А. С. Антонов

Технологические аспекты адаптации последовательных программ для массивно-параллельных компьютеров

Введение

Всё более широкое распространение массивно-параллельных супер-ЭВМ и перспективы дальнейшего развития этого направления заставляют программистов разрабатывать новые технологии создания программ для компьютеров с подобной архитектурой, а накопленный за многие годы большой багаж последовательных программ определяет необходимость разработки методов адаптации уже существующих программ на принципиально новую архитектуру. Поскольку как те, так и другие методы развиты явно недостаточно для разработки больших и сложных программ, эффективное использование массивно-параллельных супер-ЭВМ является до сих пор очень трудоёмким делом. Данная статья на основе опыта адаптации программы FLO52 из пакета Perfect Club для супер-ЭВМ CRAY T3D демонстрирует возникшие на этом пути трудности и предлагает некоторые способы их преодоления.

Массивно-параллельная супер-ЭВМ (CRAY T3D, Intel Paragon, IBM SP2 и др.) представляет из себя набор однотипных процессорных элементов (ПЭ) с распределённой памятью, соединённых коммуникационной средой. Конечно, на применяемые в том или ином случае методы определённое влияние может оказать и оказывает сама конкретная архитектура, а именно, характеристики ПЭ и коммуникационной среды, однако общий подход остаётся один и тот же.

Будем предполагать, что имеется $nproc$ ПЭ, которые нумеруются от 1 до $nproc$, и, возможно, HOST-машина, которая имеет номер 0. Мы будем рассматривать только SPMD-модель (Single Program Multiple Data) параллельного программирования, поэтому распределение работ производится только по номеру ПЭ. Пересылки данных осуществляются с помощью функций типа *send* и *receive*, которые предполагают, что отсылающий ПЭ вызывает функцию *send* и продолжает выполнение программы независимо от того, готов ли к приёму другой ПЭ, а принимающий ПЭ вызывает функцию *receive* и в случае отсутствия сообщения блокируется до его прихода.

Целью преобразования программы является минимизация времени её исполнения. Для этого необходимо использовать оптимальное (заметим, что не всегда максимальное) число ПЭ, максимально загружать каждый из используемых ПЭ и минимизировать количество пересылок между ПЭ. Число процессоров ограничивается константой $nproc$, но не совсем ясно, всегда ли необходимо использовать все доступные ПЭ; оптимизация загрузки и пересылок достигается при помощи разумного распределения работ между ПЭ. Работы надо распределять так, чтобы по возможности равномерно загружать все ПЭ, а также чтобы на каждом ПЭ действия выполнялись по возможности над одним и тем же и, желательно, независимым от других ПЭ набором данных. Если при выполнении программы на каком-то из ПЭ модифицируются некоторые данные, которые позднее (по тексту последовательной программы) будут использоваться на других ПЭ, то эти данные должны быть разосланы всем таким ПЭ, причём эти ПЭ должны блокироваться до прихода этих данных.

Чаще, проще и логичнее всего выделяются в отдельные цепочки вычислений (которые затем распределяются между ПЭ) итерации циклов программы, между которыми нет зависимостей. Распределение итераций отдельного цикла нужно производить исходя из тех же соображений равномерной загруженности и минимизации числа пересылок. Первой цели надо пытаться достичь при помощи анализа тела самого цикла (в простейшем случае распределять итерации можно равными порциями), а для достижения второй цели необходим анализ контекста — на каких ПЭ ранее вычислялись данные, используемые на той или иной итерации, а, возможно, и на каких ПЭ будут использоваться далее. Для решения этой задачи чаще всего оказывается недостаточным изучение текста той подпрограммы, в которой встретился данный цикл, поэтому распределение итераций любого цикла нужно проводить с учётом анализа всей программы. Распределение данных по процессорным элементам должно быть согласовано с распределением соответствующих итераций так, чтобы обеспечивать их правильное выполнение.

1 Технология адаптации последовательных программ

1.1 Общий подход

Все действия по адаптации последовательной программы на массивно-параллельную архитектуру можно разделить на два этапа: этап анализа программы и этап её преобразования. На этапе анализа необходимо составить модель параллельного выполнения программы со всеми необходимыми коммуникациями между ПЭ. На основе этой модели (реально существующей в виде графа или же абстрактной) осуществляется второй этап. Необходимость разработки чёткой последовательности действий на этапе преобразования программы следует из того, что объём программы и объём необходимых изменений может быть очень большим, а поиск ошибок в параллельной программе — очень трудоёмкое дело. Это определяет необходимость разработки такой технологии преобразований, которая обеспечивала бы эквивалентность получаемых в ходе работы вариантов исходной программе. Кроме того, может оказаться важным уже на раннем этапе определить, насколько эффективно реализуется на данной архитектуре разработанный на основе параллельной модели программы алгоритм её выполнения.

1.2 Этап анализа программы

1.2.1 Последовательность проводимого анализа

Анализ программы подразделяется на статический анализ, то есть анализ только текста программы без её выполнения, и анализ динамики её исполнения на одном процессоре массивно-параллельной супер-ЭВМ. Поскольку доступ на массивно-параллельные компьютеры является и долго ещё будет оставаться ограниченным и дорогим, важно как можно большую часть анализа проводить статически.

