Автоматическое распараллеливание Фортран-программ. Отображение на кластер*

М.С. Клинов, В.А. Крюков

В работе рассматриваются алгоритмы автоматического отображения на кластер последовательной программы на языке Фортран. Они базируются на результатах статического анализа программы, который может при необходимости дополняться или корректироваться программистом. Автоматическое отображение на кластер основано на построении различных вариантов распараллеливания (распределения данных и вычислений между процессорами, а также организации доступа к удаленным данным - данным, расположенным на других процессорах) и их оценке путем моделирования выполнения соответствующих вариантов параллельных программ.

1. Введение

Высокопроизводительные вычислительные машины продолжают постоянно совершенствоваться, улучшаются технологии передачи данных, увеличивается количество ядер внутри процессоров. Это делает их использование все более и более привлекательным для программистов. Эти машины позволяют не только решать задачи быстрее, но и оперировать с большим объемом данных, тем самым, обеспечивая более точный расчет. Однако разрабатывать программы для таких машин трудно.

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. Это часто заставляет программистов сначала написать и отладить программы без использования параллелизма, а потом их распараллеливать. Процесс распараллеливания отлаженной последовательной программы (будем называть ее исходной программой) целесообразно максимально автоматизировать, а в идеале – осуществлять полностью автоматически, без участия программиста.

Среди возможных подходов к автоматизации распараллеливания программ можно выделить два основных:

- использовать диалог с программистом и для уточнения свойств последовательной программы, и для выбора наилучших решений по ее распараллеливанию;
- использовать диалог с программистом только для уточнения свойств последовательной программы, а выбор наилучших решений по ее распараллеливанию осуществлять полностью автоматически, без участия программиста.

С точки зрения этих двух подходов можно взглянуть на имеющиеся программные средства, которые помогают разрабатывать программы для высокопроизводительных ЭВМ.

Среди программных средств для машин с общей памятью можно выделить такие разработки как Polaris[1], CAPO[2], WPP[3], SUIF[4], VAST/Parallel[5], OSCAR[6], Intel/OpenMP[7]. Они позволяют получать ускорения времени выполнения программ в 10-20 раз на современных ЭВМ с общей памятью.

При использовании кластера можно получать ускорения в сотни раз. Однако автоматизация распараллеливания для кластеров сложнее. Наиболее важное место среди автоматизированных средств занимает разработка ParaWise[8]. Она базируется на первом подходе к распараллеливанию – в ней подразумевается постоянное участие пользователя на всех стадиях работы системы: анализ зависимостей, коррекция типов циклов, способы распределения данных,

* Работа выполнялась в рамках научно-технической программы Союзного государства «Развитие и внедрение в государствах-участниках Союзного государства наукоёмких компьютерных технологий на базе мультипроцессорных вычислительных систем» («ТРИАДА»).
Работа поддержана также грантом Президента РФ № НШ-383.2006.9 для ведущих научных школ и грантом РФФИ № 07-07-00221.

организации коммуникаций и генерация кода. Некоторый опыт в автоматизации распараллеливания программ на кластер накопился в разработках FORGE Magic/DM[9], BERT77[10], PARADIGM[11], OlyMPIx[12]. Отсутствие демонстрации результатов распараллеливания, например, в BERT77 и FORGE, не позволяет судить об эффективности их работы. Проекты PARADIGM и OlyMPIx демонстрируют ускорения в 2-5 раз на простых программах, объемом в несколько сотен строк. Активно ведутся исследования в Открытой распараллеливающей системе [13], в том числе и по распараллеливанию на кластер.

В настоящее время можно утверждать, что для кластеров не существует систем, которые достаточно эффективно распараллеливают программы на языках Fortran или C, и базируются на втором подходе – участие программиста ограничивается только указанием некоторых свойств программы.

Причинами отсутствия таких систем являются существенные трудности и проблемы, которые возникают при автоматическом отображении на кластер:

- огромное количество возможных вариантов распределения данных;
- сложность оценки эффективности этих вариантов.

