РАЗРАБОТКА ПАРАЛЛЕЛЬНЫХ ПРОГРАММ ДЛЯ ВЫЧИСЛИТЕЛЬНЫХ КЛАСТЕРОВ И СЕТЕЙ[1].

В.А. Крюков
Институт прикладной математики им. М.В. Келдыша РАН
e-mail: krukov@keldysh.ru

Аннотация.

В работе делается сравнительный анализ четырех разных подходов к созданию параллельных программ для проведения научно-инженерных расчетов на вычислительных кластерах и сетях (MPI, HPF, OpenMP+MPI и DVM) со следующих позиций: легкость разработки и сопровождения параллельных программ, эффективность разработанных программ, переносимость и повторное использование программ.

1. Введение

Последние годы во всем мире происходит бурное внедрение вычислительных кластеров. Это вызвано тем, что кластеры стали общедоступными и дешевыми аппаратными платформами для высокопроизводительных вычислений. Одновременно резко возрос интерес к проблематике вычислительных сетей (GRID) и широко распространяется понимание того, что внедрение таких сетей будет иметь громадное влияние на развитие человеческого общества, сравнимое с влиянием на него появления в начале века единых электрических сетей [1]. Поэтому, рассматривая проблемы освоения кластеров необходимо принимать во внимание и то, что они являются первой ступенькой в создании таких вычислительных сетей.

Поскольку единого определения вычислительного кластера не существует, для упрощения дальнейшего изложения введем некоторую классификацию, которая будет отражать свойства программно-аппаратной платформы, существенные с точки зрения разработки прикладных параллельных программ.

Вычислительный кластер – это *мультикомпьютер*, состоящий из множества отдельных компьютеров (*узлов*), связанных между собой единой коммуникационной системой. Каждый узел имеет свою локальную оперативную память. При этом общей физической оперативной памяти для узлов не существует. Если в качестве узлов используются *мультипроцессоры* (мультипроцессорные компьютеры с общей памятью), то такой кластер называется SMP-кластером. Коммуникационная система обычно позволяет узлам взаимодействовать между собой только посредством передачи сообщений, но некоторые системы могут обеспечивать и односторонние коммуникации - позволять любому узлу выполнять массовый обмен информацией между своей памятью и локальной памятью любого другого узла.

Если все входящие в состав вычислительного кластера узлы имеют одну и ту же архитектуру и производительность, то мы имеем дело с *однородным* вычислительным кластером. Иначе – с *неоднородным*.

С точки зрения разработки прикладных параллельных программ нет каких-либо принципиальных различий между однородными кластерами и MPP, такими как IBM SP-2. Различие, в основном, заключается в большей доступности и меньшей стоимости кластеров по сравнению с мультипроцессорными ЭВМ с распределенной памятью, в которых используются специальные коммуникационные системы и специализированные узлы.

[1] Работа поддержана Российским фондом фундаментальных исследований, гранты № 02-01-00752 и 02-07-90027

В настоящее время, когда говорят о кластерах, то часто подразумевают однородность. Однако, для того, чтобы сохранить высокий уровень соотношения производительность/стоимость приходится при наращивании кластера использовать наиболее подходящие в данный момент процессоры, которые могут отличаться не только по производительности, но и по архитектуре. Поэтому постепенно большинство кластеров могут стать неоднородными кластерами.

Неоднородность же вносит следующие серьезные проблемы.

Различие в производительности процессоров требует соответствующего учета при распределении работы между процессами, выполняющимися на разных процессорах.

Различие в архитектуре процессоров требует подготовки разных выполняемых файлов для разных узлов, а в случае различий в представлении данных может потребоваться и преобразование информации при передаче сообщений между узлами (не говоря уже о трудностях использования двоичных файлов).

Тем не менее, любой кластер можно рассматривать как единую аппаратно-программную систему, имеющую единую коммуникационную систему, единый центр управления и планирования загрузки.

Вычислительные сети (GRID) объединяют ресурсы множества кластеров, многопроцессорных и однопроцессорных ЭВМ, принадлежащих разным организациям и подчиняющихся разным дисциплинам использования. Разработка параллельных программ для них усложняется из-за следующих проблем.

Конфигурация выделяемых ресурсов (количество узлов, их архитектура и производительность) определяется только в момент обработки заказа на выполнение вычислений. Поэтому программист не имеет возможностей для ручной настройки программы на эту конфигурацию. Желательно осуществлять настройку программы на выделенную конфигурацию ресурсов динамически, без перекомпиляции.

К изначальной неоднородности коммуникационной среды добавляется изменчивость ее характеристик, вызываемая изменениями загрузки сети. Учет такой неоднородности коммуникаций является очень сложной задачей.

Все это требует гораздо более высокого уровня автоматизации разработки параллельных программ, чем тот, который доступен в настоящее время прикладным программистам.

С 1992 года, когда мультикомпьютеры стали самыми производительными вычислительными системами, резко возрос интерес к проблеме разработки для них параллельных прикладных программ. К этому моменту уже было ясно, что трудоемкость разработки прикладных программ для многопроцессорных систем с распределенной памятью является главным препятствием для их широкого внедрения. За прошедший с тех пор период предложено много различных подходов к разработке параллельных программ, созданы десятки различных языков параллельного программирования и множество различных инструментальных средств. Среди них можно отметить следующие интересные отечественные разработки – Норма [2], Fortran-GNS [3], Fortran-DVM [4], mpC [5], T-система [6].

Целью данной работы является сравнительный анализ четырех различных подходов к разработке параллельных программ для проведения научно-инженерных расчетов на вычислительных кластерах и сетях – MPI[7,8], HPF[9,10], OpenMP[11]+MPI и DVM[4,12]. Выбор для анализа именно этих подходов объясняется следующими соображениями:

- Все эти подходы ориентированы на программистов, использующих стандартные языки Фортран или Си. Именно эти программисты и разрабатывают, в основном, параллельные вычислительные программы.

- В основе этих подходов лежат существенно различающиеся модели и языки параллельного программирования.
- На базе этих подходов созданы инструментальные средства разработки параллельных программ, доступные на многих аппаратных платформах.
- И, наконец, для всех этих подходов имеется информация об эффективности выполнения соответствующих реализаций тестов NPB 2.3 [13], что позволяет объективно судить о пригодности этих подходов для разработки сложных параллельных программ.

Предпочтительность использования того или иного подхода определяют следующие факторы:

- Легкость разработки и сопровождения параллельных программ;
- Эффективность выполнения параллельных программ;
- Переносимость и повторное использование параллельных программ.

Именно с этих позиций анализируются указанные подходы в данной работе.

2. Модели и языки параллельного программирования

Как было сказано выше, основной моделью параллельного выполнения программы на кластере является модель передачи сообщений.

В этой модели параллельная программа представляет собой систему процессов, взаимодействующих посредством передачи сообщений.

Можно выбрать модель передачи сообщений и в качестве модели программирования. При этом возможны три способа построения языка программирования:

- Расширение стандартного языка последовательного программирования библиотечными функциями (например, Фортран+MPI);
- Расширение стандартного языка последовательного программирования специальными конструкциями (например, Fortran-GNS);
- Разработка нового языка (например, Occam).

Однако модель передачи сообщений является слишком низкоуровневой, непривычной и неудобной для программистов, разрабатывающих вычислительные программы. Она заставляет программиста иметь дело с параллельными процессами и низкоуровневыми примитивами передачи сообщений.

Поэтому вполне естественно, что прикладной программист хотел бы получить инструмент, автоматически преобразующий его последовательную программу в параллельную программу для кластера. К сожалению, такое автоматическое распараллеливание невозможно в силу следующих причин.

Во-первых, поскольку взаимодействие процессоров через коммуникационную систему требует значительного времени (латентность – время самого простого взаимодействия - велика по сравнению со временем выполнения одной машинной команды), то вычислительная работа должна распределяться между процессорами крупными порциями.

Совсем другая ситуация была на векторных машинах и на мультипроцессорах, где автоматическое распараллеливание программ на языке Фортран реально использовалось и давало хорошие результаты. Для автоматического распараллеливания на векторных машинах (векторизации) достаточно было проанализировать на предмет возможности параллельного выполнения (замены на векторные операции) только самые внутренние циклы программы. В случае мультипроцессоров приходилось уже анализировать объемлющие циклы для нахождения более крупных порций работы, распределяемых между процессорами.