1.2.2 Статический анализ

Сначала в программе необходимо каким-то образом выделить наиболее важные циклы и гнёзда циклов, то есть циклы, на выполнение которых затрачивается наибольшее время. Это можно сделать, анализируя граф вызовов подпрограмм, циклические профили подпрограмм и заголовки циклов (хотя в заголовках циклов часто используются символьные параметры, значения которых неизвестны, — это затрудняет анализ). Выделение таких циклов, правда, не гарантирует, что, распределив их итерации даже оптимальным образом, мы достигнем поставленной цели, так как возможно, например, наличие в программе большого числа "менее важных" циклов, для выполнения каждого из которых придётся делать огромное количество пересылок. Однако, в первую очередь необходимо выполнить анализ наиболее важных циклов, так как без их эффективной реализации невозможна успешная адаптация всей программы.

Далее для всех выделенных циклов и гнёзд циклов необходимо определить возможность параллельного выполнения их итераций. Такая возможность определяется отсутствием зависимости по данным между итерациями данного цикла. Для осуществления такой проверки хорошим средством является разработанная в лаборатории Нелинейных вычислений НИВЦ МГУ система анализа программ V-ray Analyser & Adviser.

Часто отдельный цикл из гнезда циклов соответствует перевычислению одного измерения некоторых вырезок из массивов. В этом случае кажется правильным распределять в первую очередь тот цикл из гнезда, который:

- соответствует измерениям массивов, перевычисляемым в наибольшем числе наиболее важных циклов;
- не имеет зависимостей между итерациями;
- требует наименьшего количества пересылок данных;
- состоит из возможно большего количества итераций.

Затем нужно выбрать метод распределения итераций по выбранной размерности циклов так, чтобы минимизировать количество пересылок внутри соответствующих подпрограмм. Для этого надо выделять такие группы итераций, внутри которых сосредоточено наибольшее число зависимостей по данным, так, чтобы при разбиении итераций цикла на эти группы разрывалось наименьшее количество связей и, следовательно, требовалось наименьшее число пересылок. Например, если внутри цикла существует зависимость между итерациями с шагом k, то наиболее естественно помещать на один ПЭ данные для итераций, отстоящих на k. При этом потребуются пересылки между ПЭ только для первой или последней (соответственно с направлением зависимости) итерации из такой группы.

Перед преобразованием последовательной программы под массивно-параллельную архитектуру необходимо для каждой распределяемой подпрограммы составить списки импорта — описания наборов входных данных, необходимых на каждой итерации, экспорта — изменяемых на каждой итерации распределяемой размерности глобальных данных и необходимых внутри данной подпрограммы пересылок.

В качестве первых двух списков в простейшем случае можно взять все параметры и все данные, используемые в COMMON-блоках данной подпрограммы. Однако это невыгодно, так как объём этих пересылок может стать огромным. Оба списка можно значительно сократить с помощью статического анализа тела подпрограммы. Из списка импорта надо исключить объекты, не встречающиеся в правых частях операторов присваивания и в управляющих операторах, а из списка экспорта — объекты, не встречающиеся в левых частях операторов присваивания. Далее, в списках импорта и экспорта надо оставить лишь те вырезки из массивов, которые соответствуют итерациям, попадающим на данный ПЭ по выбранной схеме распределения итераций. Кроме того, методами межпроцедурного анализа можно определить объекты, изменяемые данной подпрограммой, но значения которых не используются в других подпрограммах (такое может быть, если, например, массив, входящий в COMMON-блок, используется во всех подпрограммах только как рабочий). Оставшиеся после этих действий списки будут являться списками импорта-экспорта данной подпрограммы. Если в составлении списка импорта возможно указать больше данных, чем реально нужно подпрограмме, и это не приведёт к ошибкам, то лишние данные, попавшие в список экспорта, могут затереть некоторые полезные данные, что приведёт к трудноотлавливаемым ошибкам.

Пересылки внутри данной подпрограммы необходимы при наличии в графе алгоритма данной подпрограммы дуг, соединяющих гиперплоскости, соответствующие двум разным итерациям распределяемой размерности, в случае если эти итерации при выбранной схеме распределения итераций попадают на разные ПЭ. Это относится как к зависимостям итераций одного цикла, так и к зависимостям между итерациями разных циклов. Как показывает практика, часто требуются дополнительные пересылки граничных элементов массивов, если, конечно, при выбранном методе распределения итераций они не попадут на нужный ПЭ. Кроме того, пересылки могут потребоваться при изменении одним или несколькими ПЭ некоторых общих данных, таких, например, как переменные, определяющие условия или параметры циклов.

1.2.3 Анализ динамики исполнения программы

Для уточнения полученного на этапе статического анализа списка наиболее важных подпрограмм и гнёзд циклов полезно провести трассировку исходного варианта программы, работающего на одном ПЭ массивно-параллельной супер-ЭВМ или же на каком-либо другом процессоре с похожей архитектурой. Для этого надо замерить время выполнения каждого вызова каждой подпрограммы и просуммировать его по всем вызовам, включая или не включая времена выполнения вызываемых из данной подпрограммы подпрограмм. При более детальной трассировке можно получить подобную информацию для отдельных циклов, фрагментов и даже отдельных операторов. Всё это позволяет уточнить и дополнить картину потенциального параллелизма программы и значимости отдельных фрагментов.