2. Подход к автоматическому отображению программ на кластер

Целью данного исследования являлось построение алгоритмов автоматического отображения программ на кластер. Предложенные авторами алгоритмы реализованы в Системе Автоматизированной Параллелизации ФОРтран-программ (САПФОР). Эта система разрабатывается совместными усилиями трех коллективов: коллектив разработчиков системы V-Ray [14] (руководитель Вл.В.Воеводин, НИВЦ МГУ), коллектив разработчиков системы НОРМА [15] (руководитель К.Н.Ефимкин, ИПМ им. М.В.Келдыша РАН) и коллектив разработчиков DVM-системы [16] (руководитель В.А.Крюков, ИПМ им. М.В.Келдыша РАН). В системе САПФОР преобразование исходной программы в программу на языке параллельного программирования осуществляется с минимальным участием программиста. Программист может дополнять информацию о свойствах исходной программы и может изменять исходную программу для повышения ее параллелизма. Всю остальную работу система делает автоматически.

Алгоритмы автоматического отображения программ на кластер опираются на результаты анализа программ, а сами методы анализа программ не являлись целью данного исследования. Мы считаем, что основную работу по анализу исходной программы можно провести автоматически, а недостающую информацию о свойствах программы может предоставить программист.

Для организации параллельного выполнения программы необходимо в результате анализа выяснить следующее:

- какие есть в программе зависимости – они могут быть между витками одного цикла или между частями программы. Для участков программы, между которыми есть зависимость, необходимо обеспечить определенный порядок выполнения и коммуникации, чтобы результат их выполнения был корректным. Иногда этот порядок сводится к последовательному выполнению.
- какие переменные являются приватными (зависимость по которым не мешает выполнять цикл параллельно, если использовать несколько экземпляров таких переменных). Поиск приватных переменных проводится посредством анализа графа управления, и, в общем случае, может требовать значительного времени.
- какие зависимости являются редукционными (зависимость по которым не мешает выполнять цикл параллельно, если использовать несколько экземпляров таких переменных и производить объединение их значений на выходе из цикла). Например, когда в цикле вычисляется сумма некоторых элементов.

При выяснении этих свойств возникают следующие проблемы:

- при наличии в программе процедур анализ зависимостей становится гораздо сложнее;
- косвенная индексация (при индексации элемента массива используется переменная, значение которой меняется) и сложные индексы (например, i*i или i+j, где i и j – переменные цикла) затрудняют статический анализ (по тексту программы) и снижают его точность, делая анализ более консервативным;

- если вместо статического анализа использовать динамический анализ (в ходе выполнения программы), то возникает проблема длительного времени анализа и нехватки памяти, а также вопрос выбора представительного набора входных данных.

Опираясь на неточный анализ, трудно получить эффективную программу для кластера. Для уточнения анализа требуется помощь программиста.

Множество программ, на которые ориентирована текущая версия системы САПФОР, ограничено – эти программы должны удовлетворять следующим требованиям:

- программы написаны на языке Fortran 77;
- не содержат процедуры или допускают их инлайн-подстановку;
- в обращениях к элементам массивов используются индексы, которые линейно зависят от одной итерационной переменной;
- границы циклов и их шаги вычисляемы через подстановку именованных констант, объявленных в программе или заданные программистом.

Таким требованиям удовлетворяют, например, программы вычислений на статических регулярных сетках. Если индексы или параметры циклов не удовлетворяют вышеуказанным требованиям, то это скажется на эффективности распараллеливания, но не приведет к отказу от распараллеливания.

Предлагаемый в текущей версии системы САПФОР подход к распараллеливанию таких программ основывается на следующих решениях:

- использовать для выявления свойств программы статический анализ;
- дополнение или коррекцию программистом результатов анализа оформлять в виде спецкомментариев, вставляемых в текст исходной программы;
- существенно сокращать количество вариантов распределения данных путем применения эвристик;
- распределение вычислений проводить на уровне распределения витков циклов по процессорам (не использовать параллельные секции или подзадачи);
- для каждого варианта распределения данных строить один вариант отображения каждого цикла программы;
- искать для каждого варианта распределения данных оптимальную решетку процессоров (необходимо для сравнения вариантов многомерных распределений данных);
- оценивать варианты путем упрощенного моделирования параллельного выполнения программы и выбирать лучший – с минимальным спрогнозированным временем выполнения.

Применение оптимизаций и более сложных способов организации параллельного выполнения может рассматриваться как область для отдельного исследования. Но и без этого в предложенном подходе есть решения, требующие пояснений и обоснования их принятия:

- существенно сокращать количество вариантов распределения данных путем применения эвристик;
- для каждого варианта распределения данных строить один вариант отображения каждого цикла программы;
- искать для каждого варианта распределения данных оптимальную решетку процессоров;
- оценивать варианты путем упрощенного моделирования параллельного выполнения программы.

