САМАРСКИЙ НАУЧНЫЙ ЦЕНТР
РОССИЙСКОЙ АКАДЕМИИ НАУК

САМАРСКИЙ ГОСУДАРСТВЕННЫЙ АЭРОКОСМИЧЕСКИЙ УНИВЕРСИТЕТ
имени академика С.П.КОРОЛЕВА

ИНСТИТУТ СИСТЕМ ОБРАБОТКИ ИЗОБРАЖЕНИЙ
РОССИЙСКОЙ АКАДЕМИИ НАУК

Попов С.Б., Скуратов С.А., Фурсов В.А.

НАЧАЛЬНЫЕ СВЕДЕНИЯ ДЛЯ РАБОТЫ НА ВЫЧИСЛИТЕЛЬНОМ КЛАСТЕРЕ

Методические указания
для курсового и дипломного проектирования
и учебно-исследовательской работы студентов

САМАРА 2003

УДК 681.324.006.3

Начальные сведения для работы на вычислительном кластере: *Попов С.Б., Скуратов С.А., Фурсов В.А.* Самар. научный центр РАН, Самар. гос. аэрокосм. ун-т. Самара. 2003. 20 с.

В методических указаниях приведены сведения об архитектуре кластера и порядке доступа к его ресурсам. Приведены минимальные сведения о библиотеке MPI (Message Passing Interface – Интерфейс Передачи Сообщений) и примеры ее использования, позволяющие разрабатывать несложные параллельные приложения, в которых реализуется передача сообщений.

Методические указания могут быть полезны при проведении прикладных научных исследований, выполнении курсовых и дипломных проектов. Подготовлено и издано по заказу и при финансовой поддержке СНЦ РАН в рамках Федеральной целевой программы “Интеграция науки и высшего образования России на 2002-2006 годы” и Американского фонда гражданских исследований и развития в рамках российско-американской Программы «Фундаментальные исследования и высшее образование», а также РФФИ (гранты № 01-01-00097, 00-01-05001, 03-01-00109).

Табл. 2. Ил. 5. Библиогр.: 5 наим.

Печатается по решению издательского совета Самарского государственного аэрокосмического университета

Рецензенты: д.т.н., профессор Коварцев А.Н.,
к.т.н., доцент Кравчук В.В.

Введение

В последние годы усиливается внимание к использованию высокопроизводительной вычислительной техники в научных исследованиях и образовании. Связано это с тем, что во многих областях знаний фундаментальные научные исследования связаны с необходимостью проведения масштабных численных экспериментов. Общая тенденция состоит в том, что происходит концентрация вычислительных ресурсов в центрах коллективного пользования и развитие инфраструктуры удаленного доступа с использованием средств телекоммуникаций.

Построение параллельных вычислительных процессов, обеспечивающих достижение высокой производительности, является самостоятельной проблемой. Известно, что перенос обычной программы на многопроцессорную вычислительную систему может не дать ожидаемого выигрыша в производительности. Более того, в результате такого переноса программа может работать медленнее, чем на компьютере с традиционной архитектурой. В то же время изменением первоначальной математической формулировки задачи часто удается добиться значительного эффекта. По-видимому, «развитие математики в определенный период времени отражает природу вычислительных средств, доступных в этот момент, в значительно большей степени, чем можно было бы предположить» [1, стр.152].

Эта точка зрения находит все большее число подтверждений по мере того, как суперкомпьютеры становятся обычным инструментом в руках математиков. Настоящее методическое пособие задумано, чтобы способствовать этому процессу. В частности, мы ставили задачу помочь широкому кругу пользователей быстро войти в круг проблем, связанных с решением задач на ресурсах Самарского регионального центра высокопроизводительной обработки информации в режиме удаленного доступа. Поэтому мы стремились сделать его максимально компактным и доступным. Это предопределило отбор материала. В пособие вошли лишь самые необходимые сведения, которые, тем не менее, в большинстве случаев оказываются достаточными при подготовке, отладке и запуске параллельных программ.

Методические указания ориентированы на использование в ходе курсового и дипломного проектирования, но может быть полезно также научным работникам при подготовке параллельных приложений.

1. Архитектура кластера

Вычислительный кластер Самарского центра высокопроизводительной обработки информации (ЦВОИ) представляет собой 24-хпроцессорную систему с укрупненным распараллеливанием (MPP-система), в узлах которой используются двухпроцессорные компьютеры SMP-архитектуры (симметричные мультипроцессорные системы). Вычислительная система состоит из 8 двухпроцессорных компьютеров на базе процессоров Pentium-II и 4 двухпроцессорных компьютеров на базе процессоров Alpha 21264/677MHz, связанных высокоскоростной коммуникационной средой Myrinet. Вследствие наличия в составе вычислительного ядра процессоров с различной разрядностью (32-разрядные процессоры Pentium-II и 64-х разрядные процессоры Alpha 21264) данная вычислительная система относится к классу гетерогенных. Это создает определенные трудности обмена данными между узлами, содержащими разнотипные процессоры. Поэтому в настоящее время принята схема раздельного использования разнотипных кластеров. Это означает, что пользователь может по своему усмотрению решать задачу либо на 16-ти процессорном Pentium-кластере (8 двухпроцессорных PC SMP Pentium), либо на 8-процессорном Alpha- кластере (4 двухпроцессорные рабочие станции Alpha).

Основные характеристики используемой на кластере коммуникационной среды Myrinet: полнодуплексные 1.28 + 1.28 Гбит/сек линки и коммутируемые порты; управление потоком и контроль ошибок на каждом линке; низкая латентность (малое время задержки). Построенная на базе коммутатора сеть Myrinet может одновременно передавать несколько пакетов, каждый – со скоростью 1.28 Гбит/с. В отличие от некоммутированных Ethernet-сетей, которые разделяют общую среду передачи, совокупная пропускная способность сети Myrinet возрастает с увеличением количества узлов.