Анализ динамики выполнения программы позволяет получить диапазоны изменения параметров циклов и значения условных выражений. Эти значения могут во многом поменять представления о схеме необходимых преобразований. Если число итераций выделенного цикла много меньше числа ПЭ, то возможно разбиение всего набора ПЭ на группы, в каждой из которых значение управляющего параметра данного цикла одинаково, а затем распределять ещё один цикл между ПЭ каждой группы. Если же, наоборот, число итераций много больше числа ПЭ, то возможно распределение итераций цикла между ПЭ. Если же число итераций близко к числу ПЭ, то возможна недозагрузка некоторых ПЭ. Эти рассуждения, правда, относятся к случаю, когда итерации примерно равноценны по времени выполнения, либо же когда время выполнения разных итераций (а, точнее, соотношение времён) заранее неизвестно и его не удаётся оценить. Если же возможна оценка соотношений времён выполнения разных итераций, то необходимо ещё пытаться сгруппировать итерации так, чтобы равномерно загрузить все ПЭ.

1.3 Этап преобразования программы

После определения метода распределения итераций выделенных циклов необходимо приступить к последовательному распределению их итераций с учётом получаемого при этом ускорения.

1.3.1 Две схемы адаптации подпрограмм

Для последовательной адаптации подпрограмм с распределяемыми итерациями под массивно-параллельную архитектуру предлагаются две схемы действий:

- в самом начале порождается $nproc$ процессов, каждый из которых независимо выполняет всю программу до вызова соответствующей подпрограммы, исполняется подпрограмма с распределёнными между ПЭ итерациями, после чего необходимо данные, вычисленные на каждом ПЭ, передать всем остальным ПЭ для продолжения автономной работы;

- изначально работает только один процесс, например, на HOST-машине или на некотором выделенном ПЭ, перед каждым вызовом соответствующей подпрограммы инициируются $nproc$ процессов, которым пересылаются необходимые данные, исполняется подпрограмма с распределёнными между ПЭ итерациями, после чего необходимо данные, вычисленные на каждом ПЭ, передать HOST-машине или выделенному ПЭ и приостановить все остальные процессы.

Преимуществом первой схемы является относительная простота действий и необходимость пересылать только результаты выполнения подпрограммы. Но при этом надо оградить ввод-вывод, чтобы не было попыток сразу нескольких процессов работать с одним файлом, а также для работы тестовых вариантов требуются пересылки от одного ПЭ всем остальным, что надо делать очень осторожно, чтобы предотвратить превышение пропускной способности коммуникационной сети. При этом не надо заботиться о необходимых для выполнения подпрограммы данных, так как они продублированы на каждом ПЭ.

Вторая схема предполагает пересылку данных как перед вызовом подпрограммы, так и после него, но зато не возникает проблем с вводом-выводом и пересылки требуются только между HOST-машиной (или выделенным ПЭ) и ПЭ, а также уменьшается общий объём пересылок за счёт того, что не надо пересылать от каждого ПЭ каждому. Кроме того, при использовании второй схемы программа может исполняться почти целиком на HOST-машине, что быстрее, чем на ПЭ.

В каждом конкретном случае та или иная схема может оказаться более выгодной. При относительно небольшом количестве входных данных, необходимых для распределяемой между ПЭ подпрограммы, и большом количестве операций ввода-вывода в рассматриваемой программе может оказаться более выгодной вторая схема, в противном случае — первая. Мы в своих экспериментах использовали обе, но после тщательного исследования их достоинств и недостатков остановились на первой.

1.3.2 Адаптация подпрограмм с вызовами

Последовательное распределение подпрограмм внутри данной программы предлагается проводить “снизу вверх”. После распределения всех вызываемых подпрограмм надо распределить саму вызывающую подпрограмму. После этого исчезают все накладные расходы на пересылку данных до и после выполнения для вызываемых подпрограмм[1], но требуются аналогичные пересылки для данной подпрограммы, а также могут появиться дополнительные пересылки, необходимость которых определяется на этапе межпроцедурного анализа. Эти пересылки появляются из-за того, что возможно изменение некоторых данных для одной из вызываемых подпрограмм в другой. Двигаясь подобным образом, можно распределить всю программу. Эффективность полученной программы зависит от качества проводимого анализа, правильности сделанных предположений и самой структуры алгоритма, которая может оказаться столь плохой, что не позволит эффективно использовать преимущества массивно-параллельной супер-ЭВМ.

[1] Надо позаботиться лишь о том, чтобы, если нужно, скорректировать списки импорта и экспорта подпрограммы в соответствии со списками вызываемых подпрограмм.

1.3.3 Оценка эффекта от распределения

Для оценки эффекта от распределения той или иной подпрограммы необходимо сравнить время выполнения исходного варианта программы на одном процессоре массивно-параллельной машины и распределённого варианта. Если времена для исходных текстов подпрограмм получены ранее на этапе трассировки программы, то остаётся получить времена для всех вариантов распределённых подпрограмм и сравнить их. Не приходится рассчитывать на то, что использование *nproc* ПЭ даст ускорение в близкое к *nproc* число раз, поэтому нельзя заранее сказать, какой результат надо считать хорошим. От специфики задачи сильно зависит, насколько хорошо она будет распараллеливаться, но по картине параллелизма программы, построенной на этапе анализа, всегда можно сказать, весь ли резерв параллелизма мы использовали.