Эффективность того или иного варианта распределения данных определяется способом организации распараллеливания вычислений. В данном подходе предлагается использовать только распределение витков циклов по процессорам, причем каждый виток цикла выполняется целиком на некотором процессоре. Для эффективной реализации такого распараллеливания необходимо, чтобы все элементы массивов, в которые на данном витке надо произвести присваивания, были распределены на тот процессор, который будет выполнять этот виток. Тогда можно еще до выполнения цикла распределить его витки по процессорам и не потребуется перераспределение данных для выполнения необходимых присваиваний этого цикла.

Для выполнения присваивания необходимо иметь также и значения элементов массивов, используемых в данном витке на чтение. Эти элементы могут быть распределены либо на процессор, выполняющий виток, либо на другой процессор, и тогда потребуется обеспечить к ним

удаленный доступ. Эффективным является передача всех необходимых данных для всех процессоров заблаговременно, если это возможно.

Таким образом, эффективное распределение витков циклов накладывает условия на взаимное распределение массивов. Эти условия двух видов. Первое требует, чтобы все элементы массивов, в которые на витке присваиваются значения, находились на одном процессоре. Иначе цикл не будет распараллелен, и каждый процессор будет выполнять все его витки, проверять все обращения к массивам на запись, и проводить присваивания только в свои элементы массивов. Такой способ выполнения цикла неэффективен. Второе условие связано с тем, что надо минимизировать доступ к удаленным данным – попытаться распределить данные, необходимые для присваиваний, на процессор, выполняющий данный виток цикла.

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

Покажем, что количество вариантов распределения данных очень велико. Пусть сумма всех измерений у всех массивов равна Nd, тогда если строить варианты только по правилу: распределять ли очередное измерение или не распределять, – то количество вариантов будет 2^{Nd}. При использовании многомерных распределений получается, что для каждого измерения массива нужно указать на какое измерение процессорной решетки его распределить. Поэтому если размерность процессорной решетки равна K, то каждое измерение можно либо не распределять, либо распределить не более чем K способами, итого K+1 способ. А общее количество таких вариантов не более $(K+1)^{Nd}$. Если использовать неодинаковое распределение измерений массивов по процессорам (например, со смещениями), то количество вариантов будет существенно больше, чем $(K+1)^{Nd}$. В любом случае количество вариантов огромно, и необходимо применение эвристик для его сокращения.

В данном подходе предлагается сокращать количество вариантов распределения данных путем связывания измерений массивов в группы. И тогда варианты распределения данных ищутся не посредством перебора различных способов распределения измерений массивов по измерениям процессорной решетки, а путем перебора различных способов распределения групп измерений массивов. Измерения массивов внутри одной группы распределяются на одно и то же измерение процессорной решетки. При объединении массивов в группы применяется еще и такая эвристика: не добавлять измерение массива ни в одну группу, если элементы этого измерения не индексируются по итерационным переменным ни одного цикла. В этом случае нет смысла распределять такое измерение, поскольку нельзя соответственным образом распределить витки какого-либо цикла. Такая ситуация часто возникает в программах на языке Fortran 77, поскольку там нет другого способа объединить разные физические величины, относящиеся к одной точке пространства. Отметим, что эти две эвристики приводят к существенному сокращению количества вариантов распределения данных.

Например, в тестах NAS (LU, BT и SP) Nd изменяется от 64 до 76. Размерность процессорной решетки K, равная максимальной размерности массивов, изменяется от 4 до 6. А при оптимальном объединении в группы, получается 5-6 групп измерений, что позволяет уменьшить количество вариантов с 7^{76} до 2^6.

Второе важное решение в подходе к автоматическому отображению на кластер заключается в уменьшении суммарного объема накладных расходов на организацию всех параллельных циклов программы. Дело в том, что некоторые параллельные циклы могут оказаться внутри непараллельных циклов. В таком случае каждый параллельный цикл будет организовываться заново на каждом витке охватывающего его цикла. Для уменьшения суммарного объема таких расходов надо отдавать предпочтение распараллеливанию внешних циклов.