Укрупнение распределяемых порций работы требует анализа более крупных фрагментов программы, обычно включающих в себя вызовы различных процедур. Это, в свою очередь, требует сложного межпроцедурного анализа. Поскольку в реальных программах на языке Фортран могут использоваться конструкции, статический анализ которых принципиально невозможен (например, косвенная индексация элементов массивов), то с увеличением порций распределяемой работы увеличивается вероятность того, что распараллеливатель откажется распараллеливать те конструкции, которые на самом деле допускают параллельное выполнение.

Во-вторых, в отличие от многопроцессорных ЭВМ с общей памятью, на системах с распределенной памятью необходимо произвести не только распределение вычислений, но и распределение данных, а также обеспечить на каждом процессоре доступ к удаленным данным - данным, расположенным на других процессорах. Для обеспечения эффективного доступа к удаленным данным требуется производить анализ индексных выражений не только внутри одного цикла, но и между разными циклами. К тому же, недостаточно просто обнаруживать факт наличия зависимости по данным, а требуется определить точно тот сегмент данных, который должен быть переслан с одного процессора на другой.

В третьих, распределение вычислений и данных должно быть произведено согласованно.

Несогласованность распределения вычислений и данных приведет, вероятнее всего, к тому, что параллельная программа будет выполняться гораздо медленнее последовательной. Если на системе с общей памятью распараллелить один цикл, занимающий 90 процентов времени решения задачи, то можно рассчитывать на почти десятикратное ускорение программы (даже если оставшиеся 10 процентов будут выполняться последовательно). На системе с распределенной памятью распараллеливание этого цикла без учета последовательной части может вызвать не ускорение, а замедление программы. Последовательная часть будет выполняться на одном процессоре или на всех процессорах. Если в этой части используются распределенные массивы, то для такого выполнения потребуется интенсивный обмен данными между процессорами.

Согласованное распределение вычислений и данных требует тщательного анализа всей программы, и любая неточность анализа может привести к катастрофическому замедлению выполнения программы.

Невозможность полностью автоматического распараллеливания имеющихся последовательных программ для их выполнения на кластерах, не означает, конечно, неактуальности работ в этом направлении. Если ввести некоторую дисциплину при написании программ, и, возможно, позволить вставлять в программу некоторые подсказки распараллеливателю, то такие программы могут автоматически преобразовываться в программы, способные выполняться параллельно на кластере. Однако в этом случае следует говорить скорее не о распараллеливании имеющихся последовательных программ, а о написании новых параллельных программ на традиционных языках последовательного программирования или их расширениях.

Теперь переходим к рассмотрению подходов, выбранных для сравнительного анализа.

2.1. Модель передачи сообщений. MPI.

В модели передачи сообщений параллельная программа представляет собой множество процессов, каждый из которых имеет собственное локальное адресное пространство. Взаимодействие процессов - обмен данными и синхронизация - осуществляется посредством передачи сообщений. Обобщение и стандартизация различных библиотек передачи сообщений привели в 1993 году к разработке стандарта

MPI (Message Passing Interface). Его широкое внедрение в последующие годы обеспечило коренной перелом в решении проблемы переносимости параллельных программ, разрабатываемых в рамках разных подходов, использующих модель передачи сообщений в качестве модели выполнения.

В числе основных достоинств MPI по сравнению с интерфейсами других коммуникационных библиотек обычно называют следующие его возможности:

- Возможность использования в языках Фортран, Си, Си++;
- Предоставление возможностей для совмещения обменов сообщениями и вычислений;
- Предоставление режимов передачи сообщений, позволяющих избежать излишнего копирования информации для буферизации;
- Широкий набор коллективных операций (например, широковещательная рассылка информации, сбор информации с разных процессоров), допускающих гораздо более эффективную реализацию, чем использование соответствующей последовательности пересылок точка-точка;
- Широкий набор редукционных операций (например, суммирование расположенных на разных процессорах данных, или нахождение их максимальных или минимальных значений), не только упрощающих работу программиста, но и допускающих гораздо более эффективную реализацию, чем это может сделать прикладной программист, не имеющий информации о характеристиках коммуникационной системы;
- Удобные средства именования адресатов сообщений, упрощающие разработку стандартных программ или разделение программы на функциональные блоки;
- Возможность задания типа передаваемой информации, что позволяет обеспечить ее автоматическое преобразование в случае различий в представлении данных на разных узлах системы.

Однако разработчики MPI подвергаются и суровой критике за то, что интерфейс получился слишком громоздким и сложным для прикладного программиста. Интерфейс оказался сложным и для реализации, в итоге, в настоящее время практически не существует реализаций MPI, в которых в полной мере обеспечивается совмещение обменов с вычислениями.

Появившийся в 1997 проект стандарта MPI-2 [8] выглядит еще более громоздким и неподъемным для полной реализации. Он предусматривает развитие в следующих направлениях:

- Динамическое создание и уничтожение процессов;
- Односторонние коммуникации и средства синхронизации для организации взаимодействия процессов через общую память (для эффективной работы на системах с непосредственным доступом процессоров к памяти других процессоров);
- Параллельные операции ввода-вывода (для эффективного использования существующих возможностей параллельного доступа многих процессоров к различным дисковым устройствам).

2.2. Модель параллелизма по данным. HPF.

В модели параллелизма по данным отсутствует понятие процесса и, как следствие, явная передача сообщений или явная синхронизация. В этой модели данные последовательной программы распределяются программистом по процессорам параллельной машины. Последовательная программа преобразуется компилятором в параллельную программу, выполняющуюся в модели передачи сообщений. При этом вычисления распределяются по правилу собственных вычислений: каждый процессор

выполняет только вычисления собственных данных, т.е. данных, распределенных на этот процессор.

Модель параллелизма по данным имеет следующие достоинства.

- Параллелизм по данным является естественным параллелизмом вычислительных задач, поскольку для них характерно вычисление по одним и тем же формулам множества однотипных величин – элементов массивов.
- В модели параллелизма по данным сохраняется последовательный стиль программирования. Программист не должен представлять программу в виде взаимодействующих процессов и заниматься низкоуровневым программированием передач сообщений и синхронизации.
- Распределение вычисляемых данных между процессорами – это не только самый компактный способ задать распределение работы между процессорами, но и способ повышения локализации данных. Чем меньше данных требуется процессору для выполнения возложенной на него работы, тем быстрее она будет выполнена (лучше используется кэш-память, меньше подкачек с диска страниц виртуальной памяти, меньше пересылок данных с других процессоров).

Обобщение и стандартизация моделей параллелизма по данным привели к созданию в 1993 году стандарта HPF (High Performance Fortran) - расширения языка Фортран 90. Аналогичные расширения были предложены для языка Си и Си++.

Краткий обзор возможностей HPF.

Как уже было сказано выше, прежде всего программист должен распределить данные между процессорами. Это распределение производится в два этапа. Сначала с помощью директивы ALIGN задается соответствие между взаимным расположением элементов нескольких массивов, а затем вся эта группа массивов с помощью директивы DISTRIBUTE отображается на решетку процессоров. Это отображение, например, может осуществляться следующим образом: каждый массив разрезается несколькими гиперплоскостями на секции примерно одинакового объема, каждая из которых будет расположена на своем процессоре. Заданное распределение данных может быть изменено на этапе выполнения программы с помощью операторов REALIGN и REDISTRIBUTE.

В HPF реализуется параллелизм следующих конструкций языка Фортран 90/95: операции над секциями массивов, DO циклы, оператор и конструкция FORALL.

Операции над секциями массивов выполняются параллельно в соответствии с распределением данных. Если для их выполнения требуются коммуникации, то они обеспечиваются компилятором.

Оператор и конструкция FORALL могут рассматриваться как обобщение и расширение операций над секциями массивов.

Многие встроенные функции имеют дело с массивами (например, редукционные функции) и могут выполняться параллельно.

Безусловно, по сравнению с MPI язык HPF намного упрощает написание параллельных программ, однако его реализация требует от компилятора очень высокого интеллекта. Конечно, самая сложная часть работы, которая вызывала проблемы при автоматическом распараллеливании – распределение данных – возлагается теперь на программиста. Но, и с оставшейся частью работы компилятор не всегда способен справиться без дополнительных подсказок программиста. Некоторые такие подсказки

были включены в HPF, но все равно оставались серьезные сомнения относительно эффективности HPF-программ.