2 Эксперименты с программой FLO52

Эксперименты проводились на компьютере CRAY T3D, имеющем 256 ПЭ. Каждый ПЭ представляет из себя процессор DEC ALPHA с локальной памятью в 8 Мслов. Для организации пересылок использовалась библиотека, реализованная на основе функций нижнего уровня, обеспечивающая порождение параллельных процессов, пересылки типа *send-receive*, синхронизацию, подсчёт глобальных сумм, поиск глобального минимума и максимума и другие возможности.

2.1 Анализ программы

Для определения структуры программы широко использовались как статический анализ, так и анализ динамики исполнения программы. В дальнейшем результаты применения этих методов не разделяются, чтобы дать более полную картину программы.

2.1.1 Общая структура программы

Программа FLO52 из пакета Perfect Club Benchmarks была выбрана в качестве модельной для проведения экспериментов по отображению последовательных программ на массивно-параллельную архитектуру. Данная программа существует в двух вариантах: в малом, получаемом по умолчанию, и в большом, получаемом с помощью подстановки больших значений размерностей вычисляемых массивов. Кроме этого с помощью аналогичной подстановки ещё больших размерностей массивов был получен ещё один вариант. В дальнейшем эти три варианта называются соответственно “малый”, “средний” и “большой”. Поскольку схема параллельного взаимодействия для всех вариантов одна и та же, а ресурс параллелизма у больших вариантов больше, с ростом размерности решаемой задачи должно вырасти и получаемое ускорение. Все эксперименты производились над малым вариантом, требующим меньше времени на выполнение и меньший объём оперативной памяти, после чего из получившейся программы аналогичной подстановкой были получены распределённые версии для больших вариантов. Результаты по отдельным подпрограммам приводятся для среднего варианта, а окончательные результаты — по всем трём вариантам.

Программа не требует слишком большого объёма памяти и времени вычисления, поэтому существует возможность выполнения исходного варианта FLO52 на одном процессоре машины T3D без каких-либо модификаций. При этом время исполнения составляет 59.8 секунды для малого варианта, 235.6 секунды для среднего и 999.3 секунды для большого. Такая возможность позволяет легко оценить ускорение, полученное при выполнении данной программы на нескольких процессорах, а также позволяет не заботиться о переописании массивов данной программы, распределяемых между ПЭ.

Главной вычислительной частью в данной программе является подпрограмма EULER, из которой в цикле вызываются наиболее важные подпрограммы. Как показал анализ динамики исполнения программы, на выполнение подпрограммы EULER вместе со всеми вызываемыми из неё подпрограммами на одном ПЭ приходится около 88% времени исполнения FLO52. Этим подпрограммам и было уделено первоочередное внимание. Подпрограмма EULER и вызываемые из неё подпрограммы имеют в основном достаточно хорошую структуру для распределения по ПЭ. Почти все циклы внутри этих подпрограмм имеют независимые итерации и одинаковые границы, а большинство гнёзд циклов — одинаковую структуру. Это заметно облегчает распределение их итераций.

В достаточно большом числе подпрограмм данной программы встречается ввод-вывод, распределение которого является скорее задачей разработчиков аппаратуры, а не программистов. Поэтому распределение таких подпрограмм не производилось. На эти подпрограммы суммарно приходится около 4% времени исполнения всей программы (в малом варианте), что является одним из сдерживающих факторов при распределении всей программы. Более того, две подпрограммы, осуществляющие вывод значений распределённых массивов, пришлось переделать так, чтобы вывод в файл осуществлялся только одним ПЭ, для чего ему необходимо переслать данные со всех остальных ПЭ.

Другими сдерживающими факторами являются подсчёт глобальных сумм, глобальных максимумов и минимумов, а также подпрограмма PSMOO, две части которой требуют разного распределения данных: первая — последовательная по измерению I с возможностью распределения по J, вторая — последовательная по J с возможностью распределения по I. Кроме того, во многих подпрограммах требуется достаточно большое количество относительно небольших пересылок граничных значений массивов. Поскольку в таких пересылках участвуют только ПЭ, на которые попадают граничные итерации, эти ПЭ заметно сдерживают все остальные.

Подпрограммы COLLC и ADDX осуществляют перераспределение данных между ПЭ при переходе от большей размерности к меньшей (подпрограмма COLLC) и наоборот (ADDX). Пересылки в этих подпрограммах оказались весьма сложными и нерегулярными и также явились сдерживающим фактором при распределении всей программы.