Предыдущие эвристики позволяют сократить количество построенных вариантов распределения данных и вычислений, но необходимо найти среди них лучший вариант, а для этого более точно оценить эффективность каждого из них. Для этого используется упрощенное моделирование параллельного выполнения программы, которое основано на анализе текста программы. Например, моделируется не каждый виток цикла, а сразу весь цикл. Конечно, по тексту программы сложно понять, сколько времени будут выполняться ее части, но маловероятно,

что погрешность в определении доли каждой части программы приведет к выбору плохого варианта распараллеливания.

Возможно использование профилировщика для оценки времен выполнения отдельных частей программы, но тогда возникают проблемы сокращения расходов времени и памяти, а также выбора представительного набора входных данных.

Различные варианты распределения данных и вычислений используют решетки процессоров разной размерности, а требуется сравнение эффективности вариантов между собой. Поэтому варианты следует сравнивать не на одинаковых процессорных решетках, а на одинаковом числе процессоров. Для каждого числа процессоров нужно находить оптимальные решетки процессоров, на которых время выполнения программы будет минимальным. Поскольку при поиске оптимальной решетки моделируется одна и та же программа, то целесообразно использовать некоторые закономерности, которые при этом возникают. На основе этих закономерностей можно сформулировать эвристики, которые позволили бы сократить область перебора различных решеток, и сделать поиск оптимальной решетки быстрее. В качестве одной такой эвристики предлагается использовать тот факт, что при увеличении числа процессоров по какому-то измерению решетки некоторые процессоры вообще не имеют элементов распределенных массивов. Можно считать, что такое распределение данных и, как следствие, распределение вычислений, проигрывают более равномерному распределению. В качестве второй такой эвристики можно использовать то обстоятельство, что если при увеличении количества процессоров по какому-то одному измерению процессорной решетки, время выполнения программы начинает увеличиваться, то нет смысла дальше увеличивать по этому измерению количество процессоров. Это связано с тем, что при росте процессоров, начиная с некоторого момента, потери из-за коммуникаций начинают возрастать быстрее, чем выигрыш от распараллеливания. Первая эвристика позволяет отбросить некоторые варианты решеток еще до их рассмотрения, а вторая работает после моделирования программы на некотором количестве решеток, т.е. работает во время поиска оптимальной решетки.

3. Реализация алгоритмов автоматического отображения на кластер

Предложенные авторами алгоритмы и эвристики были реализованы в системе САПФОР. Хотя эта система является автоматизированной и подразумевает участие программиста, но компонента системы, осуществляющая собственно отображение исходной программы на кластер (эксперт), работает в автоматическом режиме. Участие программиста может потребоваться при анализе исходной программы для указания некоторых свойств программы, например размеров циклов и массивов, в тех случаях, когда затруднительно их извлечь автоматически методами статического анализа.

3.1 Состав системы САПФОР

Текущая версия системы распараллеливания состоит из 3 типов компонент:

1. Анализаторы
2. Эксперты
3. База данных (БД)

Анализатор получает на вход исходную Fortran-программу, строит по ней ее структуру, анализирует программу на предмет наличия в ней параллелизма. В конце своей работы анализатор записывает результаты анализа и структуру программы в таблицы БД системы. Некоторые данные представлены в БД в удобном для последующей работы виде, например, выражения записываются по возможности в разобранном виде. В текущей версии системы используется статический анализатор ВерБа [15], но допускается использование нескольких анализаторов с тем, чтобы можно было дополнять БД более точными результатами анализа. Методы анализа могут отличаться (статические или динамические), и алгоритмы, используемые при анализе, могут по-разному справляться с некоторыми частями программы. Некоторые алгоритмы могут быть по сравнению с другими алгоритмами более быстрыми, но более грубыми в определении того, есть ли в программе какое-то свойство или его нет.

Экспертом называется программная компонента системы, знающая и использующая согласно своему алгоритму специфику конкретного языка параллельного программирования. Допускается использование нескольких альтернативных экспертов для того, чтобы сравнивать их преимущества и недостатки. Эксперты могут отличаться не только своими алгоритмами, но и разными языками программирования. Эксперт извлекает из БД необходимую ему информацию о структуре программы и результаты анализа, строит различные схемы распараллеливания для исходной программы, оценивает построенные схемы по некоторым критериям и строит тексты программ на языках параллельного программирования. Под *схемой распараллеливания* понимается набор правил преобразования текста исходной программы. Правила могут быть нескольких типов: вставка специальных комментариев, вставка и удаление описаний переменных, вставка и удаление операторов.

