Руководство пользователя `mpich`, переносимой реализации MPI, версии 1.2.2

William Gropp Ewing Lusk [1]

[1] Перевод Алексея Отвагина

Содержание

Это руководство пользователя соответствует `mpich` версии 1.2.2.

Аннотация

MPI(Message Passing Interface — интерфейс передачи сообщений) является стандартной спецификацией для библиотек передачи сообщений. `mpich` является переносимой реализацией полной спецификации MPI для широкого множества сред параллельных и распределенных вычислений. Это руководство поясняет, как создавать и запускать программы MPI, используя реализацию `mpich`.

Этот документ описывает использование `mpich` [9], переносимой реализации стандарта передачи сообщений MPI. Детали получения и инсталляции реализации `mpich` представлены в отдельном *руководстве по инсталляции* для `mpich` [6]. Версия 1.2.2 `mpich` в основном исправляет ошибки и увеличивает переносимость, особенно для кластеров на базе LINUX.

Новое в 1.2.2:

- Существенно улучшено устройство `ch_p4mpd`
- Улучшенная поддержка различных компиляторов Фортран и Фортран 90, включая вычисление во время компиляции констант Фортрана, используемых в реализации `mpich`.
- Улучшенное устройство `globus2`, предоставляющее большую производительность.
- Новый режим `bproc` для устройства `ch_p4` поддерживает Scyld Beowulf.
- Множественные улучшения производительности TCP для устройств `ch_p4` и `ch_p4mpd`.
- Множество исправлений и улучшений кода.
  См. `www.mcs.anl.gov/mpi/ mpich/r1_2_2changes.html` о полном списке изменений.

Новые возможности, включенные в 1.2.1:

- Улучшенная поддержка различных компиляторов Фортран и Фортран 90. В частности, одна версия `mpich` может теперь быть построена с использованием нескольких различных компиляторов Фортрана; см. руководство по инсталляции (в doc/install.ps.gz) для пояснений.
- Облегчено использование компилятора C для программ MPI, использующих `mpich`, являющегося отличным от того, с помощью которого строился `mpich`; см. руководство по инсталляции.
- Известные проблемы и ошибки данной реализации документированы в `'mpich/KnownBugs'`.
- Существует раздел ЧаВО (Часто задаваемые ВОпросы) на `http://www.mcs.anl.gov/mpi/mpich/faq.html`. См. сюда, если вы получаете при запуске `mpich` сообщения вида “permission denied”, “connection reset by peer”, “poll: protocol failure in circuit setup”.
- Статья о `jumpshot` доступна на `ftp://ftp.mcs.anl.gov/pub/mpi/jumpshot.ps.gz`. Статья о MPD доступна на `ftp://ftp.mcs.anl.gov/pub/mpd.ps.gz`.

1 Введение

mpich является свободно доступной реализацией стандарта MPI, который работает на множестве вариантов систем. Детали реализации mpich описаны в [9]; статьи по этой теме также включают [7] и [8]. Этот документ предполагает, что mpich уже инсталлирован; если это не так, Вы должны сначала прочесть *руководство по инсталляции* mpich, *переносимой реализации MPI* [5]. Более конкретно, этот документ предполагает, что реализация mpich установлена в '/usr/local/mpich' и что Вы добавили '/usr/local/mpich/bin' в Ваш путь поиска. Если mpich установлен в другом месте, Вы должны сделать необходимые изменения. Если mpich был собран для нескольких различных архитектур и/или механизмов коммуникации (называемых в mpich *устройствами*), Вы должны выбрать каталоги соответствующим образом; сверьтесь с тем, кто установил mpich в Вашей системе.

Основные возможности mpich:

- Полное соответствие стандарту MPI 1.2, включая отмену передачи.
- Привязки C++ к стандарту MPI-2 доступны из функций MPI-1.
- Привязки к Фортрану 77 и Фортрану 90, включая 'mpif.h' и модуль MPI.
- Версия для Windows NT доступна в качестве открытых исходных кодов. Инсталляция и использование этой версии отличается; это руководство относится только к Unix-версии mpich.
- Поддержка широкого ряда сред, включая кластеры SMP и компьютеры массового параллелизма.
- Следование большинству (но еще не всем) целям построения и инсталляции, рекомендуемым GNU, включая VPATH.
- Поддерживаются части MPI-2:
  - Большая часть MPI-IO поддерживается через реализацию ROMIO (См. 'romio/README' о деталях).
  - Поддержка MPI_INIT_THREAD (но лишь MPI_THREAD_SINGLE).
  - Различные новые процедуры MPI_Info и MPI_Datatype.
- mpich также включает компоненты параллельной среды программирования, в том числе
  - Инструменты трассировки и регистрации, основанные на профилирующем интерфейсе MPI, включая расширяемый формат log-файла (SLOG).
  - Параллельные инструменты визуализации производительности (upshot и jumpshot).
  - Обширные тесты корректности и производительности.