Кроме высокоскоростной сети Myrinet для связи между узлами используется также служебная сеть FastEthernet. Эта сеть может использоваться также и для обмена данными между параллельными процессами. Однако это целесообразно лишь на Pentium-составляющей кластера при решении задач, в которых интенсивность обмена данными невысока. При решении задач на Alpha-кластере время, затрачиваемое на проведение собственно вычислений, намного меньше, что приведет к существенному относительному возрастанию накладных расходов и снижению эффективности его использования.

2. Порядок доступа к ресурсам кластера

Доступ к ресурсам объединенного кластера Самарского центра высокопроизводительной обработки информации может осуществляться с любого персонального компьютера, подключенного к сети Internet. Для получения доступа необходимо пройти процедуру регистрации и представить данные о решаемой задаче.

Доступ сотрудников академических организаций, а также преподавателей и студентов государственных вузов к ресурсам ЦВОИ осуществляется без ограничений по «Заявке на предоставление доступа к ресурсам Центра высокопроизводительной обработки информации СНЦ РАН» установленной формы (рис. 1). Порядок доступа к ресурсам приведен на сайте Самарского научного центра РАН (http://www.ssc.smr.ru), а саму заявку можно скопировать по адресу http://www.ssc.smr.ru/scomp/Open_Form_2.doc.

Для решения вопроса о допуске к ресурсам Центра пользователь должен заполнить заявку, заверить ее печатью организации и лично прибыть в Центр (г. Самара, Студенческий пер., д. 3-а, СНЦ РАН, комн. № 8) для получения первичного пароля и ознакомления с правилами работы на его ресурсах. Для сотрудников коммерческих вузов и организаций Самарской области регистрация проводится после заключения договора с Самарским научным центром РАН, регламентирующим порядок доступа и ответственность пользователей. Сроки предоставления ресурсов пользователям также регламентируются договором.

Персонал Центра обеспечивает поддержку пользователей, которая включает в себя:

-необходимые изменения в режиме работы;
-изменения в конфигурации технических средств;
-изменения в программном обеспечении;
-график проведения регламентных работ.

Пользователи, прошедшие регистрацию и получившие доступ к ресурсам Центра, имеют возможность работать круглосуточно за исключением времени, отводимого на регламентные и профилактические работы, о чем пользователи извещаются по электронной почте.

Заявка на предоставление доступа к ресурсам
Центра высокопроизводительной обработки информации СНЦ РАН

Исх № _____ Вх. № _____
(присваивается организацией) (присваивается в СНЦ РАН)

Прошу предоставить доступ до "____" _____ 200_ г. пользователю:

ФИО (полностью) _____
_____
Должность _____
Организация _____
Контактный телефон_____
Адрес _____
_____
E-mail _____

Руководитель организации _____

"_____"_____ 200__ г.

М.П.

З а п о л н я е т с я в С Н Ц Р А Н

С правилами работы и сроками предоставления ресурсов ознакомлен:

Пользователь "___" _____ 200__г.

Разрешается допуск к ресурсам до **"_____"_____ 200 г.**

Основание: _____

Нач. отдела высокопроизводительной
обработки информации СНЦ РАН **(_____)**

"___" _____ 200__г.

Доступ к ресурсам предоставлен:
Системный администратор **(_____)**

"___" _____ 200__г.

Пароль получил:
login name "_____"

Пользователь

"___" _____ 200__г.

Рис. 1. Форма заявки на предоставление доступа к ресурсам Центра высокопроизводительной обработки информации СНЦ РАН.

3. Рекомендации по созданию параллельных приложений

Обычной является ситуация, когда пользователь намерен с наименьшими усилиями перенести имеющуюся у него последовательную программу на кластер. Понятно, что, просто оттранслировав эту программу на кластере, ускорения, скорее всего, достичь не удастся. Тем не менее, это полезно для отработки технологии удаленных вычислений, описанной в разделе 5 настоящего пособия. После того как работа на кластере в режиме удаленного доступа освоена, можно попытаться ускорить работу программы. Для этого, обычно, требуется «ручная» переработка последовательной программы.

Усилия, затрачиваемые на эту переработку, в значительной степени зависят от типа решаемой задачи. С точки зрения распараллеливания вычислений задачи можно разбить на следующие два класса: допускающие распараллеливание по данным (*параллелизм данных*), и процедуры, в которых имеет место лишь *параллелизм задач* [2]. Важной отличительной особенностью задач, допускающих распараллеливание по данным, является то, что одна операция или совокупность операций выполняются над всеми элементами массива данных. Это существенно облегчает переработку последовательной программы. В этом случае задача может свестись к разбиению массива исходных данных на фрагменты, обработка которых ведется независимо на различных процессорах. Если производительность процессоров различна, это надо учитывать при выборе размеров фрагментов, так чтобы обеспечивалась их равномерная загрузка. Для этого можно воспользоваться технологией последовательного планирования распределения ресурсов кластера, описанной в работе [3].

Технология заключается в попеременном запуске параллельной программы, реализующей прикладную задачу, и программы планирования ресурсов. Программа планирования ресурсов анализирует эффективность выполнения прикладной программы на предыдущих запусках и формирует новый план распределения процессоров. Поскольку решение принимается на основании результатов реального исполнения задачи, то автоматически учитываются все временные затраты (в том числе на пересылку сообщений).