2.1.2 Выбор размерности для распределения

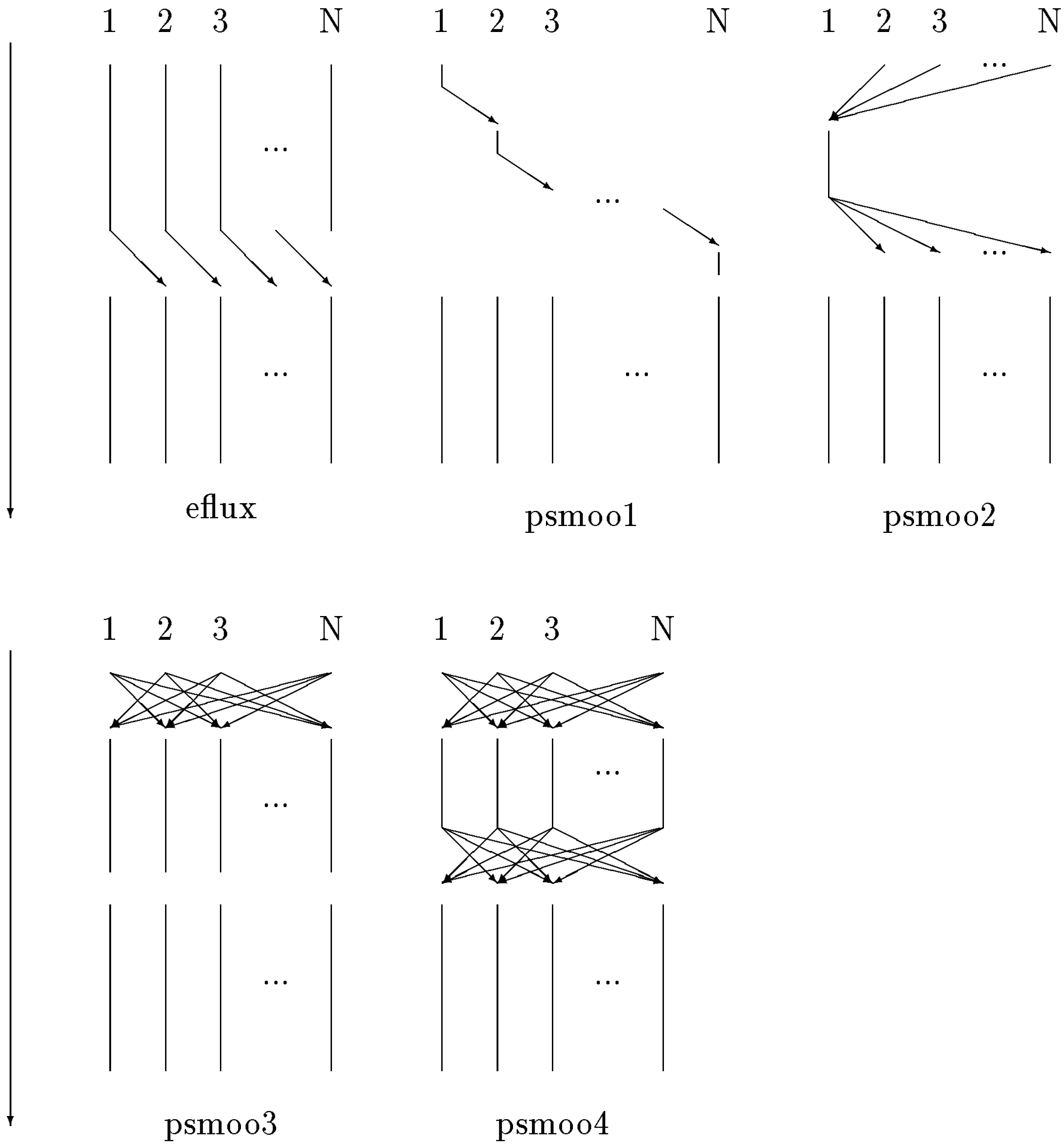
Исходя из перечисленных во второй части требований, надо выбрать размерность для первоочередного распределения соответствующих ей итераций между ПЭ. Цикл по J сразу отбрасывается — для него потребовалось бы слишком большое число пересылок, поскольку во многих местах на J-й итерации требуются данные, вычисленные ранее на других итерациях, да и выясненное экспериментально число итераций слишком мало — от 5 до 33 (малый вариант). Цикл по N хорош тем, что при распределении его итераций не потребуется вообще никаких пересылок данных, но он очень короткий — всего 4 итерации, поэтому после него пришлось бы распределять ещё один цикл, что сильно усложнило бы программу; кроме того, не во все циклические гнёзда этой программы входит цикл по N. Поэтому самым подходящим для распределения выглядит цикл по I. Количество итераций этого цикла меняется от 21 до 161 (в малом варианте), а пересылки по этой размерности требуются в относительно немногих местах. Конечно, при распределении итераций только цикла по I мы используем не весь ресурс параллелизма, заложенный в программе, но дополнительное распределение итераций циклов по J (и, может быть, по N) привело бы к усложнению требуемых пересылок, что сделало бы значительно более трудоёмким процесс адаптации данной программы.

2.1.3 Выбор метода распределения итераций

Метод распределения итераций в данном случае достаточно очевиден — большинство пересылок осуществляется между соседними итерациями, поэтому наиболее естественным кажется помещать на один ПЭ как можно больше последовательных итераций. Было решено распределять итерации по возможности равными последовательными порциями между всеми ПЭ. Количество итераций цикла по I делится на количество ПЭ, и каждый ПЭ получает порцию из такого количества последовательных итераций или на одну больше, если при делении получается ненулевой остаток. При этом нескольким последним ПЭ может не достаться ни одной итерации, но зато минимизируется количество пересылок, а время исполнения определяется тем же количеством итераций, которое получилось бы при попытке частично загрузить последние ПЭ за счёт предыдущих. Кроме последних, в некоторых случаях, например, когда итерации цикла начинаются не с 1, несколько недозагруженным может оказаться первый ПЭ. Однако такая недозагрузка, естественно, предпочтительнее перераспределения данных для достижения более равномерной загруженности.

2.2 Последовательное распределение основных подпрограмм

Результаты последовательного распределения рассматриваемых ниже подпрограмм приведены в следующем пункте, здесь же обсуждаются проводившиеся преобразования и встретившиеся трудности. Следующий рисунок демонстрирует схемы взаимодействия процессоров, встретившиеся в обсуждаемых ниже подпрограммах.