К сожалению, эти сомнения оказались не напрасными. В течение нескольких лет так не удалось создать компилятора с приемлемой эффективностью. В 1997 году появился проект стандарта HPF2 [10] , в котором существенно расширены возможности программиста по спецификации тех свойств его программы, извлечь которые на этапе компиляции очень трудно или даже вообще невозможно.

2.3. Гибридная модель параллелизма по управлению с передачей сообщений. OpenMP+MPI.

Модель параллелизма по управлению (в западной литературе используется и другое название – модель разделения работы, work-sharing model) возникла уже давно как модель программирования для мультипроцессоров. На мультипроцессорах в качестве модели выполнения используется модель общей памяти. В этой модели параллельная программа представляет собой систему нитей, взаимодействующих посредством общих переменных и примитивов синхронизации. Нить (по-английски ”thread”) – это легковесный процесс, имеющий с другими нитями общие ресурсы, включая общую оперативную память.

Основная идея модели параллелизма по управлению заключалась в следующем. Вместо программирования в терминах нитей предлагалось расширить языки специальными управляющими конструкциями – параллельными циклами и параллельными секциями. Создание и уничтожение нитей, распределение между ними витков параллельных циклов или параллельных секций (например, вызовов процедур) – все это брал на себя компилятор.

Первая попытка стандартизовать такую модель привела к появлению в 1990 году проекта языка PCF Fortran (проект стандарта X3H5). Однако, этот проект [14] тогда не привлек широкого внимания и, фактически, остался только на бумаге. Возможно, что причиной этого было снижение интереса к мультипроцессорам и всеобщее увлечение мультикомпьютерами и HPF.

Однако, спустя несколько лет ситуация сильно изменилась. Во-первых, успехи в развитии элементной базы сделали очень перспективным и экономически выгодным создавать мультипроцессоры. Во-вторых, широкое развитие получили мультикомпьютеры с DSM (distributed shared memory - распределенная общая память), позволяющие программам на разных узлах взаимодействовать через общие переменные также, как и на мультипроцессорах (Convex Exemplar, HP 9000 V-class, SGI Origin 2000). В-третьих, не оправдались надежды на то, что HPF станет фактическим стандартом для разработки вычислительных программ.

Крупнейшие производители компьютеров и программного обеспечения объединили свои усилия и в октябре 1997 года выпустили описание языка OpenMP Fortran – расширение языка Фортран 77. Позже вышли аналогичные расширения языков Си и Фортран 90/95.

Краткий обзор возможностей OpenMP.

OpenMP – это интерфейс прикладной программы, расширяющий последовательный язык программирования набором директив компилятора, вызовов функций библиотеки поддержки выполнения и переменных среды.

Программа начинает свое выполнение как один процесс, называемый главной нитью. Главная нить выполняется последовательно, пока не встретится первая параллельная область программы. Параллельная область определяется парой директив PARALLEL и END PARALLEL. При входе в параллельную область главная нить порождает некоторое число подчиненных ей нитей, которые вместе с ней образуют текущую группу нитей. Все операторы программы, находящиеся в параллельной конструкции, включая и вызываемые изнутри нее процедуры, выполняются всеми нитями текущей группы параллельно, пока

не произойдет выход из параллельной области или встретится одна из конструкций распределения работы - DO**,** SECTIONS или SINGLE.

Конструкция DO служит для распределения витков цикла между нитями, конструкция SECTIONS – для распределения между нитями указанных секций программы, а конструкция SINGLE указывает секцию, которая должна быть выполнена только одной нитью.

При выходе из параллельной конструкции все порожденные на входе в нее нити сливаются с главной нитью, которая и продолжает дальнейшее выполнение.

В программе может быть произвольное число параллельных областей, причем допускается их вложенность.

При параллельной области можно указать классы используемых в ней переменных (общие или приватные).

Имеются директивы высокоуровневой синхронизации (критические секции, барьер, и пр.).

Набор функций системы поддержки и переменных окружения служит для управления количеством создаваемых нитей, способами распределения между ними витков циклов, для низкоуровневой синхронизации нитей с помощью замков.

Интересно, что подход OpenMP является диаметрально противоположным к подходу HPF:

- Вместо параллелизма по данным – параллелизм по управлению;
- Вместо изощренного статического анализа для автоматического поиска операторов, способных выполняться параллельно – явное и полное задание параллелизма программистом;
- Вместо языка, требующего специального HPF-компилятора даже для работы на последовательной ЭВМ – язык, позволяющий на последовательной ЭВМ компилироваться и выполняться в стандартной среде языка Фортран.

Объединение подходов OpenMP и MPI.

Успешное внедрение OpenMP на мультипроцессорах и DSM-мультикомпьютерах резко активизировало исследования, направленные на поиски путей распространения OpenMP на мультикомпьютеры, кластеры и сети ЭВМ. Эти исследования сосредоточились, в основном, на двух направлениях:

- Расширение языка средствами описания распределения данных;
- Программная реализация системы DSM, использующей дополнительные указания компилятора, вставляемые им в выполняемую программу.

Первое направление представляется гораздо более перспективным для кластеров и сетей ЭВМ, однако трудно ожидать появления в ближайшие годы время стандарта нового языка (расширенного OpenMP).

Поэтому все шире начинает использоваться гибридный подход, когда программа представляет собой систему взаимодействующих MPI-процессов, а каждый процесс программируется на OpenMP.

Такой подход имеет преимущества с точки зрения упрощения программирования в том случае, когда в программе есть два уровня параллелизма – параллелизм между подзадачами и параллелизм внутри подзадачи. Такая ситуация возникает, например, при использовании многообластных (многоблочных) методов решения вычислительных задач. Программировать на MPI сами подзадачи гораздо сложнее, чем их взаимодействие, поскольку распараллеливание подзадачи связано с распределением элементов массивов и витков циклов между процессами. Организация же взаимодействия подзадач таких сложностей не вызывает, поскольку сводится к обмену между ними граничными

значениями. Нечто подобное программисты делали раньше на однопроцессорных ЭВМ, когда для экономии памяти на каждом временном шаге выполняли подзадачи последовательно друг за другом.

Широкое распространение SMP-кластеров также подталкивает к использованию гибридного подхода, поскольку использование OpenMP на мультипроцессоре может для некоторых задач (например, вычислений на неструктурных сетках) дать заметный выигрыш в эффективности.

Основной недостаток этого подхода также очевиден - программисту надо знать и уметь использовать две разные модели параллелизма и разные инструментальные средства.

2.4. Модель параллелизма по данным и управлению. DVM.

Эта модель, положенная в основу языков параллельного программирования Fortran-DVM и C-DVM, объединяет достоинства модели параллелизма по данным и модели параллелизма по управлению. Базирующаяся на этих языках система разработки параллельных программ (DVM) создана в Институте прикладной математики им. М.В. Келдыша РАН при активном участии студентов и аспирантов факультета ВМиК МГУ им. М.В.Ломоносова.

В отличие от модели параллелизма по данным, в системе DVM программист распределяет по процессорам виртуальной параллельной машины не только данные, но и соответствующие вычисления. При этом на него возлагается ответственность за соблюдение правила собственных вычислений. Кроме того, программист определяет общие данные, т.е. данные, вычисляемые на одних процессорах и используемые на других процессорах. И, наконец, он отмечает точки в последовательной программе, где происходит обновление значений общих данных.

При построении системы DVM был использован новый подход, который характеризуется следующими принципами.

1. Система должна базироваться на высокоуровневой модели выполнения параллельной программы, удобной и понятной для программиста, привыкшего программировать на последовательных языках. Такая модель (DVM-модель) была разработана в 1994 году [4].
2. Языки параллельного программирования должны представлять собой стандартные языки последовательного программирования, расширенные спецификациями параллелизма. Эти языки должны предлагать программисту модель программирования, достаточно близкую к модели выполнения. Знание программистом модели выполнения его программы и ее близость к модели программирования существенно упрощает для него анализ производительности программы и проведение ее модификаций, направленных на достижение приемлемой эффективности.
3. Спецификации параллелизма должны быть прозрачными для обычных компиляторов (например, оформляться в виде специальных комментариев). Во-первых, это упрощает внедрение новых параллельных языков, поскольку программист знает, что его программа без каких-либо изменений может выполняться в последовательном режиме на любых ЭВМ. Во-вторых, это позволяет использовать следующий метод поэтапной отладки DVM-программ. На первом этапе программа отлаживается на рабочей станции как последовательная программа, используя обычные методы и средства отладки. На втором этапе программа выполняется на той же рабочей станции в специальном режиме проверки DVM-указаний. На третьем этапе программа может быть выполнена в специальном режиме, когда промежуточные результаты параллельного выполнения сравниваются с эталонными результатами (например, результатами последовательного выполнения).

