МАТЕМАТИЧЕСКОЕ МОДЕЛИРОВАНИЕ ОПТИМАЛЬНОЕ УПРАВЛЕНИЕ

УДК 519.685.1

АВТОМАТИЧЕСКОЕ РАСПАРАЛЛЕЛИВАНИЕ ФОРТРАН-ПРОГРАММ. ОТОБРАЖЕНИЕ НА КЛАСТЕР

М.С. Клинов, В.А. Крюков

Институт прикладной математики им. М.В. Келдыша РАН, Москва

klinov@keldysh.ru

Поступила в редакцию 30.12.2008

Рассматриваются алгоритмы автоматического отображения на кластер программ на языке Фортран. Алгоритмы используют результаты 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, расположенным на других процессорах) и их оценке путем моделирования параллельного выполнения соответствующих им программ.

Ключевые слова: высокопроизводительные вычисления, параллельные вычисления, автоматическое распараллеливание, автоматизированное распараллеливание, кластер.

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 быстрее решать задачи, 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 отлаживать программы на этих языках сложнее. Это часто заставляет программистов сначала написать и отладить программы без использования параллелизма, а потом их распараллеливать. Существуют целые программные комплексы, которые тоже требуют распараллеливания для эффективного выполнения на новых ЭВМ. Процесс распараллеливания отлаженной 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:

- использование диалога с программистом и для уточнения свойств последовательной программы, и для выбора наилучших решений по ее распараллеливанию;
- использование диалога 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 рассмотреть 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 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 система ParaWise [8]. Она базируется на первом подходе к распарал-

леливанию – в ней подразумевается постоянное участие программиста на всех стадиях работы системы: анализ зависимостей, коррекция типов циклов, способы распределения данных, организации коммуникаций и генерация кода. Некоторый опыт в области 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.

1. 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 зависимости есть в программе – 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 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 путем 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 трудно отследить) и сложные индексные выражения (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;
- программы не содержат процедуры или допускают их инлайн-подстановку;
- в обращениях к элементам массивов используются индексные выражения, которые линейно зависят от одной итерационной переменной;
- границы циклов и их шаги вычисляемы через подстановку именованных констант, объявленных в программе или заданных программистом.

Таким требованиям удовлетворяют, например, программы вычислений на статических регулярных сетках. Если индексные выражения 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 программ из описанного множества 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;
- оценивать варианты распараллеливания программы путем упрощенного моделирования параллельного выполнения и выбирать лучший – с минимальным прогнозируемым временем выполнения на заданном числе процессоров;
- при поиске минимального времени выполнения варианта программы использовать эвристические методы сокращения области перебора.

Применение оптимизаций и более сложных способов организации параллельного выполнения является областью 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;
- оценивать варианты распараллеливания программы путем упрощенного моделирования параллельного выполнения;
- использовать эвристики поиска минимального времени выполнения варианта распараллеливания программы на заданном числе процессоров.

2. Обоснование принятых в подходе решений

Эффективность того или иного варианта распределения данных определяется способом последующего распараллеливания вычислений. В данном подходе предлагается использовать только распределение витков циклов по процессорам, причем каждый виток цикла выполняется целиком на некотором процессоре. Это накладывает два вида условий на взаимное распределение массивов. Первое требует, чтобы все элементы массивов, в которые на витке присваиваются значения, находились на одном процессоре. Иначе цикл не будет распараллелен, 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 элементов по процессорам, а процессоры тогда организуются логически в многомерную решетку. 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 по всем массивам равна Nd, тогда если строить варианты только по правилу: распределять ли очередное измерение или не распределять, то количество вариантов будет 2^{Nd}. При использовании многомерных распре-

делений 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. В таком случае варианты распределения данных будут строиться не посредством перебора различных способов распределения измерений массивов по измерениям процессорной решетки, а путем перебора различных способов распределения групп измерений массивов. Измерения массивов одной группы будут распределяться на одно и то же измерение процессорной решетки.

Связывание измерений массивов в группы происходит по некоторым оценкам (стремимся эффективнее распараллелить какой-то цикл или уменьшить коммуникации при выполнении другого цикла), которые в силу своей неточности становятся эвристическими. Также при построении групп применяется 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 соответствующим 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 эвристики приводят к существенному сокращению количества вариантов распределения данных. Например, в тестах NAS [17] (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 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 стремлении уменьшить суммарный объем накладных расходов на организацию выполнения 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 вариантов распределения данных и вычислений при их построении, но необходимо также найти среди них лучший вариант. Для оценки эффективности используется упрощенное моделирование параллельного выполнения программы. При этом время выполнения последовательных участков программы может определяться при статическом анализе программы. И хотя по тексту программы невозможно точно оценить, сколько времени будут выполняться ее части, но маловероятно, что погрешность в определении доли каждой части программы приведет к выбору плохого варианта распараллеливания. Конечно, для оценки времен выполнения отдельных частей программы возможно использование профилировщика, но тогда возникают проблемы сокращения расходов времени и памяти, а также выбора представительного набора входных данных.