Для реализации указанной технологии прикладная программа должна быть соответствующим образом подготовлена. В частности, необходимо каким-либо образом задать начальный план распределения ресурсов для одного или нескольких первых запусков программы. Программа планирования ресурсов функционирует независимо и, вообще говоря, не требует проведения специальных испытаний прикладной программы, направленных на оптимизацию ресурсов. Целесообразно решение задачи распределения ресурсов совместить с завершающим этапом отладки программы, когда осуществляются пробные запуски программы при различных исходных данных. Консультации по применению технологии можно получить у сотрудников центра высокопроизводительной обработки информации СНЦ РАН.

Если при распараллеливании по данным необходимо осуществлять обмен данными между параллельно работающими процессорами, эффективность программы будет зависеть от соотношения временных затрат на проведение вычислений на фрагментах исходных данных и пересылку данных (накладные расходы) [4]. Интуитивно кажется, чем на большее число фрагментов (каждый из которых обрабатывается отдельным процессором) будет разбит исходный массив данных, тем меньшее время будет выполняться программа. В действительности это не так. По мере увеличения числа (а значит уменьшения размеров) фрагментов, объем вычислений на каждом фрагменте уменьшается. При этом накладные расходы могут оставаться прежними, как в силу особенностей задачи, так и вследствие латентности (связанной с потерями на передачу сообщений нулевой длины) коммуникационной среды.

Можно рекомендовать следующий простой способ построения эффективной программы, основанной на свойстве параллелизма данных. Размеры фрагментов массива исходных данных следует уменьшать (соответственно увеличивать число параллельно работающих процессоров) до тех пор, пока имеет место линейное ускорение. Если при увеличении числа процессоров линейного ускорения не происходит, это означает, что накладные расходы стали заметными и дальнейшее распараллеливание по данным приведет к недостаточной загрузке процессоров.

Если задача не допускает распараллеливания по данным, т.е. возможен лишь *параллелизм задач*, трудности существенно возрастают. Подход к программированию, основанный на параллелизме задач, подразумевает, что вычислительная задача разбивается на несколько относительно самостоятельных подзадач, и каждый процессор загружается своей собственной подзадачей. Для каждой подзадачи пишется своя собственная программа. Чем больше подзадач, тем большее число процессоров можно использовать и тем большего ускорения можно ожидать (если удастся обеспечить равномерную загрузку процессоров и минимизировать обмен данными между ними).

Для построения эффективного кода в этом случае необходимо проводить анализ затрачиваемого времени разными частями программы с целью выявления наиболее ресурсопотребляющих частей. При этом могут использоваться различные формальные модели: пространственно-временные диаграммы, модели в виде ориентированных графов и сетей Петри и др. Для детального изучения основных подходов и методов решения этих задач можно рекомендовать работу ведущих российских ученых в области параллельных вычислений Воеводина В.В. и Воеводина Вл. В. [2] или учебное пособие [4].

4. Использование библиотеки MPI для создания параллельных приложений

Наиболее распространенный подход к параллельному программированию на MPP-системах (massively parallel processing systems – системах с укрупнено распараллеленной обработкой) заключается в следующем. На каждом компьютере параллельной вычислительной системы (например, кластерной архитектуры) запускается программа, написанная на стандартном языке последовательного программирования. Каждая такая программа либо обрабатывает свою порцию данных, либо выполняет свою операцию обработки. Взаимодействие этих программ обеспечивается путем вызова функций синхронизации, посылки и получения сообщений, которые объединены в специальную библиотеку.

Фактическим стандартом параллельного программирования с использованием такого подхода сейчас является библиотека MPI (Message Passing Interface – Интерфейс Передачи Сообщений).

MPI-программа представляет собой программу, которая запускается одновременно на нескольких процессорах. Каждая копия программы выполняется как отдельный процесс. С помощью служебных функций MPI процесс может получить информацию о количестве одновременно запущенных процессов и свой номер. Таким образом, в одной и той же программе в зависимости от номера процесса может выполняться различный код, или же код программы параметризуется номером процесса. Коммуникационные функции MPI предоставляют процессам MPI-программы различные способы взаимодействия. Возможны как индивидуальные, так и групповые операции обмена данными.

Для запуска MPI-программы используется команда `mpirun` (специальный командный файл – скрипт), при этом указывается количество запускаемых процессов и имя файла исполняемого модуля (программы). Дополнительно можно задать конфигурацию (список компьютеров) на которой запускается программа. Количество процессов не обязательно равно количеству процессоров. Возможен запуск MPI-программы с количеством процессов большим, чем число доступных процессоров, при этом несколько процессов должны одновременно выполняться на одном процессоре. Выполнение программы с указанием запуска нескольких процессов на однопроцессорном компьютере позволяет производить первоначальную отладку параллельного приложения.

При инициализации MPI создается предопределенная область связи, содержащая все процессы MPI-программы, с которой связывается предопределенный коммуникатор `MPI_COMM_WORLD`. В большинстве случаев именно он используется в качестве коммуникатора при задании параметров функций MPI. Используемые здесь и далее понятия *область связи* и *коммуникатор области связи*, подробно рассматривались в [5]. При начальном знакомстве с библиотекой MPI достаточно знать, что они автоматически создаются при запуске любой программы.

Предварительно рассмотрим основные соглашения по использованию библиотеки MPI в языках программирования Си и Фортран.