Вертикальные линии на рисунке показывают последовательные вычисления на соответствующем ПЭ, стрелки изображают необходимые пересылки, оси показывают направления последовательности вычислений.

Распределение итераций подпрограмм по выбранному измерению было начато с подпрограммы EFLUX. На выполнение на одном ПЭ этой небольшой подпрограммы, состоящей из пяти циклических гнёзд, тратится около 28% времени исполнения всей программы. Для распределения циклов по I в данной подпрограмме необходимы пересылки части массива с последней итерации, попавшей на некоторый ПЭ, на следующий ПЭ (см. рисунок). После выполнения подпрограммы необходимо вычисляемые в ней на каждом ПЭ результаты переслать всем остальным ПЭ. Это же относится и ко всем рассматриваемым далее подпрограммам. Поскольку внутри данной подпрограммы только в одном месте требуются относительно небольшие по объёму пересылки, на данной подпрограмме удаётся достичь достаточно хорошего ускорения.

В подпрограмме DFLUX дополнительно к уже описанным пересылкам добавляются многочисленные пересылки граничных значений массивов и пересылки части данных, вычисленных на I-й итерации, на I−2-ю итерацию. Поскольку подпрограмма DFLUX требует почти такого же суммарного времени исполнения, как и EFLUX, хорошее её распределение не менее важно, чем распределение EFLUX, несмотря на сложности, связанные с большим объёмом подпрограммы и большим количеством требуемых пересылок. Ускорение, полученное на данной подпрограмме, сравнимо с ускорением, полученным на подпрограмме EFLUX.

Намного сложнее ситуация обстоит с подпрограммой PSMOO. Делалось несколько попыток её эффективной реализации, поскольку уже после первых экспериментов стало ясно, что эта подпрограмма во многом будет являться узким местом при распределении FLO52 из-за необходимости различного распределения данных для двух частей данной подпрограммы.

Сначала предпринимались попытки реализовать PSMOO, не осуществляя перераспределения данных между ПЭ. Это предполагает, что итерации первой половины подпрограммы будут исполняться строго последовательно, а распределены между ПЭ будут только итерации второй половины. При этом была надежда, что некоторая экономия на пересылках компенсирует проигрыш, полученный за счёт потери части параллелизма.

Первый вариант (psmoo1) исполнялся строго в соответствии с алгоритмом, то есть каждый ПЭ просчитывал свою порцию итераций, передавал данные с последней итерации следующему ПЭ, после чего тот ПЭ начинал работать (см. рисунок). Однако с ростом числа ПЭ растёт число необходимых пересылок, а, следовательно, и время исполнения PSMOO растёт катастрофически и становится определяющим фактором для времени исполнения всей программы.

Второй вариант (psmoo2) предусматривал исполнение всех итераций первой половины PSMOO на ПЭ с номером 1. В начале PSMOO все ПЭ отсылали первому свои порции исходных данных, а после исполнения первой половины подпрограммы этот ПЭ рассылал всем остальным их порции результатов (см. рисунок). При этом, как видно, требуется всего две, хотя и весьма больших, пересылки. Оказалось, что потеря параллелизма в первой половине подпрограммы и наличие пересылок больших объёмов данных не позволяют значительно улучшить ситуацию по сравнению с предыдущим вариантом.

Третий вариант (psmoo3) предусматривал выполнение всех итераций первой половины PSMOO на каждом ПЭ. Для этого требовалась единственная рассылка перед выполнением подпрограммы порций исходных данных от каждого ПЭ каждому (см. рисунок). Если на малом числе ПЭ этот вариант незначительно превосходит предыдущий, то на 128 и 256 ПЭ оказывается даже хуже его.

Четвёртый вариант предполагает перераспределение данных и соответствующих итераций так, чтобы использовать в первой части PSMOO параллелизм по J, а во второй — по I (psmoo4). Для этого до и после первой части требуется рассылка от каждого ПЭ каждому, но относительно небольшой части входных данных (см. рисунок). Данный вариант оказался в итоге наиболее удачным.

В пятом варианте была реализована параллельная схема вычисления рекурсии первого порядка, подобная схеме каскадного суммирования (psmoo5). Этот вариант отличается от всех предыдущих своей масштабируемостью, то есть тем, что время вычисления уменьшается с ростом числа ПЭ. Однако время вычисления полученного варианта на малом числе ПЭ оказалось неприемлемо большим, поэтому было принято решение в зависимости от числа ПЭ вызывать либо этот вариант PSMOO, либо предыдущий.

Подпрограмма FORCF вызывается всего 150 раз и занимает очень мало времени при последовательном выполнении. Однако в ней встретился принципиально новый момент — необходимость глобального суммирования по всем ПЭ. Сама операция не очень сложная, но реализовать её эффективно на компьютере с распределённой памятью не очень просто. Для данной операции использовалась библиотечная функция, оптимизированная специально под архитектуру CRAY T3D.

Подпрограммы BCFAR и BCWALL не поставили никаких принципиально новых проблем, за исключением того, что подпрограмма BCFAR содержит в себе вызов подпрограммы FORCF. В этом случае, однако, никаких дополнительных пересылок не потребовалось, так как результатом подпрограммы FORCF являются лишь три числа, вычисленных в глобальном суммировании, которые по смыслу операции уже находятся на всех ПЭ. Подпрограмма BCWALL не получила ускорения при увеличении числа ПЭ из-за наличия достаточно большого числа коротких пересылок граничных значений.

