой конфликтной функции и области распараллеливания, где эта функция вызывается, создаётся копия функции, после чего в области производится замена вызовов оригинальной функции на функцию-копию. Для каждого конфликтного массива, за исключением common-массива, и области распараллеливания, в которой этот массив используется, создаётся массив-копия и добавляется оператор копирования данных из оригинального массива в массив-копию до начала входа в область и оператор копирования из массива-копии в оригинальный массив после выхода из области.

Глава 5 посвящена алгоритмам построения распределения данных и вычислений, отображению на гетерогенный кластер. В разделе 5.1 дается описание способов отображения на кластер и их характеристика. Существует два основных подхода — с размножением данных и с распределением данных на узлы кластера. Последний подход является более перспективным, так как при размножении данных невозможно решить задачу больше, чем помещается на один узел кластера. В данной работе реализован подход распределения данных посредством указания соответствующих директив DVMH-модели. Для эффективной работы параллельной программы необходимо обеспечить лучшее распределение данных для конкретной программы, чтобы количество пересылок, а также их объем, между процессорами были минимальны.

Для построения параллельных схем необходимо определить, какие массивы надо отображать на узлы кластера и как они будут связаны между собой. Связывание массивов между собой позволит существенно сократить количество построенных параллельных схем. 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. Тогда если строить все варианты только по правилу распределять ли очередное измерение или не распределять, то количество вариантов будет $2^{S_{dim}}$.

В разделе 5.2 описывается алгоритм связи массивов в программе между собой. Так как основной ресурс параллелизма это циклы, то именно они задают правила связи массивов или их измерений. Связыванию подлежат измерения разных массивов, в индексных выражениях к которым присутствует одна и та же переменная цикла. После обработки всех операторов в программе, будет построена информация об обращениях к массивам с привязкой этих обращений к циклам с коэффициентами $D * x + E$, где D,E — вычисленные константы, а x — итерационная переменная цикла. Остальные обращения к массивам будут порождать неизбежные обмены между узлами кластера.

В результате работы алгоритм создает граф массивов, который 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 исследования в области параллельной обработки графов. Для реализации был выбран формат графа CSR (Compressed Sparse Rows). Данный формат получил широкое распространение для хранения разреженных матриц и графов, и позволяет эффективным образом реализовывать различные алгоритмы поиска наиболее важных компонент в графе. Каждое измерение массива становится узлом графа, а дуги показывают то, как одно измерение массива связано с другим. Для заполнения графа массивов необходимо обработать информацию о вычисленных коэффициентах и связи с циклом. Сложность алгоритма, который заполняет граф — линейная в зависимости от количества операторов в программе.

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:

– 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 вычисляются 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. Вес цикла оценивается статическим образом, либо путем динамического профилирования (получение времени выполнения данного цикла). Данный вес показывает количество раз, которое данный цикл был выполнен за все время работы в программе. Например, если цикл выполняется всего один раз (момент инициализации), то можно пожертвовать количеством коммуникаций в пользу итерационного цикла, который может выполняться сотни, а то и тысячи раз, где каждый переданный лишний 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.

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, так как распределение данных и вычислений для них происходит независимо.

Граф массивов в общем случае может содержать абсолютно разные дуги между измерениями массивов. Построенные связи в графе могут приводить к конфликтам отображения на узлы кластера, поэтому их

необходимо устранить. Для поиска лучшего решения (и устранения конфликтов) было выполнено исследование алгоритмов выбора совокупности дуг в графе.

Так как дуги задают приоритет их значимости с помощью веса, первым был исследован и реализован переборный алгоритм выбора наиболее значимого набора дуг для устранение конфликтов массивов. Для того, чтобы выбрать какой-либо набор дуг, необходимо оценить потери в случае отказа от других дуг. Отказ от какой либо дуги в графе приводит к неизбежным коммуникациям между узлами кластера. Тем самым, для того, чтобы минимизировать объем коммуникаций, необходимо минимизировать суммарный вес удаляемых дуг из графа массивов. Для этого необходимо перебрать все возможные варианты конфликтных ситуаций и выполнить поиск наиболее перспективного по заданному критерию.

Так как данная задача является NP-трудной, сложность алгоритма даже с использованием оптимизации в виде отсечения и предварительной сортировки узлов и дуг графа по возрастанию их веса, является степенной в зависимости от количества узлов графа. Такая сложность не позволяет за приемлемое время и количество используемой памяти строить лучший вариант для распараллеливаемой программы. Точное решение можно получить только на небольших программах.