Регистр символов при указании имен функций (процедур) и именованных констант существенен в Си, и не играет роли в Фортране. Все идентификаторы начинаются с префикса `MPI_`. Не рекомендуется использовать собственные идентификаторы, начинающиеся с этого префикса, а также с префиксов `MPID_`, `MPIR_` и `PMPI_`, которые применяются в служебных целях. Имена констант (и неизменяемых пользователем переменных) записываются полностью заглавными буквами: `MPI_COMM_WORLD, MPI_FLOAT`. В именах функций первая за префиксом буква – заглавная, остальные маленькие: `MPI_Send`, `MPI_Comm_size`. Определение всех именованных констант, прототипов функций и определение типов выполняется в языке Си подключением файла `mpi.h`, а в Фортране – `mpif.h`.

В операциях пересылки и преобразовании данных необходимо указывать собственные типы MPI. Это связано как с обеспечением переносимости программ между различными компиляторами и платформами, так и с возможностью запуска параллельного приложения на гетерогенных вычислительных системах. В последнем случае MPI обеспечивает автоматическое преобразование типов данных при пересылках.

Список предопределенных типов для Си-версии библиотеки MPI приводится в таблице 1, а для языка Фортран – в таблице 2..

Таблица 1

Соответствие между MPI-типами и типами языка Си

тип MPI	*тип языка Си*
MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_BYTE	
MPI_PACKED	

Таблица 2

Соответствие между MPI-типами и типами языка Фортран

тип MPI	*тип языка Фортран*
MPI_INTEGER	INTEGER
MPI_REAL	REAL
MPI_DOUBLE_PRECISION	DOUBLE PRECISION
MPI_COMPLEX	COMPLEX
MPI_LOGICAL	LOGICAL
MPI_CHARACTER	CHARACTER(1)
MPI_BYTE	
MPI_PACKED	

Тип MPI_BYTE используется для передачи двоичной информации без какого-либо преобразования. Тип MPI_PACKED используется для передачи в одном сообщении разнотипных данных, предварительно упакованных специальными функциями. Кроме того, программисту предоставляются средства создания собственных типов на базе стандартных.

Изучение MPI традиционно начнем с создания простейшей программы, которая сообщает о себе окружающему миру, т.е. каждый запускаемый процесс выдает следующее сообщение:

```
Hello world from process i of n
```

Здесь i – номер процесса, а n – количество процессов.

Используя редактор, набираем текст программы helloworld.c:

```
#include <stdio.h>
#include "mpi.h"

int main( argc, argv )
int  argc;
char **argv;
{
    int rank, size;
    MPI_Init( &argc, &argv );
    MPI_Comm_size( MPI_COMM_WORLD, &size );
    MPI_Comm_rank( MPI_COMM_WORLD, &rank );
    printf( "Hello world from process %d of %d\n", rank, size );
    MPI_Finalize();
    return 0;
}
```

В этой простой программе используется 4 функции MPI. Далее будет введено еще только 2 функции. В настоящем пособии, в соответствии с поставленной задачей, мы намеренно ограничимся изучением лишь 6 функций MPI, достаточных для написания простейших параллельных программ. Для более полного знакомства с библиотекой MPI можно рекомендовать пособие [5].

Любая прикладная MPI-программа (приложение) должна начинаться с вызова функции инициализации MPI (функция MPI_Init). В результате выполнения этой функции создается группа процессов, в которую помещаются все процессы приложения, и создается область связи, описываемая предопределенным коммуникатором MPI_COMM_WORLD. Эта область связи объединяет все процессы приложения. Процессы в группе упорядочены и пронумерованы от 0 до groupsize-1, где groupsize равно числу процессов в группе. В нашем случае величина groupsize равна числу процессоров, выделенных задаче.

Синтаксис функции инициализации **MPI_Init** значительно отличается в языках Си и Фортран:

C
```
int MPI_Init(int *argc, char ***argv)
```
FORTRAN
```
INTEGER IERROR
MPI_INIT(IERROR)
```

В программах на Си каждому процессу при инициализации передаются аргументы функции `main`, полученные из командной строки. В программах на языке Фортран параметр `IERROR` является выходным и возвращает код ошибки.

Функция завершения MPI-программ **`MPI_Finalize`**. Функция закрывает все MPI-процессы и ликвидирует все области связи.

```
C        int MPI_Finalize(void)
FORTRAN  INTEGER IERROR
         MPI_FINALIZE(IERROR)
```

Функция определения числа процессов в области связи **`MPI_Comm_size`**.

```
C        int MPI_Comm_size(MPI_Comm comm, int *size)
FORTRAN  INTEGER COMM, SIZE, IERROR
         MPI_COMM_SIZE(COMM, SIZE, IERROR)
```

Параметры функции `MPI_Comm_size`:

IN `comm` – коммуникатор;
OUT `size` – число процессов в области связи коммуникатора `comm`.

(здесь и далее при обсуждении параметров процедур символами **IN** будем указывать входные параметры процедур, символами **OUT** выходные, а **INOUT** – входные параметры, модифицируемые процедурой).

Функция возвращает количество процессов в области связи коммуникатора `comm`. До создания явным образом групп и связанных с ними коммуникаторов единственно возможными значениями параметра `COMM` являются `MPI_COMM_WORLD` и `MPI_COMM_SELF`, которые создаются автоматически при инициализации MPI.

Функция определения номера процесса **`MPI_Comm_rank`**.

```
C        int MPI_Comm_rank(MPI_Comm comm, int *rank)
FORTRAN  INTEGER COMM, RANK, IERROR
         MPI_COMM_RANK(COMM, RANK, IERROR)
```

Параметры функции `MPI_Comm_rank`:

IN `comm` – коммуникатор;
OUT `rank` – номер процесса, вызвавшего функцию.

Функция возвращает номер процесса, вызвавшего эту функцию. Номера процессов лежат в диапазоне `0..size-1` (значение `size` может быть определено с помощью предыдущей функции).