Различные варианты распараллеливания рассчитаны на использование решеток процессоров разной размерности. Поэтому для каждого варианта и заданного числа процессоров нужно находить оптимальную решетку процессоров, на которой время выполнения программы будет минимальным. При поиске возникают некоторые закономерности, на их основе можно сформулировать эвристики, которые позволяют сократить область перебора и делают 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 не имеют элементов распределенных массивов (например, если количество процессоров больше количества элементов). Можно считать, что такое распределение данных и, как следствие, распределение вычислений, будет менее эффективным, чем более равномерное распределение, но использующее меньшее количество процессоров. В качестве второй эвристики 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 числа процессоров начиная с некоторого момента потери из-за коммуникаций начинают возрастать быстрее, чем выигрыш от распараллеливания. Первая эвристика позволяет отбросить некоторые варианты решеток еще до их рассмотрения, а вторая используется после моделирования программы на некотором количестве решеток, т.е. во время поиска оптимальной решетки.

3. Текущая реализация системы САПФОР

Предложенные авторами алгоритмы и эвристики были реализованы в системе САПФОР. Хотя эта система является автоматизированной и подразумевает участие программиста, но компонент системы, осуществляющий собственно отображение исходной программы на кластер, работает в автоматическом режиме. В текущей версии системы САПФОР используется статический анализатор ВерБа [18], но допускается использование нескольких анализаторов (статические или динамические) для получения более точных результатов анализа.

Важнейшим решением при проектировании системы САПФОР был выбор языка параллельного программирования. Среди языков параллельного программирования для кластеров выделяются распространенностью языки стандарта MPI. Программирование вычислительных задач на MPI, и особенно распараллеливание уже имеющихся программ, является непростым. С появлением многоядерных и многопоточных процессоров все более ощущается потребность в параллельных языках более высокого уровня.

Одним из таких языков является Fortran-DVM [16]. Программа на этом языке в результате компиляции преобразуется в программу на стандартном языке Fortran, которая выполняется в узлах кластера и использует библиотеку MPI для организации взаимодействия между процессорами. Программирование на 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. Это было основной причиной выбора Fortran-DVM в качестве языка параллельного программирования для автоматического отображения исходных Fortran-программ на кластер. К тому же для этого языка есть развитые средства функциональной отладки (динамический контроль и сравнительная отладка) и отладки эффективности, что упрощает поиск ошибок распараллеливания, есть готовая библиотека прогнозирования времени выполнения параллельной программы [19], которая необходима для оценки вариантов распараллеливания. Поскольку разработчики DVM-системы входят в состав разработчиков системы САПФОР, то возможности DVM можно расширять по мере необходимости.

4. Результаты

Система автоматизированного распараллеливания САПФОР, в которой был реализован предложенный подход, была успешно проверена на ряде простых тестов. Также были получены результаты по распараллеливанию более крупных программ. Среди них тест NAS LU [17] и программа трехмерного моделирования сферического взрыва во внешнем магнитном поле с помощью решения уравнений идеальной магнитной гидродинамики [20].

4.1. Программа LU. LU (Lower-Upper Solver) – нахождение конечно-разностного решения 3-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 (162x162x x162 и 250 итераций). Текст программы был получен из DVM-версии этого теста путем удаления всех директив DVM. Для полученного текста была произведена инлайн-подстановка процедур в тело головной подпрограммы. После этого текст программы содержал 3641 строку, 30 массивов (из них 4 массива пятимерной размерности) и 424 цикла. При помощи специальных указаний программиста 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 вариантов распараллеливания, выбрал лучший и для него сгенерировал программу, времена выполнения которой на кластере МВС-100К [21] 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 САПФОР и ее компонентов.

Таблица 1

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

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

Таблица 2

Время работы компонентов 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%

Таблица 3

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

Варианты	1 процессор	8 процессоров	256 процессоров	1024 процессора
САПФОР	3703.23	500.78	34.75	12.78
Ручной	3574.29	486.74	32.15	10.98

Таблица 4

Время работы компонентов 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%

Полученная разница во временах выполнения программы LU на одном 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 [22].

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 вариантов распараллеливания и для лучшего варианта 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 САПФОР и ее компонентов.

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, которые не содержали вызовов процедур.

Тем не менее получены обнадеживающие результаты, 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 (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 развивать по мере расширения класса распараллеливаемых программ.

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

Работа выполнялась в рамках научно-технической программы Союзного государства «Развитие и внедрение в государствах-участниках Союзного

государства наукоемких 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.