В связи с этим был рассмотрен, исследован и реализован другой алгоритм устранения конфликтов дуг в графе — модифицированный алгоритм поиска минимального остовного дерева. 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 качестве модификации был рассмотрен поиск максимального остовного дерева, такого дерева, где сумма весов ребер будет максимальной. Данная модификация алгоритма решает поставленную задачу выбора набора дуг с линейной сложностью в зависимости от количества узлов или дуг графа. Минусом данного алгоритма является то, что мы можем получить один из перспективных вариантов распределения данных, при котором будет получена эффективная параллельная программа, но мы не можем гарантировать, что будет выбран наилучший вариант распределения данных.

В результате применения полученных алгоритмов на практике было выяснено, что при стремлении количества узлов к бесконечности, а также стремлении степени связанности графа к максимальной (когда каждая вершина связана с каждой) с помощью двух алгоритмов могут быть получены одинаковые решения по распределению данных.

В разделе 5.3 описывается алгоритм построения распределения вычислений и доступа к удаленным данным. 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 разным. Так, например, в случае отказа от распределения всех измерений распределяемых массивов, нет необходимости организовывать теневые обмены между узлами кластера, так как все необходимые данные и так будут находиться на каждом узле в таком варианте программы.

В разделе 5.4 описывается алгоритм расстановки вычислительных регионов — фрагментов программы, которые целесообразно выполнять на ускорителях. Для этого выполняется проверка процедур на наличие побочных эффектов (данные эффекты не позволяют выполнять процедуру параллельно). Также происходит объединение подряд идущих регионов для уменьшения количества входов в них. Для управления актуализацией данных (их перемещением между оперативной памятью ЦПУ и оперативной памятью ускорителей) используется анализ потока данных по графу потока управления. С помощью расширенного межпроцедурного анализа достигающих определений, анализируются используемые данные внутри и вне регионов для организации их актуализации.

В разделе 5.5 описывается алгоритм выбора эффективных схем распараллеливания программы посредством грубого подсчета характеристик ее параллельного выполнения. Для выполнения такой оценки, необходимо наличие следующей информации: размеры распределенных массивов, количество раз, которое выполняется каждый оператор, количество витков цикла, время выполнения гнезда циклов. Исходя из этих данных, а также на основе расставленных директив распределения данных и вычислений, система SAPFOR 2 оценивает потери на коммуникации и степень параллелизма циклов. Суммарная оценка показывает, насколько эффективной является текущая схема распараллеливания по сравнению с остальными схемами распараллеливания.

В **главе 6** приводятся основные результаты исследования реализованных алгоритмов автоматизированного распараллеливания в системе SAPFOR 2 на тестовых и прикладных программах, характеристики системы SAPFOR 2 и выносимые на защиту основные результаты работы.

Распараллеливание тестов NAS версии NPB 3.3. Распараллеливанию подвергались исходные версии основных тестов BT, LU, SP, EP, CG, FT и MG. Из всех тестов LU и MG требуют существенных преобразований кода. В первом из них используется схема SSOR с зависимостями по всем трем измерениям обрабатываемых массивов. Циклы, в которых обрабатываются данные массивы, разбиты вызовами процедур, поэтому для приведения к тесно-гнездовому виду потребуется подстановка процедур. Во втором — используется моделирование динамических массивов.

Распараллеливание LU и MG потребовало следующих преобразований: подстановка необходимых процедур для образования тесно-гнездового

цикла, объединение циклов, разделение циклов, внос инварианта, расширение приватных переменных, сужение приватных переменных, переход от моделирования массивов к обычным. Все перечисленные преобразования, кроме последнего, можно сделать с помощью системы SAPFOR 2, указав соответствующие директивы системе.

Таблица 1 — Основные характеристики 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
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

В Таблице 1 представлены основные характеристики распараллеливаемых программ. Каждая исходная версия программы преобразовывалась в потенциально параллельную. Потенциально параллельная версия — это такая версия, которая может быть автоматически отображена системой SAPFOR 2 в эффективную параллельную программу в модели DVMH без участия пользователя.

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 подстановки процедур делает программу более приближенной к исходной последовательной версии и существенно ускоряет анализ кода системой SAPFOR 2.