Теперь необходимо откомпилировать, скомпоновать и запустить программу **`helloworld`**, например, на 4 процессорах.

```
% mpicc -o helloworld helloworld.c
% mpirun -np 4 helloworld
Hello world from process 0 of 4
Hello world from process 3 of 4
Hello world from process 1 of 4
Hello world from process 2 of 4
%
```

Заметим, что порядок появления сообщений от различных процессов не определен. Если появляется какое-либо сообщение об ошибке, например:

```
mpicc: Command not found.
```
то, скорее всего, необходимо в переменной окружения PATH указать каталог, где находятся исполняемые файлы библиотеки MPI. Например,

```
setenv PATH /usr/local/mpi/bin:$PATH
rehash
```

Далее, для того чтобы завершить рассмотрение всех шести функций MPI, которыми мы решили ограничиться в настоящем пособии, создадим программу, которая рассылает некоторое сообщение (данные) по цепочке запущенных процессов (рис. 2).

Рис. 2.

Данные состоят из одного целочисленного значения, которое процесс 0 получает от пользователя. Признаком завершения рассылки является ввод отрицательного числа.

Текст программы **`ring.c`**:

```
#include <stdio.h>
#include "mpi.h"

int main( argc, argv )
int argc;
char **argv;
{
    int rank, value, size;
    MPI_Status status;

    MPI_Init( &argc, &argv );

    MPI_Comm_rank( MPI_COMM_WORLD, &rank );
    MPI_Comm_size( MPI_COMM_WORLD, &size );
    do {
     if (rank == 0) {
         scanf( "%d", &value );
         MPI_Send(&value, 1, MPI_INT, rank + 1, 0, MPI_COMM_WORLD);
     }
     else {
         MPI_Recv(&value, 1, MPI_INT, rank - 1, 0, MPI_COMM_WORLD,
                &status);
         if (rank < size - 1)
          MPI_Send(&value, 1, MPI_INT, rank+1, 0, MPI_COMM_WORLD);
     }
     printf( "Process %d got %d\n", rank, value );
    } while (value >= 0);

    MPI_Finalize( );
    return 0;
}
```

В приведенной программе используются все 6 функций MPI, которые изучаются в настоящем пособии. Четыре из них уже использовались нами в примере на странице 11. Поэтому подробно рассмотрим лишь две из них (MPI_Send, MPI_Recv), которые введены в настоящем примере впервые.

Функция передачи сообщения **MPI_Send**.

```
C       int MPI_Send(void* buf, int count, MPI_Datatype datatype,
                     int dest, int tag, MPI_Comm comm)
```

```
FORTRAN  <type> BUF(*)
         INTEGER COUNT, DATATYPE, DEST, TAG, COMM, IERROR
         MPI_SEND(BUF, COUNT, DATATYPE, DEST, TAG, COMM, IERROR)
```

Параметры функции MPI_Send:

IN	buf	–	адрес начала расположения пересылаемых данных;
IN	count	–	число пересылаемых элементов;
IN	datatype	–	тип посылаемых элементов;
IN	dest	–	номер процесса-получателя в группе, связанной с коммуникатором comm;
IN	tag	–	идентификатор сообщения (очень часто является порядковым номером сообщения в последовательности);
IN	comm	–	коммуникатор области связи.

Функция выполняет посылку count элементов типа datatype сообщения с идентификатором tag процессу dest в области связи коммуникатора comm. Переменная buf – это, как правило, массив или скалярная переменная. В последнем случае значение count = 1.

Функция приема сообщения **MPI_Recv**.

```
C       int MPI_Recv(void* buf, int count, MPI_Datatype datatype,
                     int source, int tag, MPI_Comm comm,
                     MPI_Status *status)
```

```
FORTRAN  <type> BUF(*)
         INTEGER COUNT, DATATYPE, SOURCE, TAG, COMM,
                 STATUS(MPI_STATUS_SIZE), IERROR
         MPI_RECV(BUF, COUNT, DATATYPE, SOURCE, TAG, COMM,
                 STATUS, IERROR)
```

Параметры функции MPI_Recv:

OUT	buf	–	адрес начала расположения принимаемого сообщения;
IN	count	–	максимальное число принимаемых элементов;
IN	datatype	–	тип элементов принимаемого сообщения;
IN	source	–	номер процесса-отправителя в группе, связанной с коммуникатором comm;
IN	tag	–	идентификатор сообщения;
IN	comm	–	коммуникатор области связи;
OUT	status	–	атрибуты принятого сообщения.

Функция выполняет прием count элементов типа datatype сообщения с идентификатором tag от процесса source в области связи коммуникатора.

В функции MPI_Recv при указании номера процесса-отправителя возможно использование специального параметра MPI_ANY_SOURCE ("принимай от кого угодно"), а в качестве идентификатора получаемого сообщения –

MPI_ANY_TAG ("принимай что угодно"). Это так называемые параметры-джокеры, MPI резервирует для них отрицательные целые числа, в то время как реальные идентификаторы процессов и сообщений лежат всегда в диапазоне от 0 до 32767. Пользоваться джокерами следует с осторожностью, потому что по ошибке таким вызовом MPI_Recv может быть захвачено сообщение, которое должно приниматься в другой части процесса-получателя.

Если логика программы достаточно сложна, использовать джокеры можно **только** в функциях проверяющих наличие сообщения для процесса (MPI_Probe и MPI_Iprobe), чтобы перед фактическим приемом узнать тип и количество данных в поступившем сообщении. Несмотря на то, что мы хотим получить "что угодно", тип принимаемых данных в функции MPI_Recv должен быть указан явно, а он может быть разным в сообщениях с разными идентификаторами.