4. Основная работа по реализации модели выполнения параллельной программы (например, распределение данных и вычислений) должна осуществляться динамически специальной системой - системой поддержки выполнения DVM-программ. Это позволяет обеспечить динамическую настройку DVM-программ при запуске (без перекомпиляции) на конфигурацию параллельного компьютера (количество процессоров, их производительность, латентность и пропускную способность коммуникационных каналов). Тем самым программист получает возможность иметь один вариант программы для выполнения на последовательных ЭВМ и параллельных ЭВМ различной конфигурации. Кроме того, на основании информации о выполнении DVM-программы на однопроцессорной ЭВМ можно посредством моделирования работы системы поддержки предсказать характеристики выполнения этой программы на параллельной ЭВМ с заданными параметрами (производительностью процессоров и коммуникационных каналов).

Большое влияние на разработку этого подхода оказали работы по языку Fortran D [15], по языку PCF Fortran, а также участие авторов в создании управляемой виртуальной памяти для ЭВМ БЭСМ-6 [16].

Краткий обзор возможностей языков Fortran-DVM и C-DVM.

Программа на языках Fortran-DVM и C-DVM, помимо описания алгоритма обычными средствами языков Фортран 77 или Си, содержит правила параллельного выполнения этого алгоритма.

Программисту предоставляются следующие возможности спецификации параллельного выполнения программы:

- распределение элементов массива между процессорами;
- распределение витков цикла между процессорами;
- спецификация параллельно выполняющихся секций программы (параллельных задач) и отображение их на процессоры;
- организация эффективного доступа к удаленным (расположенным на других процессорах) данным;
- организация эффективного выполнения редукционных операций - глобальных операций с расположенными на различных процессорах данными (таких, как их суммирование или нахождение их максимального или минимального значения).

Модель выполнения программы можно упрощенно описать следующим образом.

Параллельная программа на исходном языке Fortran-DVM (или C-DVM) превращается в программу на языке Фортран 77 (или Си), содержащую вызовы функций системы поддержки, и выполняющуюся в соответствии с моделью SPMD (одна программа – много данных) на каждом выделенном задаче процессоре.

В момент запуска программы существует единственная её ветвь (поток управления), которая начинает свое выполнение с первого оператора программы сразу на всех процессорах многопроцессорной системы.

Многопроцессорной системой (или системой процессоров) называется та машина, которая предоставляется программе пользователя аппаратурой и базовым системным программным обеспечением. Для распределённой ЭВМ примером такой машины может служить MPI-машина. В этом случае, многопроцессорная система – это группа MPI-процессов, которые создаются при запуске параллельной программы на выполнение. Число процессоров многопроцессорной системы и её представление в виде многомерной решетки задаётся в командной строке при запуске программы.

Все объявленные в программе переменные (за исключением специально указанных "распределённых" массивов) размножаются по всем процессорам.

При входе в параллельную конструкцию (параллельный цикл или область параллельных задач) ветвь разбивается на некоторое количество параллельных ветвей, каждая из которых выполняется на выделенном ей процессоре многопроцессорной системы.

При выходе из параллельной конструкции все ветви сливаются в ту же самую ветвь, которая выполнялась до входа в параллельную конструкцию.

Недостатком системы DVM является то, что предоставляются только параллельные расширения языков Фортран 77 и Си, а расширения языков Фортран 90/95 и Си++ отсутствуют. Правда, следует сказать, что Fortran-DVM базируется на расширенном языке Фортран 77, уже включающем в себя ряд возможностей Фортрана 90, и планируется дальнейшее такое расширение.

3. Легкость разработки и сопровождения параллельных программ

Корректно сравнить описанные выше подходы по этому фактору, конечно же, не представляется возможным. Однако можно высказать ряд суждений по следующим вопросам:

- адекватность модели и языка программирования рассматриваемому классу задач;
- какие методы и средства отладки предоставляются программистам;
- сколько вариантов программы приходится сопровождать программисту.

Адекватность модели и языка программирования

Многолетний опыт использования вычислительных машин позволяет сделать вывод, что выбранному классу задач (проведение научно-инженерных расчетов) вполне адекватен язык Фортран.

Возьмем простейший, но достаточно характерный вычислительный алгоритм, реализующий метод релаксации Якоби для решения систем линейных уравнений, и сравним размеры его последовательной версии и трех параллельных версий (MPI, HPF, DVM). Версия алгоритма с использованием гибридного подхода OpenMP+MPI не рассматривалась, поскольку невозможно применить возможности OpenMP для упрощения реализации данного алгоритма. Размер в строках последовательной программы на языке Fortran 77 равен 17, а размеры параллельных версий для MPI, HPF и DVM равны соответственно 55, 24 и 21. Тексты программ можно найти в приложении 1.

Теперь проведем такое же сравнение на широко известных тестах NAS (NPB 2.3).

Эти тесты хорошо отражают характер вычислительных задач различных классов, за исключением задач с нерегулярными сетками. Ниже дается краткая характеристика тестов, и приводятся их размеры в строках для трех версий каждой программы – последовательной версии, MPI-версии и DVM-версии. Информации о размерах программ с использованием OpenMP+MPI нет, но можно точно утверждать, что эти размеры лишь немного превышают размеры MPI-версий. Информации о размерах HPF-версий также нет, но можно предполагать, что они незначительно отличаются от размеров DVM-версий.

Тест	Характеристика теста	SEQ	MPI	DVM	MPI/ SEQ	DVM /SEQ
BT	3D Навье-Стокс, метод переменных направлений	3929	5744	3991	1.46	1.02
CG	Оценка наибольшего собственного значения симметричной разреженной матрицы	1108	1793	1118	1.62	1.01
EP	Генерация пар случайных чисел Гаусса	641	670	649	1.04	1.01
FT	Быстрое преобразование Фурье, 3D спектральный метод	1500	2352	1605	1.57	1.07
IS	Параллельная сортировка	925	1218	1067	1.32	1.17
LU	3D Навье-Стокс, метод верхней релаксации	4189	5497	4269	1.31	1.02
MG	3D уравнение Пуассона, метод Multigrid	1898	2857	2131	1.50	1.12
SP	3D Навье-Стокс, Beam-Warning approximate factorization	3361	5020	3630	1.49	1.08
Σ		17551	25151	18460	1.43	1.05

SEQ – последовательная программа
MPI – параллельная программа с использованием Fortran 77+MPI или C+MPI (IS)
DVM – параллельная программа на языке Fortran-DVM или C-DVM (IS)

В качестве грубой оценки сложности программирования вполне можно использовать данные о соотношении количества дополнительных операторов, которые пришлось при распараллеливании тестов NAS добавить в их последовательные версии – 43% для MPI и 5% для DVM. Следует отметить при этом, что дополнительные операторы DVM-программы являются специальными комментариями, не зависящими от размеров массивов и числа процессоров. Дополнительный код MPI-программы представляет собой сложную систему программ управления передачей сообщений, зависящих от размеров массивов и числа процессоров.

Методы и средства отладки

Отладка параллельной программы является процессом более трудоемким, чем отладка последовательной программы. Причиной этого является не только сложность параллельной программы, но и ее недетерминированное поведение, серьезно затрудняющее и функциональную отладку (достижение правильности результатов), и отладку эффективности программы. Раньше с подобными трудностями сталкивались, в основном, разработчики операционных систем и систем реального времени, которые сами и создавали для себя специальные средства отладки. Развитые средства отладки могут существенно упростить разработку параллельных программ прикладными программистами.

Большинство современных средств отладки параллельных программ основано на представлении программы как совокупности выполняющихся процессов.