Разработчиками тестов NAS предоставляются варианты параллельных программ с использованием MPI. В Таблице 2 представлено количество строк кода исходных последовательных версий программ, потенциально параллельных версий — таких версий, для которых с помощью системы SAPFOR 2 можно автоматически получить эффективные параллельные программы, и MPI версий (количество строк кода считалось для фиксированного формата).

В данном случае по количеству строк кода можно оценить трудоемкость распараллеливания 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. Полученная параллельная версия в модели DVMH лишена данных особенностей

Таблица 2 — Количество строк кода различных версий NPB 3.3

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

и может использовать любое количество процессов. Также в общем случае DVMH-программа может быть эффективно выполнена на различной архитектуре без дополнительных преобразований кода (кластер, кластер с многоядерными процессорами и графическим ускорителями).

С MPI версиями производилось сравнение эффективности полученных системой SAPFOR 2 параллельных DVMH-программ. В Таблице 3 представлены результаты запусков исследуемых программ на 1 и на 16 узлах кластера K100 с использованием 1 процесса на узел. Версии программ, полученные с помощью системы SAPFOR 2, не уступают и в некоторых случаях выигрывают у MPI версий, распараллеленых вручную.

Таблица 3 — Времена (в секундах) запусков программ системы SAPFOR 2 и MPI-версий 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

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.

В Таблице 4 приведены результаты запуска на одном узле с использованием двух 16-ти ядерных процессоров Intel Xeon Gold и одного графического ускорителя Tesla V100 следующих программ: исходной (SERIAL), написанной на MPI, а также полученной системой SAPFOR 2.

Таблица 4 — 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

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. 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.

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.

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 распределенными. 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 требовалось.

Для распараллеливания данной программы системе SAPFOR 2 от пользователя понадобилась 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 исходной версии программы была написана параллельная версия с использованием технологии MPI. Размер кода данной программы оценивается в 10000 строк. Трудоемкость написания такой версии достаточно высока, так как используются асинхронные пересылки. Для добавления OpenMP директив потребовалось бы еще несколько сотен строк кода, а также времени на отладку. Подключение графических ускорителей потребовало бы как минимум двухкратное увеличение строк кода, так как для каждого цикла необходимо создать его копию-ядро, которое будет выполнено 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. Раздел, на котором установлены графические ускорители NVidia Tesla V100, содержит 8 узлов, каждый из которых состоит из двух 16ти 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 приходится примерно по 12ГБ данных. Всего на один узел для использования четырех ГПУ отображались 4 процесса, таким образом в совокупности на один узел приходится примерно 48ГБ данных. Результаты слабой масштабируемости представлены в Таблице 5.

Всего было задействовано 8 узлов кластера: 256 ядер ЦПУ и 32 ГПУ, размер задачи при этом составил примерно 384ГБ (исходя из 48ГБ данных на один узел). Из Таблицы 5 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 9-кратное по отношению ко всем ядрам ЦПУ.

Таблица 5 — 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

Распараллеливание программы моделирования многокомпонентной фильтрации при разработке месторождений нефти и газа. В данной задаче применялся подход инкрементального (поэтапного) распараллеливания. Композиционные модели фильтрации используются при подробном моделировании залежей, содержащих легкие углеводороды (конденсат и газ), в том случае, когда необходимо более точно описывать массообмен между фазами. Часто эти модели используются для изучения методов увеличения нефтеотдачи при закачке газов высокого давления, азота, углекислого газа и других агентов.

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 входных данных. После расстановки одной области распараллеливания вокруг итерационного цикла количество необходимого кода для анализа и распараллеливания составило 4100 строк.

В самом начале работы над данной программой в область были включены самые затратные расчетные функции итерационного цикла. Так как вход и выход в каждом фрагменте области это достаточно дорогие операции, область вручную была расширена на весь итерационный цикл. Для успешного распараллеливания система SAPFOR 2 дополнительно потребовала указать приватизируемые переменные-массивы для потенциально параллельных циклов. Было добавлено 6 директив системе 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 на основе построенного распределения данных и директив распределения вычислений в момент их вставки.

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),

Таблица 6 — Времена в секундах выполнения программы

# 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

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 анализ для расстановки такой директивы достаточно трудоемкий для системы SAPFOR 2 на данный момент.

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. В плане вычислений 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. В Таблице 6 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.