Пример запуска программы ring.c приводится ниже.

```
% mpicc -o ring ring.c
% mpirun -np 4 ring
10
Process 0 got 10
-1
Process 0 got -1
Process 3 got 10
Process 3 got -1
Process 2 got 10
Process 2 got -1
Process 1 got 10
Process 1 got -1
%
```

Описанные выше функции реализуют *стандартный режим с блокировкой.* Следует заметить, что возврат из блокирующей функции передачи данных еще не означает, что передача завершена, гарантируется только возможность повторного использования буфера данных для внесения в него изменений, которые уже не повлияют на отправляемые данные.

В коммуникационных операциях типа точка-точка всегда участвуют не более двух процессов: передающий и принимающий. В MPI имеется множество функций, реализующих такой тип обменов. В дополнение к стандартному режиму возможно использование синхронной, буферизованной или согласованной передачи, как с блокировкой, так и без блокировки. С помощью только пары операций MPI_Send/MPI_Recv возможно реализовать практически любой параллельный алгоритм.

Пользователи, освоившие работу с описанными в настоящем пособии шестью функциями MPI, могут попытаться построить более эффективный параллельный код с использованием так называемых коллективных функций MPI. Подробное описание коллективных функций и примеры параллельных программ, в которых они используются, можно найти в учебном пособии [5]. В заключение еще раз подчеркнем, что для написания многих параллельных программ, обычно достаточно рассмотренных в настоящем пособии шести функций.

5. Запуск параллельных приложений на кластере

Доступ к ресурсам кластеров ИСОИ РАН и СНЦ РАН может осуществляться с любого персонального компьютера, подключенного к сети Internet. При работе на кластерах в режиме удаленного доступа используются протокол ssh и (необязательно) FTP. Поэтому требования к аппаратуре определяются установленным программным обеспечением, реализующим работу по указанным протоколам, а также дополнительным программным обеспечением, установленным в целях повышения эффективности работы с кластером и снижения расходов на разработку собственного программного обеспечения.

Кластер работает под управлением операционной системы Linux (RedHat 6.2). Поэтому в качестве операционной системы удаленной машины также рекомендуется использовать Linux. Это представляется целесообразным по следующим причинам:

1. Эти системы являются свободно распространяемыми.
2. Указанные операционные системы являются UNIX-системами и обладают схожим набором команд (по крайней мере, в объеме, необходимом при работе с кластером).
3. В них установлены аналогичные наборы свободно распространяемых трансляторов C/C++/FORTRAN77, что гарантирует хорошую переносимость на кластеры программ, разработанных и отлаженных на локальной машине.
4. Операционная система Linux обеспечивает работу в сетевых протоколах ssh и FTP.
5. На них установлен пакет программ MPICH, реализующий интерфейс передачи сообщений (MPI).

Сказанное вовсе не означает, что нельзя использовать другие операционные системы (например, Microsoft Windows). Однако в этом случае при отладке программ на локальном компьютере следует учитывать возможные различия в трансляторах (см. п. 3), а пользователь, скорее всего, будет лишен возможности осуществлять запуск exe-модулей, предварительно отлаженных на локальном персональном компьютере. Однако он может выполнить компиляцию исходного кода программы непосредственно на кластере.

Запуск программ на кластере осуществляется следующим образом. Перед входом на кластер нужно скопировать исходный текст программы в вашу домашнюю директорию на кластере. Для этого в терминале нужно ввести команду:

```
% scp my_program.c name@h1.ssc.smr.ru:~/
```

где `my_program.c` – это исходный текст программы на языке C, `name` – ваше имя, под которым вы зарегистрированы на кластере. При выполнении этой команды будет запрашиваться пароль.

Далее нужно зайти на кластер, для этого в терминале введите команду:

```
% ssh name@h1.ssc.smr.ru
```

При этом необходимо ввести свой пароль, выданный вам системным администратором.

Для входа на кластер выделено два адреса: `h8.ssc.smr.ru` – при входе по данному адресу библиотека MPI по умолчанию в качестве коммуникационной среды использует сеть Myrinet, что накладывает ограничение на количество одновременно выполняемых параллельных приложений (одно на весь кластер, если не использовать специальные настройки), `h1.ssc.smr.ru` – при входе по данному адресу библиотека MPI по умолчанию в качестве коммуникационной среды использует сеть FastEthernet, что позволяет в полном объеме вести отладку и запускать параллельные приложения без оглядки на других пользователей.

После того как вы получили доступ к кластеру, необходимо откомпилировать вашу программу. Для компиляции программы нужно использовать команду `mpicc`:

```
name@h1:~$ mpicc -o my_program my_program.c
```

В результате этого в директории появится файл `my_program`. Это исполняемый файл программы. Для запуска программы нужно использовать команду `mpirun`:

```
name@h1:~$ mpirun -np <NP> ./my_program
```

где `<NP>` – количество процессов. Так для запуска программы с тремя процессами строка будет иметь вид:

```
name@h1:~$ mpirun -np 3 ./my_program
```

При работе в среде Myrinet перед запуском программы полезно убедиться в отсутствии уже запущенных параллельных приложений, что можно сделать командой `gm_local_show_processes.sh`, например:

```
name@h8:~$ gm_local_show_processes.sh
HOST: h8
HOST: h7
HOST: h6
HOST: h5
HOST: h4
HOST: h3
HOST: h2
HOST: h1
name@ h8:~$
```

Снять запущенное приложение, которое использует среду Myrinet, можно командой `gm_local_cleanup_processes.sh`, например:

```
name@h8:~$ mpirun -np 4 ./my_program
. . .
name@h8:~$ gm_local_ cleanup _processes.sh
HOST: h8
HOST: h7
HOST: h6
HOST: h5
HOST: h4
```

```
HOST: h3
HOST: h2
Sending Signal 11 to pid 14807
Sending Signal 9 to pid 14807
Sending Signal 11 to pid 14806
Sending Signal 9 to pid 14806
HOST: h1
Sending Signal 11 to pid 31234
Sending Signal 9 to pid 31234
Sending Signal 11 to pid 31236
Sending Signal 9 to pid 31236
name@ h8:~$
```

Для выхода на кластер с компьютера, работающего на платформе Microsoft Windows по протоколу ssh, можно воспользоваться свободно распространяемым ssh-клиентом, например, PuTTY*, который можно загрузить по ссылке http://the.earth.li/~sgtatham/putty/latest/x86/putty-0.53b-installer.exe .