Следующий шаг в распределении программы заключается в распределении итераций подпрограммы EULER, так как к этому моменту распределены все вызываемые из неё подпрограммы. Успешное распределение подпрограммы EULER имеет первостепенное значение для получения хорошего ускорения всей программы. Анализ последовательности вызовов подпрограмм из подпрограммы EULER показывает, что при распределении подпрограммы EULER не требуется практически никаких дополнительных пересылок, связанных с передачей результатов выполнения вызываемых подпрограмм, так как единая для всех подпрограмм схема распределения итераций позволяет оставлять на ПЭ результаты I-й итерации, так как далее они потребуются только на этом же ПЭ. Требуются только пересылки, необходимые по тексту подпрограммы EULER.

В этот момент исчезают все накладные расходы на пересылки до и после вызываемых из подпрограммы EULER подпрограмм. Необходимы лишь пересылки после выполнения EULER данных, вычисленных в этой подпрограмме и во всех вызываемых из неё подпрограммах. За счёт этого уже в данный момент мы получили ускорение для всей программы. Кроме упомянутых пересылок в подпрограмме EULER требуется уже встречавшееся в подпрограмме FORCF глобальное суммирование, а также нахождение глобального максимума.

После распределения подпрограммы EULER со всеми вызываемыми из неё подпрограммами оказалось, что очень существенное влияние на общее время выполнения FLO52 оказывает рассылка данных от каждого ПЭ каждому после вызовов данной подпрограммы, появившаяся только из-за используемой последовательности действий по адаптации программы, причём время на эту рассылку быстро растёт с ростом числа ПЭ. Для дальнейшего ускорения FLO52 необходимо избавиться от этой рассылки, что возможно только при переносе выбранной схемы распределения на уровень всей программы.

Для этого оказалось достаточным правильно осуществить перераспределение в подпрограммах COLLC и ADDX и разобраться с выводом в выходные файлы.

В подпрограммах COLLC и ADDX требуются более сложные пересылки, чем встречавшиеся ранее, так как приходится пересылать данные с итераций, распределение которых производилось при одних значениях параметров, на итерации, распределение которых производится при других значениях параметров. Если в подпрограмме COLLC удалось получить некоторое ускорение, то подпрограмма ADDX оказалась ещё одним узким местом.

Остальные подпрограммы FLO52 требуют весьма малого времени исполнения либо занимаются вводом-выводом. Поэтому их распределение (даже когда оно возможно) потребует, возможно, значительных усилий, но не принесёт большого эффекта. Поэтому было решено этим подпрограммам внимания не уделять.

После получения распределённого варианта программы была произведена дополнительная оптимизация вычислений на каждом ПЭ, заключавшаяся в более эффективном использовании кэш-памяти, улучшении сбалансированности вычислений и в уменьшении размерностей временных массивов. В наибольшей степени результаты этой оптимизации сказались на малых количествах ПЭ, так как при этом на каждый ПЭ приходится достаточно большая порция итераций.

2.3 Результаты экспериментов

В следующей таблице приведены времена исполнения распределявшихся подпрограмм в зависимости от числа процессоров для среднего варианта. Время исполнения на одном процессоре указано для исходного последовательного варианта программы.