Этот документ предполагает, что mpich уже создан и установлен в соответствии с инструкциями из *руководства по инсталляции* mpich.

2 Компоновка и запуск программ

mpich предлагает инструменты, упрощающие создание исполняемых файлов MPI. Поскольку программы mpich могут потребовать специальных библиотек и опций компиляции, Вам необходимо использовать команды, предлагаемые mpich для компиляции и компоновки программ. Однако, для специальных случаев, раздел 2.2 указывает, как определить необходимые библиотеки и опции, не используя скрипты mpich.

2.1 Скрипты для компиляции и компоновки приложений

Реализация `mpich` предлагает четыре команды для компиляции и компоновки программ на C (`mpicc`), Фортране 77 (`mpif77`), C++ (`mpiCC`) и Фортране 90 (`mpif90`).

Поддерживаются также следующие специальные опции:

-mpilog Создавать версию, генерирующую log-файлы MPE.

-mpitrace Создавать версию с трассировкой.

-mpianim Создавать версию, генерирующую анимацию в реальном времени.

-show Показать команды, которые можно использовать без их действительного запуска.

Используйте эти команды как обычные компиляторы C, Фортрана 77, C++ или Фортрана 90. Например,

```
mpicc -c foo.c
mpif77 -c foo.f
mpiCC -c foo.C
mpif90 -c foo.f
```

и

```
mpicc -o foo foo.o
mpif77 -o foo foo.o
mpiCC -o foo foo.o
mpif90 -o foo foo.o
```

Команды для компоновщика могут включать дополнительные библиотеки. Например, для использования процедур из библиотеки `math` для C, используйте

```
mpicc -o foo foo.o -lm
```

Комбинирование компиляции и компоновки в единой команде, как показано здесь

```
mpicc -o foo foo.c
mpif77 -o foo foo.f
mpiCC -o foo foo.C
mpif90 -o foo foo.f
```

также можно использовать.

Заметьте, что хотя суффиксы `.c` для программ C и `.f` для программ Фортрана являются стандартными, такого же соглашения о суффиксах для программ на C++ и Фортране 90 нет. Примеры, показанные выше подходят ко многим, но не ко всем системам. `mpich` пытается определить приемлемые суффиксы, но не всегда в состоянии сделать это.

Вы можете переопределить выбор компилятора определением переменных окружения `MPICH_CC`, `MPICH_F77`, `MPICH_CCC`, `MPICH_F90`. Однако, помните, что это будет работать, только если альтернативный компилятор совместим с компилятором по умолчанию (под совместимостью мы подразумеваем, что они используют те же самые размеры для типов данных и генерируют объектный код, который может использоваться библиотеками `mpich`). Если Вы хотите переопределить компоновщик, испорльзуйте переменные окружения `MPICH_CLINKER`, `MPICH_F77LINKER`, `MPICH_CCLINKER`, `MPICH_F90LINKER`.

2.1.1 Использование разделяемых бибилиотек

Разделяемые бибилиотеки могут помочь уменьшить размер исполняемых файлов. Это особенно ценно для кластеров рабочих станций, где исполняемые файлы обычно копируются по сети на каждую машину, которая исполняет параллельную программу. Однако, существует несколько практических проблем использования разделяемых библиотек; этот раздел обсуждает некоторые из них и методы решения большинства этих проблем. В настоящее время разделяемые библиотеки не поддерживаются для C++.

Чтобы создать разделяемые библиотеки для `mpich`, Вы должны конфигурировать и собирать `mpich` с опцией `--enable-sharedlib`. Поскольку каждая Unix-система и практически каждый компилятор использует различные и зачастую несовместимые множества опций для создания разделяемых объектов и библиотек, `mpich` может не определить коректные опции. В настоящее время `mpich` воспринимает Solaris, GNU `gcc` (на большинстве платформ, включая LINUX и Solaris) и IRIX. Информацию о построении разделяемых библиотек на друих платформах можно присылать на `mpi-bugs@mcs.anl.gov`.