После установки данного клиента на компьютере, и запуска программы PuTTY, появляется окно конфигурации сеанса (рис.3).

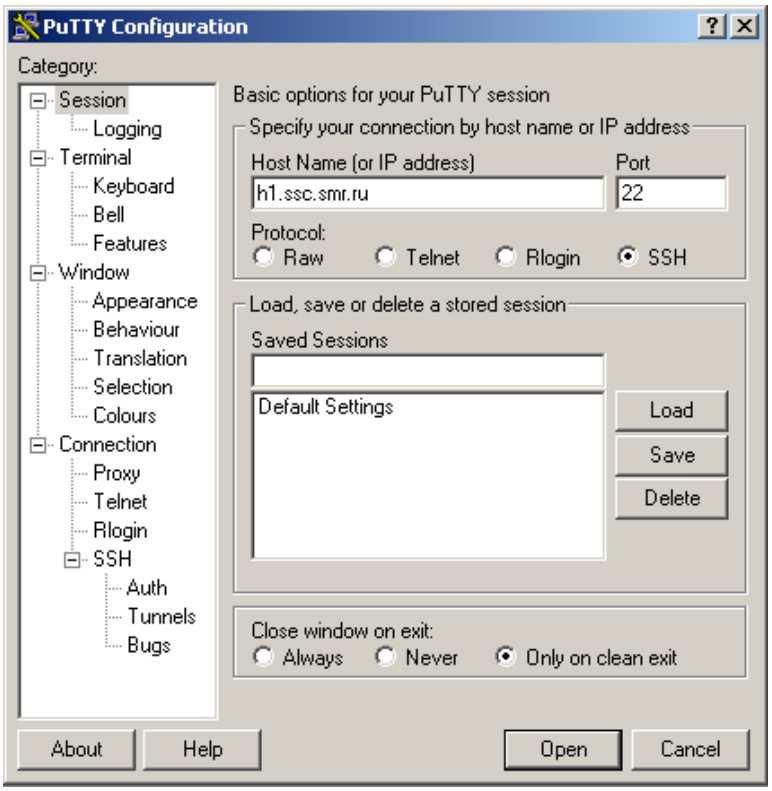

Рис. 3. Начальное окно сеанса программы PuTTY.

В строке Host Name (or IP address) необходимо ввести имя управляющего компьютера на кластере (в нашем случае это: `h1.ssc.smr.ru`)и выбрать протокол доступа (SSH). Далее нажать кнопку Open. При первом запуске появится предложение о сохранении необходимой служебной информации о компьютере, к которому производится подключение, для последующих подключений. Можно выбрать Yes, и данное окно не будет больше появляться. Если компью-

* http://www.chiark.greenend.org.uk/~sgtatham/putty/

тер с указанным именем найден, и он поддерживает протокол ssh, появится окно терминального клиента с приглашением `login as:`, после ввода своего имени будет предложено ввести пароль, после его успешной проверки выдается информационное сообщение от администратора системы и приглашение для ввода команды (рис. 4).