База данных хранит всю необходимую информацию для всех компонент системы, тем самым, обеспечивая их взаимодействие. Там хранятся структура программы, результаты анализа, указания пользователя (заданные в тексте исходной программы в виде спецкомментариев или). Для каждой исходной программы пользователя (состоящей из одного или нескольких файлов) создается отдельная БД.

3.2 Сценарий использования системы САПФОР

Сначала исходный текст Fortran-программы подается на вход анализатору. Он записывает результаты анализа в БД. При желании пользователя и наличии нескольких анализаторов можно уточнять результаты анализа.

Полученная БД подается на вход одному из экспертов. Он находит схемы распараллеливания исходной программы и строит тексты программ на языках параллельного программирования. Дальнейшую компиляцию, выполнение этих программ и исследование их эффективности можно проводить на ЭВМ, которые поддерживают работу с этими языками программирования.

Если эффективность полученной программы не устраивает пользователя, то он может вставить в текст исходной программы указания по корректировке результатов анализа, либо более существенно изменить свою программу, и подать измененную программу на вход анализатору.

3.3 Особенности алгоритмов отображения программ на кластер в DVM-эксперте

Среди языков параллельного программирования для кластеров выделяется своей распространенностью стандарт MPI. Программирование вычислительных задач на MPI, и особенно распараллеливание уже имеющихся программ, является непростым делом, которое требует не столько выделение в программе параллельных и последовательных участков, сколько организацию параллельного выполнения в виде процессов. С появлением многоядерных и многопоточных процессоров все более ощущается потребность в параллельных языках более высокого уровня.

Одним из таких языков высокого уровня является язык Fortran-DVM [16]. Поскольку в результате компиляции программа на языке Fortran-DVM преобразуется в программу на стандартном языке Fortran, выполняющуюся в узлах кластера и использующую библиотеку MPI для организации взаимодействия между процессорами, то программирование на языке Fortran-DVM может рассматриваться как более простой способ написания MPI-программ. Компилятор с языка Fortran-DVM применяет готовые приемы распараллеливания и освобождает программиста от рутинной работы, приводящей к многочисленным ошибкам распараллеливания. По этой причине в качестве языка параллельного программирования при отображении исходных Fortran-программ на кластер был выбран язык Fortran-DVM. К тому же для этого языка есть развитые средства функциональной отладки и отладки эффективности, что упрощает поиск ошибок распараллеливания, есть готовая библиотека прогнозирования времени выполнения параллельной программы, которая необходима для оценки вариантов распараллеливания. В состав разработчиков системы САПФОР входят и разработчики DVM-системы, что позволяет развивать язык Fortran-DVM по мере расширения класса входных для распараллеливания программ.

В системе САПФОР компонента, использующая при распараллеливании язык Fortran-DVM, называется DVM-экспертом. Эксперт распараллеливает исходную программу путем вставки DVM-директив. Полученную программу можно скомпилировать на любой машине, где установлена DVM-система, выполнить ее там, получить результаты, получить характеристики эффективности параллельного выполнения, а также убедиться в правильности полученных результатов при помощи средств отладки DVM-системы (динамический контроль и сравнительная отладка).

Особенность текущей реализации DVM-эксперта заключается в том, что входная программа должна состоять из одной программной единицы. Это обеспечивается при помощи инлайн-подстановки, которая подставляет все процедуры программы в текст головной процедуры, и исходный текст программы преобразуется в требуемый вид. Далее анализатор и эксперт работают с этой программой.

Использование предложенных авторами эвристик, реализованных в алгоритмах работы DVM-эксперта, позволяет сократить число схем распараллеливания и получить результаты распараллеливания за приемлемое время.

При построении схем DVM-эксперт не разбирает сложные выражения (отличные от a*i+b, где a и b – целочисленные константы, а i – индексная переменная цикла) и поэтому принимает грубые решения по распараллеливанию программ с наличием таких выражений. Например, пересылка всего распределенного массива всем процессорам.

Эксперт учитывает имеющиеся ограничения языка Fortran-DVM, например, на использование распределенных массивов в операторах ввода-вывода, что влияет на его решения при распараллеливании программы.

Для группировки измерений массивов используется граф. Дуги в графе отображают связь двух измерений каких-либо массивов между собой. Каждая дуга состоит из двух выражений (при каждой из вершин) и веса. Выражения соответствуют выражениям, которые использовались при индексации элементов массивов в программе. Дуги строятся для пар использований массивов в одном цикле. Вес назначается дуге по некоторым правилам в зависимости от того, используется ли переменная на запись или только на чтение. Вес отображает временные задержки, которые возникнут, если распределение данных в программе не будет соответствовать соотношению выражений при дуге (т.е. будет им противоречить).