Средства функциональной отладки, как правило, предоставляют тот же набор примитивов, что и обычные последовательные отладчики, расширенный с учетом специфики параллельного выполнения. Сюда входят следующие базовые примитивы:

- инициализация выполнения программы;
- завершение программы;
- приостановка и продолжение выполнения программы;
- задание точки останова, проверка заданных условий, просмотр и модификация значений переменных;
- пошаговое исполнение.

Кроме того, широко используются средства накопления и анализа трассировки.

Наиболее развитыми системами функциональной отладки MPI-программ являются TotalView [17] и PGDBG [18].

При отладке эффективности используются разного рода профилировщики, выдающие после завершения программы информацию о временах вычислений, обменов сообщениями и синхронизации. Примерами таких систем, используемых для анализа эффективности MPI-программ, являются: Nupshot [19], Pablo [20], Vampir [21].

Несмотря на обилие различных инструментов для отладки MPI-программ положение дел в этой области нельзя признать удовлетворительным по следующим причинам.

Во-первых, каждая система отладки предоставляет свой набор возможностей и свой интерфейс с пользователем.

Во-вторых, каждая достаточно развитая система ориентирована на работу с конкретными компиляторами и библиотеками MPI. В результате, при переходе пользователя на другую машину, вероятнее всего, он не найдет там привычной для него системы отладки.

Некоторые из систем отладки MPI-программ были адаптированы для отладки HPF-программ, а также программ, использующих гибридный подход OpenMP+MPI.

Для функциональной отладки DVM-программ была предложена и реализована следующая методика поэтапной отладки программ.

На первом этапе программа отлаживается на рабочей станции как последовательная программа, используя обычные методы и средства отладки. На втором этапе программа выполняется на той же рабочей станции в специальном режиме моделирования параллельного выполнения для проверки корректности распараллеливающих указаний. На третьем этапе программа может быть выполнена на параллельной машине в специальном режиме, когда промежуточные результаты параллельного выполнения сравниваются с эталонными результатами (например, результатами последовательного выполнения).

Для обеспечения второго и третьего этапов этой методики служит специальный DVM-отладчик.

Для отладки эффективности DVM-программ используется анализатор производительности**,** который позволяет пользователю получить информацию об основных характеристиках эффективности выполнения его программы (или ее частей) на параллельной системе.

Для облегчения отладки эффективности можно использовать специальный инструмент - предиктор, позволяющий на рабочей станции смоделировать выполнение DVM-программы на параллельной ЭВМ с заданными параметрами (топологии коммуникационной сети, ее пропускной способности, а также производительности процессоров) и получить прогнозируемые характеристики эффективности ее выполнения.

Сопровождаемые варианты программы

Если параллельная программа разработана с использованием MPI, то использовать ее на обычных персональных компьютерах или рабочих станциях затруднительно по двум основным причинам:

Во-первых, для ее компиляции и выполнения требуется наличие библиотеки MPI, а во-вторых, программа должна быть написана так, чтобы она могла выполняться на одном процессоре. Поэтому, как правило, программист имеет и вынужден сопровождать два варианта программы – для последовательного и для параллельного выполнения.

Программа на языке HPF способна выполняться на одном процессоре. Однако использовать ее на обычных персональных компьютерах или рабочих станциях невозможно, если там отсутствует компилятор с языка HPF.

Поскольку DVM-указания прозрачны для обычных компиляторов, то один вариант DVM-программы может использоваться и для параллельного выполнения, и для последовательного выполнения на персональных компьютерах или рабочих станциях.

4. Эффективность выполнения параллельных программ

Эффективность выполнения программ всегда являлась очень важным фактором, определявшим в значительной степени успех и распространение языков программирования, предназначенных для создания вычислительных программ.

Джон Бэкус [22] выразил эту мысль примерно следующими словами: “Если бы в первые месяцы после появления компилятора с языка Фортран он при трансляции представительных вычислительных программ генерировал бы коды, выполняющиеся в два раза медленнее написанных на ассемблере программ, то распространение языка Фортран было бы под угрозой”.

В настоящее время, когда распараллеливание программы ускоряет ее выполнение в сотни раз, а трудоемкость создания параллельных программ является основным препятствием для широкого использования высокой производительности современных вычислительных систем, эффективность выполнения программ, тем не менее, по-прежнему остается важным фактором при выборе программистом того или иного подхода для разработки параллельных программ.

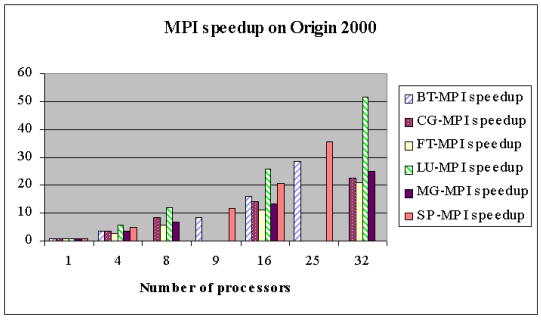
Ниже приводятся данные об эффективности выполнения шести тестов из пакета NPB 2.3 (BT, CG, FT, LU, MG, SP) класса A, реализованных с использованием подходов, выбранных для сравнительного анализа.

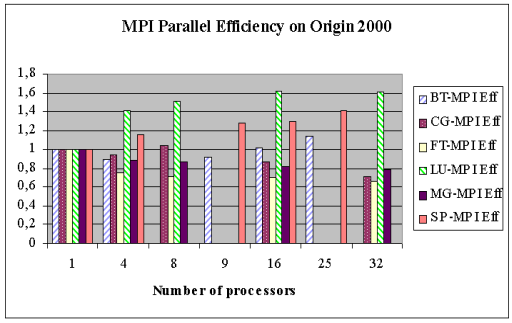
Два других теста из пакета NPB 2.3 (IS и EP) не использовались при исследовании применимости HPF-подхода и гибридного подхода OpenMP+MPI. Причиной этого было то, что первый из них написан на языке Си и не является представительным (параллельная сортировка целых чисел), а второй служит для демонстрации абсолютного параллелизма и его распараллеливание ни у кого никаких проблем не вызывает.

Использованные в приведенных ниже диаграммах данные, заимствованные из различных источников [23,24], были получены на разных параллельных системах. Тем не менее, они позволяют сделать достаточно объективные выводы.

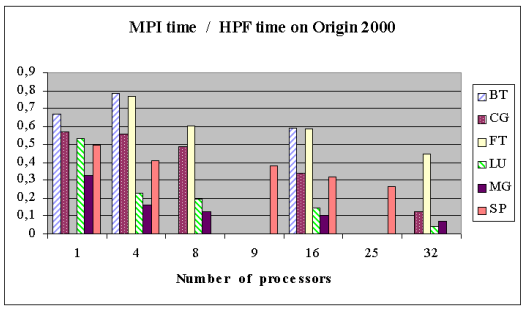
В приведенных диаграммах используются следующие обозначения:

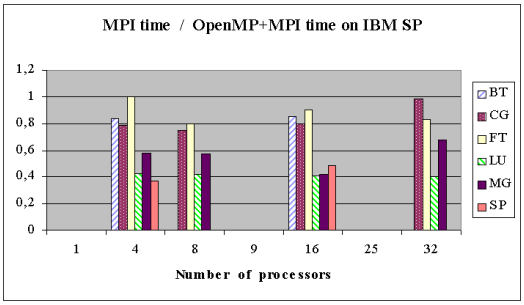
- *Speedup* – ускорение выполнения параллельной программы на нескольких процессорах по отношению к ее выполнению на одном процессоре.
- *Efficiency* – эффективность параллельного выполнения, равная отношению ускорения к числу процессоров.

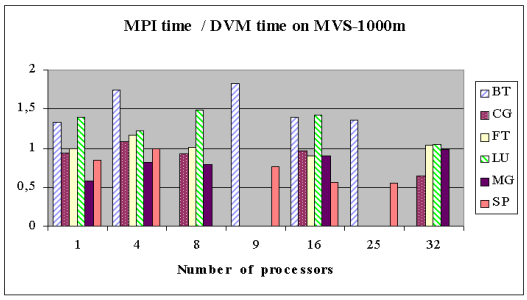


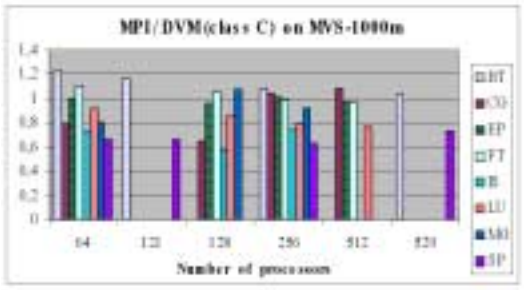


Ниже для каждого теста приведены отношения времени выполнения его MPI-версии к времени выполнения версии этого теста, разработанной с использованием трех других подходов (HPF, OpenMP+MPI, DVM).