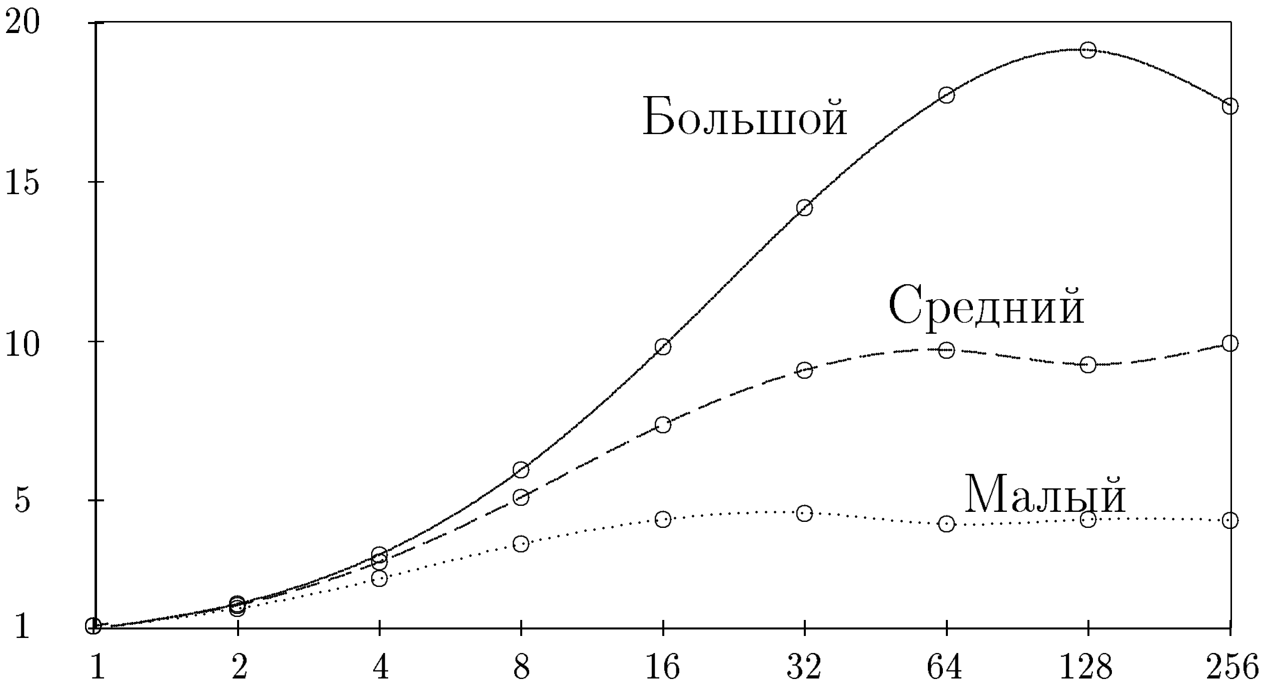
Времена выполнения подпрограмм, с.									
Средний вариант									
П/п	Количество ПЭ								
	1	2	4	8	16	32	64	128	256
eflux	68.7	38.2	19.3	10.2	5.74	3.49	2.32	1.69	1.41
dflux	53.6	29.0	15.2	8.27	4.83	3.11	2.20	1.76	1.57
psmoo1	27.5	31.0	28.8	29.9	34.5	43.2	58.6	81.4	107.6
psmoo2	27.5	35.7	32.9	34.2	39.5	41.7	44.2	50.3	58.2
psmoo3	27.5	30.3	30.3	33.9	37.1	39.7	44.5	51.3	65.6
psmoo4	27.5	22.6	14.2	9.23	7.15	7.03	8.69	12.0	15.0
psmoo5	27.5	166.7	—	—	41.96	—	15.91	10.25	8.22
forcf	0.054	0.031	0.018	0.012	0.009	0.008	0.008	0.008	0.008
bcfar	1.18	0.56	0.33	0.19	0.12	0.09	0.07	0.06	0.05
bcwall	1.85	1.83	1.51	1.29	1.18	1.12	1.10	1.09	1.08
euler	56.1	24.6	13.5	7.88	5.14	3.72	2.95	2.64	2.43
collc	3.93	2.43	1.61	1.43	1.42	1.05	0.73	0.70	0.65
addx	3.79	3.45	2.47	2.16	2.31	2.82	2.80	4.01	4.51

Как видно из таблицы, ускорение по сравнению с однопроцессорным вариантом получилось почти на всех рассмотренных подпрограммах, однако упоминавшиеся в предыдущем пункте узкие места действительно оказывают значительное сдерживающее влияние.

В следующей таблице приводятся окончательные результаты, полученные на всех трёх рассматриваемых вариантах.

Времена выполнения вариантов FLO52, с.									
Вариант	Количество ПЭ								
	1	2	4	8	16	32	64	128	256
Малый	59.7	37.2	23.4	16.4	13.6	13.0	14.0	13.6	13.7
Средний	235.6	136.3	76.5	46.1	31.9	25.9	24.3	25.4	23.7
Большой	999.3	566.7	301.8	167.0	101.5	70.3	56.3	52.2	57.4

Следующий график демонстрирует ускорение, полученное на всех трёх вариантах в зависимости от числа используемых ПЭ.



Как видно из графика и из приведённых результатов, полученная распределённая подпрограмма является масштабируемой и при достаточно большом объёме вычислений позволяет получать существенное ускорение при использовании большого числа ПЭ. Относительно небольшое ускорение, получаемое на малом и среднем вариантах, объясняется недостаточным размером задачи, а также большим числом точек параллельного взаимодействия процессоров (суммарно более 35000 за время работы программы). Ускорение, полученное на малом варианте при переходе от 64 к 128 ПЭ и на среднем варианте при переходе от 128 к 256 ПЭ, объясняется использованием в этих случаях варианта PSMOO, в котором реализуется параллельная рекурсия (psmoo5). Большой вариант оказался максимально возможным, который без существенных изменений может поместиться в локальную память одного ПЭ. Дальнейшего увеличения размерности задачи можно было бы достичь переописанием используемых массивов таким образом, чтобы на каждом ПЭ выделялось место лишь для той порции данных, которая на нём будет использоваться.

3 Заключение

Поскольку данная статья писалась лишь на основе опыта проведения экспериментов с программой FLO52, она не претендует на полноту охвата материала, то есть на разработку методики адаптации любой последовательной программы на массивно-параллельную архитектуру, и решение всего разнообразия встающих при этом перед программистом проблем. Поэтому за рамками этой статьи заведомо оказывается некоторое количество вопросов, таких как, например, явное разбиение массивов на части, отдаваемые отдельным ПЭ в случае, если размеры данных программы таковы, что они не помещаются в оперативную память одного ПЭ. Предполагается, что методика действий в подобных случаях будет разрабатываться по мере возникновения практической необходимости.

Литература

[1] Grassl C. M. *Parallel performance of applications on supercomputers*// Parallel Computing. 1991. N17. P.1257-1273.

[2] Blume W. J. *Success and limitations in automatic parallelization of the Perfect Benchmarks programs*// CSRD Tech. Rep. 1992. N1249.

[3] Cybenko G., Kipp L., Pointer L., Kuck D. *Supercomputer performance evaluation and the Perfect Benchmarks*// CSRD Tech. Rep. 1990. N965.

[4] Pointer L. *Perfect: Performance Evaluation for Cost-Effective Transformations Report 2*// CSRD Tech. Rep. 1990. N964.