Временные задержки могут быть двух видов:

1. Задержки из-за дублирования вычислений и проверки правила собственных вычислений для непараллеленных циклов. Тогда на всех витках цикла каждый процессор проверяет, находится ли на нем данный элемент массива, и только для своих элементов производит в них присваивания. Такие задержки замедляют весь цикл на время, сравнимое со временем его выполнения. Если данный цикл вызывается несколько раз (например, вложен в еще один цикл), то задержки надо оценивать с учетом количества вызовов данного цикла. Поэтому Вес = TLoop*Q, где TLoop - время выполнения всего цикла, а Q - количество вызовов данного цикла.
2. Задержки из-за пересылок удаленных данных с одного процессора на другой. В худшем случае всем процессорам пересылается весь массив, используемый на чтение. Вес = (Tstart+N*Tbyte)*Q, где Tstart и Tbyte – характеристики текущей машины, N – число байт в массиве, Q - количество повторений цикла

Если дуга связывает два массива, используемые на запись, то ее вес будет отражать временную задержку первого вида.

Для остальных трех комбинаций пар массивов – вес будет отражать временную задержку второго вида.

Вычисление по таким правилам веса дуги является эвристикой. Эта эвристика опирается на модель коммуникаций и на оценки замедления выполнения циклов в случае дублирования вычислений и вставки проверок на наличие у данного процессора элемента массива. Вес дуги будет определять, какие дуги стоит рассматривать, а какие следует игнорировать, что повлияет на получающиеся варианты распределения данных. В DVM-эксперте используется модель коммуникации, которая задается двумя величинами: латентностью и накладными расходами на передачу каждого последующего байта.

После построения графа массивов производится удаление в нем циклов (замкнутых путей). Цикл может быть противоречивым (есть два пути от одной вершины цикла до другой, и они отражают разные соотношения между двумя вершинами графа), так и непротиворечивым. В любом случае, цикл устраняется путем удаления в нем дуги с наименьшим весом. Это является эвристикой (удаление дуг, не изменяя вес других дуг) наравне с вычислением весов для всех дуг. Циклы удаляются по мере увеличения количества дуг в них, начиная с 2-х дуг и заканчивая 2*(N-1) дуг, где N – число вершин в графе.

Также противоречием считается путь, связывающий две вершины одного массива. Это противоречие устраняется путем удаления дуги с наименьшим весом среди дуг этого пути. После удаления циклов и противоречий весь граф сам разбивается на группы вершин. По этим группам строятся различные варианты распределения данных.

Для каждого варианта распределения данных, согласно принятому подходу, строится один вариант распределения вычислений. Предпочтение отдается распараллеливанию внешних циклов.

Оценка схем распараллеливания в DVM-эксперте строится на основе моделирования параллельного выполнения программы на кластере с заданными параметрами, если к исходной Fortran-программе применить данную схему распараллеливания. Для этого используются алгоритмы моделирования, использующиеся и в DVM-предикторе [17] для прогнозирования характеристик выполнения программ по трассе их запуска на одном или нескольких процессорах. Но в отличие от работы [17], оценка схем использует не трассу запуска программы, а проводится по тексту программы. Результаты моделирования представлены в виде спрогнозированных временных характеристик параллельного выполнения всей программы и ее частей. В них представлено время выполнения, затраты на коммуникации, время работы процессора, времена, потерянные из-за дублирования вычислений, времена простоя процессоров и другие детальные характеристики. Они позволяют проанализировать эффективность программы, точнее и лучше понять причины, повлиявшие на ее эффективность.

Основными объектами моделирования являются циклы программы. Причем непараллельные циклы обрабатываются следующим образом: выполняется первый виток (внутри него могут быть параллельные и непараллельные циклы), запоминаются текущие значения всех характеристик, выполняется второй виток и вычисляется разность, которая говорит о характеристиках, получаемых при каждом последующем витке непараллельного цикла. Эти характеристики умножаются на недостающее количество витков и складываются с характеристиками первого витка. Это позволяет быстрее моделировать параллельное выполнение программы и делать это достаточно точно.