Диаграммы показывают, что MPI-программы, как правило, выполняются быстрее остальных. При этом, обычно разрыв увеличивается по мере роста числа процессоров.

При использовании подхода OpenMP+MPI в MPI-программу для отдельного узла вставлялись директивы OpenMP. Отсюда довольно неожиданный вывод – даже на мультипроцессоре с 4 процессорами на общей памяти, используемом в качестве узла IBM SP, OpenMP проигрывает на регулярных программах MPI-подходу. Это объясняется лучшей локализацией доступа к данным в MPI-программе, что приводит к более эффективному использованию кэш-памяти.

Большой проигрыш HPF при увеличении числа процессоров объясняется принципиальными недостатками HPF-подхода, о которых говорилось выше.

Эффективность DVM-программ гораздо выше, чем эффективность HPF, и вполне сравнима с эффективностью MPI. Для оценки масштабируемости DVM-программ приведены данные о соотношении времен выполнения MPI-версий и DVM-версий на ЭВМ МВС-1000м [25] для всех восьми тестов класса C (тесты этого класса ориентированы на высокопроизводительные параллельные системы, требуют много памяти и поэтому не могут выполняться на малом числе процессоров).

Из них следует, что в среднем эффективность DVM-версий составляет около 80% от эффективности MPI-версий.

Конечно, сравнение на тестах NPB 2.3 не вполне правомерно – они написаны на очень высоком профессиональном уровне и являются объектом пристального внимания многих специалистов. При разработке реальных параллельных программ, как правило, достижение высокой эффективности требует многократных изменений программы для поиска наилучшей схемы ее распараллеливания. Успешность такого поиска определяется простотой модификации программы. Кроме того, прикладному программисту трудно реализовать многие часто используемые приемы распараллеливания так же эффективно, как они реализуются системами программирования. Поэтому на реальных программах MPI-подход, как правило, проигрывает по эффективности DVM-подходу.

5. Переносимость и повторное использование параллельных программ

В настоящее время, когда программист имеет возможность запускать свою программу на разных, порою географически удаленных параллельных системах, важность переносимости программ (их способности выполняться на различных вычислительных системах с приемлемой эффективностью) трудно переоценить.

Создать для новых параллельных систем прикладное программное обеспечение, необходимое для решения важнейших научно-технических задач, вряд ли возможно без повторного использования уже созданных программ, без накопления и использования богатых библиотек параллельных программ.

Поэтому переносимость программ и их способность к повторному использованию должны рассматриваться как самые первостепенные показатели качества параллельных программ.

Как уже говорилось выше, широкое внедрение MPI обеспечило переносимость программ, разрабатываемых в рамках MPI-подхода для однородных кластеров. Правда, определенные проблемы возникают из-за различий в организации внешней памяти на разных кластерах. Например, для эффективной работы с локальными дисками, имеющимися на каждом узле кластера, программа должна соответствующим образом это организовать. Переход на кластер, на котором используется единый файл-сервер и нет локальных дисков, потребует изменения такой программы. Использовать же параллельный ввод-вывод, предложенный в стандарте MPI-2, тяжело из-за его сложности, и из-за того, что он не на всех кластерах поддерживается.

Гораздо хуже обстоят дела с повторным использованием MPI-программ. Очень сложно разрабатывать программы, способные выполняться на различном числе процессоров с разными по объему данными. Многие программисты предпочитают иметь разные варианты программ для разных конфигураций данных и процессоров, а также иметь отдельный вариант программы для работы на однопроцессорной ЭВМ. В таких условиях использование чужой программы представляется гораздо менее вероятной, чем это было на традиционных ЭВМ.

Но главные трудности связаны с отсутствием в языке программирования такого понятия, как распределенный массив. Вместо такого единого массива в программе используется на каждом процессоре свой локальный массив. Их соответствие исходному массиву, с которым программа имела бы дело при ее выполнении на одном процессоре, зафиксировано только в голове программиста и, возможно, в комментариях к программе.

Отсутствие в языке понятия распределенного массива является серьезным препятствием для разработки и использования библиотек стандартных параллельных программ. В результате, каждая такая библиотека вынуждена вводить свое понятие распределенного массива и реализовывать свой набор операций над ним. Однако вызов таких стандартных программ из прикладной программы требует от программиста согласования разных представлений распределенных массивов.

Все вышесказанное в значительной мере относится и к гибридному подходу OpenMP+MPI.

Таких принципиальных проблем с повторным использованием параллельных программ нет, если эти программы разрабатываются в рамках подходов HPF или DVM. Более того, в рамках этих подходов можно обеспечить удобный вызов стандартных параллельных программ, разработанных на других языках и входящих в состав известных библиотек (для которых известно их внутреннее представление распределенных массивов). Например, можно позволить вызывать функции пакета ScaLAPACK [26] из HPF-программ или DVM-программ.

Переносимость HPF-программ определяется наличием компилятора HPF для конкретной параллельной системы, а точнее для процессора, используемого в узлах этой

системы. Поскольку качественных и свободно распространяемых компиляторов, способных генерировать программы для всех процессоров, нет, то при переносе HPF-программ на некоторые платформы возникнут серьезные проблемы.

Переносимость DVM-программы на однородные кластеры обеспечивается тем, что она преобразуется в программу на языке Фортран 77 (или Си), содержащую вызовы функций системы поддержки, которая для организации межпроцессорного взаимодействия использует библиотеку MPI. Такая программа может выполняться всюду, где есть MPI и компиляторы с языков Си и Фортран 77. Система поддержки может учесть особенности организации внешней памяти и обеспечить ее эффективное использование, не требуя изменений в DVM-программе.

Кроме того, программы на языке Fortran-DVM могут автоматически конвертироваться в программы на языке HPF, HPF2 и OpenMP.

Способность параллельных программ эффективно выполняться на неоднородных кластерах и сетях ЭВМ требует отдельного рассмотрения.

Все три подхода, базирующиеся на существующих стандартах (MPI, HPF, OpenMP+MPI), не рассчитаны на применение на неоднородных распределенных системах.

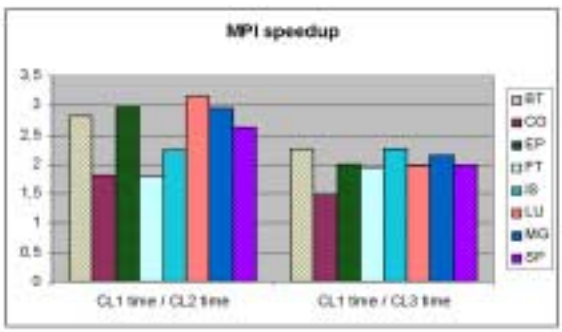
Поэтому перенос таких программ на неоднородные кластеры и сети ЭВМ приведет к заметной потере их эффективности, поскольку прикладной программист фактически не имел возможностей для написания программ, способных настраиваться на различную производительность процессоров и коммуникационных каналов.

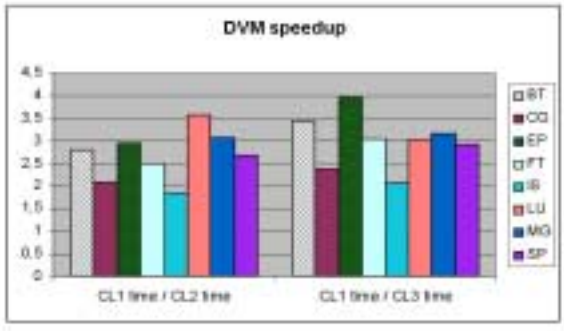
В DVM-системе реализованы возможности задания производительностей процессоров (или их автоматического определения при запуске программы) и их учета при распределении данных и вычислений между процессорами. Это позволяет DVM-программам эффективно выполняться на неоднородных кластерах. Это подтверждают результаты пропуска тестов NAS на модели неоднородного кластера.

Неоднородный кластер был смоделирован на машине МВС-1000м путем увеличения процессорных времен между последовательными обращениями к MPI функциям.