Реализация

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, появление в системе SAPFOR 2 новых возможностей, автоматизирующих выполнение преобразований и обеспечивающих инкрементальное распараллеливание позволило существенно расширить множество программ, которые могут быть успешно распараллелены для гибридных кластеров. Несмотря на это, система 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 ее параллельную с помощью системы SAPFOR 2 будет направлена на преодоление описанных ограничений — создание типовых автоматических преобразований кода.

Основные результаты работы

- спроектированы, разработаны и реализованы алгоритмы распараллеливания последовательных Фортран-программ для кластеров с ускорителями (графическими процессорами):
  - 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.
- алгоритмы построения схем распараллеливания реализованы в автоматически распараллеливающем компиляторе, являющимся ядром новой системы автоматизации распараллеливания SAPFOR 2,

которая позволяет программисту активно участвовать в процессе распараллеливания своей программы, если это необходимо (задавать свойства программы, выполнять ее преобразования, корректировать результаты алгоритмов автоматического распараллеливания);
- для поддержки инкрементального (пошагового) распараллеливания спроектированы и разработаны средства выделения в программе областей и задания режимов их распараллеливания;
- система автоматизации распараллеливания SAPFOR 2 была апробирована при распараллеливании тестов NAS NPB 3.3 и нескольких Фортран-программ, созданных в ИПМ им. М.В. Келдыша РАН и других организациях. Результаты апробации подтвердили качество разработанных алгоритмов, продемонстрировав эффективность получаемых параллельных программ и значительное ускорение и упрощение их разработки.

Публикации автора по теме диссертации

1. А. С. Колганов, С. В. Яшин. Автоматическое инкрементальное распараллеливание больших программных комплексов с помощью системы SAPFOR // Труды Международной научной конференции «Параллельные вычислительные технологии». — 2019. — С. 275—287.
2. A. S. Kolganov, N. A. Kataev. The experience of using DVM and SAPFOR systems in semi automatic parallelization of an application for 3D modeling in geophysics // Journal Springer International Publishing Parallel Computing Technologies. — 2018. — P. 286—291. — DOI: 10.1007/s11227-018-2551-y.
3. А. С. Колганов, Н. А. Катаев, А. А. Смирнов. Поддержка интерактивности в системе САПФОР // Труды Всероссийской научной конференции «Научный сервис в сети Интернет». — 2017. — С. 243—249. — DOI: 10.20948/abrau-2017.
4. А. С. Колганов, В. А. Бахтин, В. А. Крюков и др. Автоматизация распараллеливания программных комплексов // Труды Всероссийской научной конференции «Научный сервис в сети Интернет». — 2016. — С. 76—85. — DOI: 10.20948/abrau-2016.
5. А. С. Колганов, В. А. Бахтин, В. А. Крюков и др. Инкрементальное распараллеливание для кластеров в системе САПФОР // Труды Всероссийской научной конференции «Научный сервис в сети Интернет». — 2017. — С. 48—52. — DOI: 10.20948/abrau-2017.
6. А. С. Колганов, Н. А. Катаев, П. А. Титов. Автоматизированное распараллеливание задачи моделирования распространения упругих волн в средах со сложной 3D геометрией поверхности на кластеры разной

архитектуры // Журнал Вестник УГАТУ «Серия управление, вычислительная техника и информатика». — 2017. — Т. 21, № 3. — С. 87—96.
7. A. S. Kolganov, N. A. Kataev, P. A. Titov. Automated Parallelization of a Simulation Method of Elastic Wave Propagation in Media with Complex 3D Geometry Surface on High-Performance Heterogeneous Clusters // Journal Springer International Publishing Parallel Computing Technologies. — 2017. — No. 10421. — P. 32—41. — DOI: 10.1007/978-3-319-62932-2_3.
8. А. С. Колганов, В. А. Бахтин, В. А. Крюков и др. Решение прикладных задач с использованием DVM-системы // Журнал ЮУрГУ «Серия Вычислительная математика и информатика». — 2019. — Т. 8, № 1. — С. 89—106. — DOI: 10.14529/cmse190106.

Подписано в печать 29.06.2020.
Формат 60×84/16. Усл. печ. л. 1. Тираж 65 экз. Заказ А-30
ИПМ им. М.В. Келдыша РАН. 125047, Москва, Миусская пл., д. 4.