4. Результаты

Система автоматизированного распараллеливания САПФОР, в которой был реализован предложенный подход, была успешно проверена на ряде простых тестов, проверяющих различные механизмы распараллеливания программ. Также были получены результаты по распараллеливанию более крупных программ. Среди них тест NAS LU [18] и программа трехмерного моделирования сферического взрыва во внешнем магнитном поле с помощью решения уравнений идеальной магнитной гидродинамики [19].

4.1 Тест LU

LU (Lower-Upper Solver) – нахождение конечно-разностного решения 3-х мерной системы уравнений Навье-Стокса для сжимаемой жидкости или газа с равномерно разреженной блочной треугольной матрицей 5x5. Применяется метод LU-разложения, с использованием алгоритма SSOR (метод верхней релаксации). Использовался класс C (162x162x162 и 250 итераций). Текст программы был получен из DVM-версии этого теста путем удаления всех директив DVM. Для полученного текста была проведена инлайн-подстановка процедур в тело головной подпрограммы. После этого текст программы содержал 3641 строки, 30 массивов (из них 4 массива пятимерной размерности) и 424 цикла. При помощи специальных указаний пользователя было специфицировано наличие редукционной зависимости по элементам массивов в теле 3 циклов.

С нахождением такой редукционной зависимости текущая версия статического анализатора не справлялась.

DVM-эксперт построил 16 схем распараллеливания и сгенерировал программу, времена выполнения которой на кластере МВС-100К [20] представлены в таблице 1. Там же содержатся времена выполнения на МВС-100К варианта программы, полученного путем ручного распараллеливания программы LU разработчиками DVM-системы. В таблице 2 приведена информация о времени работы системы и ее компонент.

Таблица 1. Время выполнения вариантов программы LU на МВС-100К (в секундах).

Варианты	Число процессоров 1	Число процессоров 8	Число процессоров 256	Число процессоров 1024
САПФОР	3482.40	1009.49	40.33	25.55
Ручной	2103.14	858.26	34.99	19.97

Таблица 2. Время работы компонент системы для распараллеливания программы LU (в секундах и в процентах от общего времени системы)

	Анализатор	Эксперт	Итого
Время	1082	96	1178
Проценты	91,85%	8,15%	100%

Полученная разница во временах выполнения программы LU на 1 процессоре объясняется тем, что для работы системы САПФОР была произведена подстановка процедур, которая повлияла на оптимизацию кода. Отметим, что в среднем эффективность написанных вручную DVM-версий тестов NAS, составляет около 80% от эффективности MPI-версий этих программ [21].

4.2 Программа mhpdv

Это программа трехмерного моделирования сферического взрыва во внешнем магнитном поле с помощью решения уравнений идеальной магнитной гидродинамики. Fortran-версия этой программы содержит 1809 строк, 33 массива (из них 11 массивов четырехмерной размерности) и 116 циклов.

Для нее DVM-эксперт также построил 16 схем распараллеливания и сгенерировал программу, времена выполнения которой на кластере МВС-100К представлены в таблице 3. Там же содержатся времена выполнения на МВС-100К варианта программы, полученного путем ручного распараллеливания программы mhpdv разработчиками DVM-системы. В таблице 4 приведена информация о времени работы системы и ее компонент.

Таблица 3. Время выполнения вариантов программы mhpdv на МВС-100К (в секундах).

Варианты	Число процессоров 1	Число процессоров 8	Число процессоров 256	Число процессоров 1024
САПФОР	3703.23	500.78	34.75	12,78
Ручной	3574.29	486.74	32.15	10.98

Таблица 4. Время работы компонент системы для распараллеливания программы mhpdv (в секундах и в процентах от общего времени системы)

	Анализатор	Эксперт	Итого
Время	113	41	154
Проценты	73,38%	26,62%	100%

5. Заключение

Автоматическое отображение Fortran-программ на кластер вполне возможно, если при их написании придерживаться определенной дисциплины и предоставить программисту средства для спецификации некоторых свойств его программы, недоступных для статического анализа.

В результате апробации системы САПФОР на двух программах среднего размера были получены программы с характеристиками эффективности, сравнимыми с характеристиками программ, распараллеленных вручную. Необходимо отметить, что для автоматического отображения были взяты программы, полученные из ранее созданных DVM-программ посредством удаления из них всех DVM-директив, и которые не содержали вызовов процедур.