Список литературы

1. Polaris Developer's Document. – URL: http:// polaris.cs.uiuc.edu/polaris/polaris_developer/polaris_developer.html (дата обращения: 26.01.2009).

2. CAPO (Computer-Aided Parallelizer and Optimizer): сайт. – URL: http://people.nas.nasa.gov/~hjin/CAPO/index.html (дата обращения: 26.01.2009).

3. Satoh M., Aoki Y., Wada K. et al. Interprocedural parallelizing compiler WPP and analysis information visualization tool Aivi. URL: http://www.compunity.org/events/pastevents/ewomp2000/MakotoPaper.pdf (дата обращения: 26.01.2009).

4. The Stanford SUIF Compiler Group: сайт. – URL: http://suif.stanford.edu (дата обращения: 26.01.2009).

5. VAST/Parallel. – URL: http://www.crescent-bay-software.com/vast_parallel.html (дата обращения: 26.01.2009).

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, Aug. 2000.

7. Intel OpenMP. – URL: http://www.intel.com/software/products/compilers/flin/docs/main_for/mergedprojects/copts_for/common_options/option_openmp.htm (дата обращения: 26.01.2009).

8. Система ParaWise: сайт. – URL: http://www.parallelsp.com (дата обращения: 26.01.2009).

9. Applied Parallel Research. FORGE Magic/DM. – URL: http://wotug.ukc.ac.uk/parallel/vendors/apr/ProductInfo/dpf_datasheet.txt (дата обращения: 26.01.2009).

10. BERT 77: Automatic and Efficient Parallelizer for FORTRAN. сайт. – URL: http://www.basement-supercomputing.com/content/view/24/50/ (дата обращения: 26.01.2009).

11. PARADIGM: A Parallelizing Compiler for Distributed Memory Message-Passing Multicomputers: сайт. – URL: http://www.ece.northwestern.edu/cpdc/Paradigm/Paradigm.html. (дата обращения: 26.01.2009).

12. Vikram K., Avijit K., Aggarwal S.K. OlyMPIx – A Program Parallelization Tool using MPI on Computational Grids. – URL: http://www.cs.cornell.edu/~kvikram/papers/pdcn.ps (дата обращения: 26.01.2009).

13. Открытая распараллеливающая система: сайт. – URL: http://www.ops.rsu.ru (дата обращения: 26.01.2009).

14. Система V-Ray: сайт. – URL: http://v-ray.parallel.ru (дата обращения: 26.01.2009).

15. Система НОРМА: сайт.- URL: http://keldysh.ru/pages/norma (дата обращения 3.02.2009).

16. DVM система: сайт. – URL: http://www.keldysh.ru/dvm (дата обращения: 26.01.2009).

17. The NAS Parallel Benchmarks: сайт. – URL: http://www.nas.nasa.gov (дата обращения: 26.01.2009).

18. Баранова Т.П., Вершубский В.Ю., Ефимкин К.Н., Федосимов В.А. Развитие возможностей анализатора последовательных программ // Труды Всерос. науч. конф. «Научный сервис в сети Интернет: решение больших задач», Новороссийск, 22–27 сентября 2008. М.: Изд-во МГУ, 2008. С. 38–42.

19. Клинов М.С., Крюков В.А. Прогнозирование характеристик параллельного выполнения DVM-программ // Труды Всерос. науч. конф. «Научный сервис в сети Интернет: технологии параллельного программирования», Новороссийск, 18–23 сентября 2006 г. М.: Изд-во МГУ, 2006. С. 113–114.

20. Устюгов С.Д., Чечеткин В.М. Взрыв сверхновой при крупномасштабной конвективной неустойчивости вращающейся протонейтронной звезды // Астрономический журнал. 1999. Т. 76. № 11. С. 816–824.

21. Суперкомпьютер «МВС-100К»: сайт. – URL: http://www.jscc.ru/hard/mvs100k.shtml (дата обращения: 26.01.2009).

22. Крюков В.А. Разработка параллельных программ для вычислительных кластеров и сетей // Информационные технологии и вычислительные системы. 2003. № 1–2. С. 42–61.

AUTOMATIC PARALLELIZATION OF FORTRAN PROGRAMS. MAPPING TO CLUSTER

M.S. Klinov, V.A. Kryukov

The algorithms of automatic mapping of Fortran programs to cluster are considered. The algorithms use the results of the program static analysis, which, if necessary, can be corrected or supplemented by a programmer. Automatic mapping to cluster is based on the construction of various parallelization variants (data distribution and computation distribution between processors, as well as providing access to remote data on other processors) and their evaluation by modeling the parallel execution of their respective programs.

Keywords: high-performance computing, parallel computing, automatic parallelization, cluster.