Ниже на диаграммах показано соотношение времен выполнения MPI и DVM версий тестов NAS (класс C) на следующих конфигурациях:

- CL1 – 128 процессоров со скоростью выполнения P,
- CL2 – 128 процессоров со скоростью выполнения 3P,
- CL3 – 128 процессоров со скоростью выполнения P и 128 процессоров со скоростью выполнения 3P (неоднородный кластер).





Из диаграммы видно, что добавление к 128-ми быстрым процессорам 128-ми медленных приводит к ускорению выполнения DVM-программ, а выполнение MPI-программ существенно замедляется. Замедление выполнения MPI-программ и является причиной того, что разработчики кластеров стараются избегать добавления к имеющимся процессорам более быстрых процессоров, а просто создают из этих процессоров новый кластер. При этом игнорируется тот факт, что на большом кластере можно решать более объемные задачи, хотя и ценой снижения эффективности его использования.

Сложнее обеспечить эффективное выполнение параллельных программ на системе, объединяющей несколько кластеров специальными коммуникационными каналами или

обычной локальной сетью. К неоднородности процессоров добавляется неоднородность коммуникаций.

Но и в этом случае, DVM-подход может обеспечить эффективное выполнение многих параллельных программ.

Трудно что-то сделать для задач, в которых процессоры интенсивно обмениваются по схеме “каждый с каждым”.

Однако для программ, в которых процессоры в основном обмениваются сообщениями со своими соседями (в логической решетке процессоров!), могут применяться следующие методы. Во-первых, для компенсации высокой латентности каналов можно так отображать задачу на неоднородную систему (выбирать логическую решетку процессоров и отображать ее на физическую решетку процессоров), чтобы минимизировать количество передаваемых по ним сообщений. При этом общий объем передаваемой информации будет оставаться неизменным. Во-вторых, можно понизить вычислительную загрузку тех процессоров, которые общаются между собой через медленные каналы.

Еще больше можно сделать для многообластных задач и задач на неструктурированных сетках. При отображении таких задач на неоднородную систему можно не только минимизировать количество сообщений, передаваемых по медленным каналам, но и снизить суммарный объем передаваемой по ним информации. И, конечно же, можно понизить вычислительную загрузку тех процессоров, которые общаются между собой через медленные каналы. Поскольку доля таких задач среди задач, требующих высокопроизводительных вычислений, будет неуклонно возрастать, то можно уверенно говорить о перспективности использования неоднородных кластеров и сетей ЭВМ.

6. Заключение

На основании проведенного анализа четырех различных подходов к разработке параллельных программ для вычислительных кластеров и сетей ЭВМ (MPI, HPF, OpenMP+MPI и DVM) можно сделать следующие выводы.

1. С точки зрения простоты разработки и сопровождения параллельных программ, а также их повторного использования явное преимущество имеют подходы HPF и DVM.
2. По эффективности выполнения программ HPF заметно отстает от остальных подходов.
3. Гибридный подход OpenMP+MPI, MPI-подход и HPF-подход, не могут обеспечить эффективного выполнения программ на неоднородных кластерах и сетях ЭВМ.

Таким образом, ни один из трех подходов (MPI, HPF, и OpenMP+MPI), базирующихся на имеющихся стандартах, не может рассматриваться в настоящее время как вполне подходящий для разработки параллельных программ для вычислительных кластеров и сетей ЭВМ. Эту точку зрения разделяют многие ведущие специалисты в области параллельных вычислений.

Освоение DVM-подхода может существенно сократить время написания, отладки и сопровождения программ, и позволит создавать параллельные программы, эффективно выполняющиеся на вычислительных кластерах и сетях.

Новый язык Fortran OpenMP/DVM (расширение Fortran OpenMP директивами DVM), реализация которого ведется в настоящее время, будет способствовать широкому внедрению DVM-подхода, поскольку параллельная программа на этом языке будет не только обладать всеми достоинствами DVM-программ, но и являться стандартной параллельной программой для мультипроцессоров и DSM-кластеров.

Однако наличие самой передовой технологии разработки параллельных программ само по себе не может решить проблему ускорения освоения параллельных вычислительных систем.

Большинство параллельных программ создаются с использованием того огромного программного задела, который был получен на последовательных ЭВМ. Необходимость разработки методики распараллеливания существующих последовательных программ ощущается очень остро. В ИПМ им.М.В.Келдыша РАН ведутся работы по обобщению опыта распараллеливания программ и созданию соответствующей методики. Такая методика должна быть поддержана специальными инструментами, автоматизирующими анализ последовательных программ и извлечение их свойств, существенных для распараллеливания этих программ. Работы В.В.Воеводина и Вл.В.Воеводина, ведущиеся в этом направлении, позволяют верить в появление таких инструментов в ближайшем будущем.

Острой является и проблема подготовки кадров, способных эффективно использовать параллельные системы. Для курсов лекций по параллельной обработке, читаемых в ВУЗах, характерен чисто теоретический уклон. В них мало внимания уделяется изучению практических технологий параллельного программирования. Как правило, такие курсы не поддержаны практическими занятиями. Первый положительный опыт по проведению практикума по параллельному программированию с выходом на реальные параллельные системы (кластер рабочих станций и многопроцессорная ЭВМ МВС-1000м), полученный в 2002 году на факультете ВМиК МГУ при поддержке фирмы Интел и Межведомственного суперкомпьютерного центра, дает основания надеяться на постепенное улучшение положения дел в этом вопросе.

Список литературы

1. Foster I., Kesselman C. The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann Publishers, Inc., San Francisco, CA, USA, 1999.
2. Андрианов А.Н., Ефимкин К.Н., Задыхайло И.Б. Язык Норма. Препринт ИПМ им. М.В.Келдыша АН СССР, № 165, 1985.
3. Zadykhailo I.B., Krukov V.A. and Pozdnjakov L.A. 'RAMPA - CASE for portable parallel programs development", Proc. of the International Conference on Parallel Computing Technologies, Obninck, Russia, 1993.
4. Konovalov N.A., Krukov V.A., Mihailov S.N. and Pogrebtsov A.A. Fortran DVM - a Language for Portable Parallel Program Development. Proceedings of Software For Multiprocessors & Supercomputers: Theory, Practice, Experience. Institute for System Programming, RAS, Moscow, 1994.
5. Lastovetsky A. mpC - a Multi-Paradigm Programming Language for Massively Parallel Computers, ACM SIGPLAN Notices, 31(2):13-20, February 1996.
6. Abramov S., Adamovitch A. and Kovalenko M. T-system: programming environment providing automatic dynamic parallelizing on IP-network of Unix-computers. Report on 4-th International Russian-Indian seminar and exibition, Sept. 15-25, 1997, Moscow. http://www.botik.ru/~abram/ts-pubs.html
7. Message-Passing Interface Forum, Document for a Standard Message-Passing Interface, 1993. Version 1.0. http://www.unix.mcs.anl.gov/mpi/
8. Message-Passing Interface Forum, MPI-2: Extensions to the Message-Passing Interface, 1997. http://www.unix.mcs.anl.gov/mpi/
9. High Performance Fortran Forum. High Performance Fortran Language Specification. Version 1.0, May 1993.
10. High Performance Fortran Forum. High Performance Fortran Language Specification. Version 2.0, January 1997.