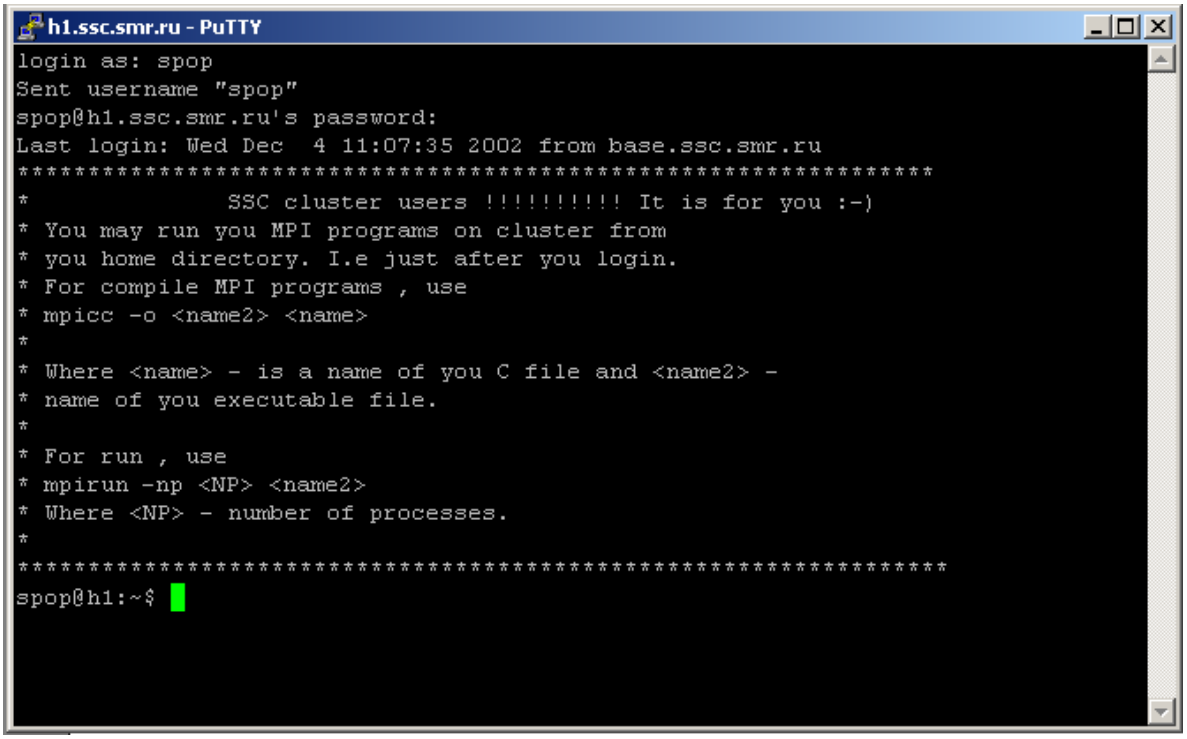

Рис. 4. Окно терминального ssh-клиента после входа в систему.

Для ввода и редактирования текста программ, работы с файлами можно запустить нортоноподобную программу mc (рис. 5).

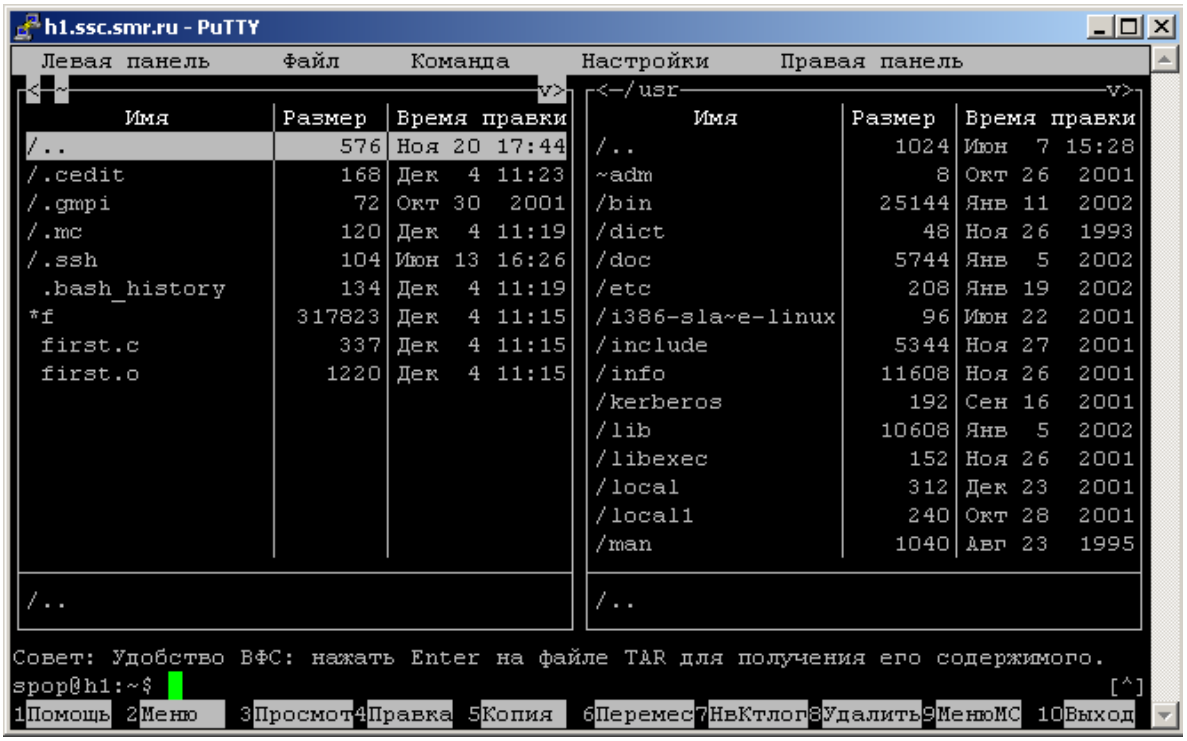

Рис.5. Окно терминального ssh-клиента с запущенной программой `mc`.

Переключение между отображением окна mc и командной строкой производится одновременным нажатием клавиш Ctrl+O. Для создания нового файла текста программы нажмите Shift+F4. В открывшемся окне редактирования можно набирать текст программы, а затем при сохранении файла будет выдан запрос на ввод имени нового файла. При редактировании можно вставлять текст из буфера обмена ОС MS Windows.

Компиляция и запуск MPI-программы на выполнение производится аналогично тому, как это делалось под ОС Linux.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Т. Тоффоли, Н. Марголус. Машины клеточных автоматов, М.: Мир,1991 г.
2. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. СПб.: БХВ-Петербург, 2002 г.
3. Фурсов В.А., Шустов В.А., Скуратов С.А. Технология итерационного планирования распределения ресурсов гетерогенного кластера. // Труды Всероссийской научной конференции "Высокопроизводительные вычисления и их приложения" г. Черноголовка , 30 октября - 2 ноября 2000 г. с.43-46.
4. Головашкин Д.Л. Методы параллельных вычислений. Часть I. – Учебное пособие. Изд. СГАУ, Самара. 2002, 92 с.
5. Введение в программирование для параллельных ЭВМ и кластеров: Учебн. пособие/ Авторы-сост.: Кравчук В.В., Попов С.Б., Привалов А.Ю., Фурсов В.А., Шустов В.А.; Самар. научный центр РАН, Самар. гос. аэрокосм. ун-т. Самара. 2000. 87 с.

Оглавление