Тем не менее, можно уверенно говорить о получении обнадеживающих результатов, которые позволяют уверенно говорить о возможности автоматического отображения на кластер и многих других программ. Преодолены принципиальные трудности – трудности статического анализа компенсируются подсказками программиста, найдены эвристики для сокращения количества вариантов распараллеливания. Автоматическое распараллеливание на языки DVM-системы осуществлять гораздо легче, чем в случае непосредственного использования MPI (в системе DVM-компиляторы используют уже готовые приемы распараллеливания, имеются развитые средства функциональной отладки и отладки эффективности, есть готовая библиотека прогнозирования характеристик выполнения DVM-программ), к тому же возможности DVM-системы можно расширять по мере расширения класса входных для распараллеливания программ.

Для практического использования системы автоматизированного распараллеливания САПФОР необходимо ее дальнейшее развитие в двух направлениях – расширении входного языка Fortran 77 до языка Fortran 90 и снятие ограничений на использование процедур.

Литература

1. Polaris Developer's Document: [http://polaris.cs.uiuc.edu/polaris/polaris_developer/polaris_developer.html], 21.04.1997.
2. CAPO (Computer-Aided Parallelizer and Optimizer): [http://people.nas.nasa.gov/~hjin/CAPO/index.html], 20.10.2002.
3. Satoh M., Aoki Y., Wada K., Iitsuka T., Kikuchi S., Interprocedural parallelizing compiler WPP and analysis information visualization tool Aivi: [www.compunity.org/events/pastevents/ewomp2000/MakotoPaper.pdf], 2000.
4. The Stanford SUIF Compiler Group: [http://suif.stanford.edu/].
5. VAST/Parallel: [http://www.crescentbaysoftware.com/vast_parallel.html].
6. Kasahara H., Obata M., Ishizaka K.. Automatic coarse grain task parallel processing on smp using openmp. // Proc. of 13 th International Workshop on Languages and Compilers for Parallel Computing 2000, Aug. 2000.
7. Intel OpenMP: [http://www.intel.com/software/products/compilers/flin/docs/main_for/mergedprojects/copts_for/common_options/option_openmp.htm].
8. Система ParaWise: [www.parallelsp.com].
9. Applied Parallel Research. FORGE Magic/DM: [http://wotug.ukc.ac.uk/parallel/vendors/apr/ProductInfo/dpf_datasheet.txt], 1993.
10. BERT 77: Automatic and Efficient Parallelizer for FORTRAN: [http://www.basement-supercomputing.com/content/view/24/50/], 20.10.2006.
11. PARADIGM: A Parallelizing Compiler for Distributed Memory Message-Passing Multicomputers: [http://www.ece.northwestern.edu/cpdc/Paradigm/Paradigm.html].
12. Vikram K., Avijit K., Aggarwal S.K. OlyMPIx – A Program Parallelization Tool using MPI on Computational Grids: [www.cs.cornell.edu/~kvikram/papers/pdcn.ps].
13. Открытая распараллеливающая система: [www.ops.rsu.ru].
14. Система V-Ray: [http://v-ray.parallel.ru].

15. Баранова Т.П., Вершубский В.Ю., Ефимкин К.Н., Федосимов В.А. Развитие возможностей анализатора последовательных программ. // Труды Всероссийской научной конференции “Научный сервис в сети Интернет: решение больших задач”, сентябрь 2008, г. Новороссийск. -М.: Изд-во МГУ, 2008. –С. 38-42.

16. DVM система: [www.keldysh.ru/dvm].

17. Клинов М.С., Крюков В.А. Прогнозирование характеристик параллельного выполнения DVM-программ. // Труды Всероссийской научной конференции “Научный сервис в сети Интернет: технологии параллельного программирования”, сентябрь 2006, г. Новороссийск. –М.: Изд-во МГУ, 2006. –С. 113-114.

18. The NAS Parallel Benchmarks: [http://science.nas.nasa.gov].

19. Устюгов С. Д., Чечеткин В. М. Взрыв сверхновой при крупномасштабной конвективной неустойчивости вращающейся протонейтронной звезды. // Астрономический журнал, 1999, том 76, №11, с. 816-824.

20. Суперкомпьютер "МВС-100К": [http://www.jscc.ru/hard/mvs100k.shtml], 24.09.2008.

21. Крюков В.А. Разработка параллельных программ для вычислительных кластеров и сетей. // Информационные технологии и вычислительные системы. –2003. –N. 1-2. –С. 42-61.