11. OpenMP Consortium: OpenMP Fortran Application Program Interface, Version 1.0, October 1997. http://www.openmp.org/
12. DVM-система. http://www.keldysh.ru/dvm/
13. Bailey D., Harris T., Saphir W., Van der Wijngaart, Woo A., Yarrow M.The NAS Parallel Benchmarks 2.0. NAS Technical Report NAS-95-020, NASA Ames Research Center, Moffett Field, CA, 1995. http://science.nas.nasa.gov/Software/NPB.
14. PCF Fortran. Version 3.1. Aug.1, 1990.
15. Hiranandani S., Kennedy K., Tseng C. Compiling Fortran D for MIMD Distributed-Memory Machines. Comm. ACM, Vol. 35,No. 8 (Aug. 1992), 66-80.
16. Коновалов Н.А., Крюков В.А., Любимский Э.З. Управляемая виртуальная память.Программирование, №1, 1977.
17. TotalView. http://www.etnus.com/Products/TotalView/index.html
18. Portland Group Debugger. http://www.pgroup.com
19. Nupshot . http://www.mcs.anl.gov/mpi/mpich/
20. Pablo. http://www-pablo.cs.uiuc.edu
21. Vampir. http://www.pallas.de/pages/vampir.htm
22. Backus J. The history of FORTRAN I, II and III. ACM SIGPLAN Notices, 13(8):165-180,1978.
23. Frumkin M., Jin H., and Yan J. Implementation of NAS Parallel Benchmarks in High Performance Fortran. NAS Technical Report NAS-98-009, NASA Ames Research Center, Moffett Field, CA, 1998.
24. Capello F., Etiemble D. MPI versus MPI+OpenMP on the IBM SP for the NAS Benchmarks. In *Proceedings of Supercomputing '2000*, 2000.
25. Елизаров Г.С., Забродин А.В., Левин В.К., Каратанов В.В., Корнеев В.В., Савин Г.И., Шабанов Б.М. Структура многопроцессорной вычислительной системы МВС-1000М. Труды Всероссийской научной конференции "Высокопроизводительные вычисления и их приложения", г.Черноголовка, 30 октября - 2 ноября 2000 г., Изд-во Московского университета, 2000.
26. Dongarra J., Walker D., and others. *ScaLAPACK Users' Guide*. Philadelphia: Society for Industrial and Applied Mathematics, 1997.

Приложение 1.

Примеры последовательной и параллельных реализаций алгоритма Якоби

Последовательная программа на языке Fortran 77

```
PROGRAM    JAC_F77
PARAMETER    (L=8,  ITMAX=20)
REAL     A(L,L), B(L,L)
PRINT *,  '**********  TEST_JACOBI   **********'
DO IT  =  1,  ITMAX
       DO  J  =  2, L-1
         DO  I  =  2, L-1
             A(I, J)  =  B(I, J)
         ENDDO
       ENDDO
       DO  J = 2,  L-1
         DO  I = 2,  L-1
             B(I, J) =  (A(I-1, J) + A(I, J-1) +
*                  A(I+1, J) + A(I, J+1)) / 4
         ENDDO
       ENDDO
ENDDO
END
```

Параллельная программа на языке Fortran 77 + MPI

```
PROGRAM    JAC_MPI
include 'mpif.h'
integer  me, nprocs
PARAMETER (L=8,  ITMAX=20, LC=2, LR=2)
```

```
      REAL A(0:L/LR+1,0:L/LC+1), B(L/LR,L/LC)
C     arrays A and B  with block distribution
      integer dim(2), coords(2)
      logical isper(2)
      integer status(MPI_STATUS_SIZE,4), req(8), newcomm
      integer srow,lrow,nrow,scol,lcol,ncol
      integer pup,pdown,pleft,pright
      dim(1) = LR
      dim(2) = LC
      isper(1) = .false.
      isper(2) = .false.
      reor = .true.
      call MPI_Init( ierr )
      call MPI_Comm_rank( mpi_comm_world, me, ierr )
      call MPI_Comm_size( mpi_comm_world, nprocs, ierr)
      call MPI_Cart_create(mpi_comm_world,2,dim,isper,
     *  .true., newcomm, ierr)
      call MPI_Cart_shift(newcomm,0,1,pup,pdown, ierr)
      call MPI_Cart_shift(newcomm,1,1,pleft,pright, ierr)
      call MPI_Comm_rank (newcomm, me, ierr)
      call MPI_Cart_coords(newcomm,me,2,coords, ierr)
C   rows of matrix I have to process
      srow = (coords(1) * L) / dim(1)
      lrow = (((coords(1) + 1) * L) / dim(1))-1
      nrow = lrow - srow + 1
C   colomns of matrix I have to process
      scol = (coords(2) * L) / dim(2)
      lcol = (((coords(2) + 1) * L) / dim(2))-1
      ncol = lcol - scol + 1
      call MPI_Type_vector(ncol,1,nrow+2,MPI_DOUBLE_PRECISION,
     * vectype, ierr)
      call MPI_Type_commit(vectype, ierr)
      IF (me .eq. 0) PRINT *,  '*****  TEST_JACOBI   *******'
      DO  IT  =  1,  ITMAX
            DO  J  =  1, ncol
              DO  I  =  1, nrow
                  A(I, J)  =  B(I, J)
              ENDDO
            ENDDO
C     Copying shadow elements of array A from
C     neighboring processors before loop execution
      call MPI_Irecv(A(1,0),nrow,MPI_DOUBLE_PRECISION,
     *      pleft, 1235, MPI_COMM_WORLD, req(1), ierr)
      call MPI_Isend(A(1,ncol),nrow,MPI_DOUBLE_PRECISION,
     *      pright, 1235, MPI_COMM_WORLD,req(2), ierr)
      call MPI_Irecv(A(1,ncol+1),nrow,MPI_DOUBLE_PRECISION,
     *      pright, 1236, MPI_COMM_WORLD, req(3), ierr)
      call MPI_Isend(A(1,1),nrow,MPI_DOUBLE_PRECISION,
     *      pleft, 1236, MPI_COMM_WORLD,req(4), ierr)
      call MPI_Irecv(A(0,1),1,vectype,
     *      pup, 1237, MPI_COMM_WORLD, req(5), ierr)
      call MPI_Isend(A(nrow,1),1,vectype,
     *      pdown, 1237, MPI_COMM_WORLD,req(6), ierr)
      call MPI_Irecv(A(nrow+1,1),1,vectype,
     *      pdown, 1238, MPI_COMM_WORLD, req(7), ierr)
      call MPI_Isend(A(1,1),1,vectype,
     *      pup, 1238, MPI_COMM_WORLD,req(8), ierr)
      call MPI_Waitall(8,req,status, ierr)
            DO  J = 2,  ncol-1
              DO  I = 2,  nrow-1
                  B(I, J) =  (A( I-1, J ) + A( I, J-1 ) +
     *                        A( I+1, J) +  A( I, J+1 )) / 4
              ENDDO
            ENDDO
      ENDDO
      call MPI_Finalize(ierr)
      END
```

Параллельная программа на языке HPF

```
      PROGRAM    JAC_HPF
      PARAMETER    (L=8,  ITMAX=20)
```

```
      REAL     A(L,L), B(L,L)
!HPF$  PROCESSORS P(3,3)
!HPF$  DISTRIBUTE     ( BLOCK,   BLOCK)   ::   A
!HPF$  ALIGN  B(I,J)  WITH  A(I,J)
C      arrays A and B  with block distribution
      PRINT *,  '**********  TEST_JACOBI   **********'
      DO IT  =  1,  ITMAX
!HPF$     INDEPENDENT
            DO  J  =  2, L-1
!HPF$       INDEPENDENT
              DO  I  =  2, L-1
                   A(I, J)  =  B(I, J)
              ENDDO
            ENDDO
!HPF$     INDEPENDENT
            DO  J = 2,  L-1
!HPF$       INDEPENDENT
              DO  I = 2,  L-1
                   B(I, J) =  (A(I-1, J) + A(I, J-1) +
     *                        A(I+1, J) + A(I, J+1)) / 4
              ENDDO
            ENDDO
      ENDDO
      END
```

Параллельная программа на языке Fortran DVM

```
      PROGRAM    JAC_DVM
      PARAMETER    (L=8,  ITMAX=20)
      REAL     A(L,L), B(L,L)
CDVM$  DISTRIBUTE     ( BLOCK,   BLOCK)   ::   A
CDVM$  ALIGN  B(I,J)  WITH  A(I,J)
C      arrays A and B  with block distribution
      PRINT *,  '**********  TEST_JACOBI   **********'
      DO IT  =  1,  ITMAX
CDVM$     PARALLEL  (J,  I)   ON  A(I,  J)
            DO  J  =  2, L-1
              DO  I  =  2, L-1
                   A(I, J)  =  B(I, J)
              ENDDO
            ENDDO
CDVM$     PARALLEL  (J,  I) ON B(I,  J),SHADOW_RENEW (A)
C         Copying shadow elements of array A from
C         neighboring processors before loop execution
            DO  J = 2,  L-1
              DO  I = 2,  L-1
                   B(I, J) =  (A( I-1, J ) + A( I, J-1 ) +
     *                        A( I+1, J ) + A( I, J+1 )) / 4
              ENDDO
            ENDDO
      ENDDO
      END
```