Когда разделяемые библиотеки построены, Вы должны сообщить командам компиляции и компоновки `mpich` об их использовании (причина, по которой разделяемые библиотеки не используются по умолчанию, будет пояснена ниже). Вы можете сделать это либо опцией командной строки `-shlib` или установкой переменной окружения `MPICH_USE_SHLIB` в значение `yes`. Например,

```
mpicc -o cpi -shlib cpi.c
```

или

```
setenv MPICH_USE_SHLIB yes
mpicc -o cpi cpi.c
```

Использование переменной окружения `MPICH_USE_SHLIB` позволяет Вам управлять использованием разделяемых библиотек без изменения команд компиляции; это очень полезно в проектах, которые используют make-файлы.

Запуск программы, построенной с испольованием разделяемых библиотек, может быть очень сложным. Некоторые (большинство?) систем *не запоминают*, где находились разделяемые библиотеки, когда компоновался исполняемый файл! Вместо этого, они пробуют найти разделяемые библиотеки либо в каталоге по умолчанию (таком, как `'/lib'`), либо в каталоге, определяемом переменной окружения, такой, как `LD_LIBRARY_PATH`, либо в каталоге, определяемом агрументом командной строки, таким, как `-R` или `-rpath` (см. ниже более подробно). `configure` для `mpich` проверяет их и сообщает, когда исполняемый файл, построенный с разделяемыми библиотеками, вспоминает их размещение. Он также пытается использовать аргументы командной строки компилятора, чтобы заставить исполняемый файл вспомнить размещение разделяемых библиотек.

Если Вам необходимо установить переменные окружения, чтобы указать, где находятся разделяемые библиотеки `mpich`, Вам необходимо убедиться в том, что и процесс, из которого Вы запустили `mpirun`, и любые процессы, запущенные `mpirun`, получат переменную окружения. Простейший способ сделать это - установить переменную окружения внутри Вашего файла `'.cshrc'` (для пользователей `csh` или `tcsh`) или `'.profile'` (для пользователей `sh` или `ksh`).

Однако, установка переменной окружения внутри скриптов запуска может вызвать проблемы, если Вы используете несколько различных систем. Например, у Вас имеется единый файл `'.cshrc'`, который Вы используете в SGI (IRIX) и Solaris. Вы не хотите установить `LD_LIBRARY_PATH`, чтобы указать SGI на Solaris-версию разделяемых библиотек `mpich`[1]. Вместо этого, Вы можете пожелать установить переменную

[1] Вы можете заставить '.cshrc' проверять, в какой системе Вы работаете и устанавливать пути нужным образом. Это не такой гибкий подход, как способ с установкой переменных окружения из запущенного shell

окружения перед запуском `mpirun`:

```
setenv LD_LIBRARY_PATH ${LD_LIBRARY_PATH}:/usr/local/mpich/lib/shared
mpirun -np 4 cpi
```

К сожалению, это не всегда сработает. В зависимости от метода, используемого `mpirun` и `mpich` для запуска процессов, переменная окружения может не передаться новому процессу. Это вызовет завершение программы с сообщением вида

```
ld.so.1: /home/me/cpi: fatal: libpmpich.so.1.0: open failed:
No such file or directory
Killed
```

Существуют различные решения этой проблемы; каждое из них зависит от конкретного устройства `mpich` (например, `ch_p4`), которое Вы используете, и эти решения обсуждаются в соответствующих разделах в разд.3.

Альтернативой использованию `LD_LIBRARY_PATH` и безопасного сервера является добавление опции к команде компоновки, предоставляющей путь для использования в поиске разделяемых библиотек. К сожалению, опция, которая Вам нужна, это "добавить этот каталог в путь поиска" (такая, какую Вы получаете, используя `-L`). Вместо этого, многие компиляторы предлагают только опцию "заменить путь поиска на данный путь"[2]. Например, некоторые компиляторы позволяют использовать `-Rpath:path:...:path` для определения заменяемого пути. Тогда, если и `mpich`, и пользователь предоставляют путь поиска библиотек через `-R`, один из путей будет утерян. В итоге, `mpicc` и другие скрипты могут проверить наличие опции `-R` и создать универсальную версию, но сейчас они этого не делают. Вы можете, однако, предложить полный путь поиска самостоятельно, если Ваш компилятор поддерживает такие опции, как `-R`.

Все предшествующее может звучать как масса лишней работы и в некоторых случаях это так. Однако, для больших кластеров эти затраты окупаются: программы будут запускаться быстрее и более надежно, поскольку уменьшается трафик по сети и файловой системе.

2.1.2 Фортран 90 и модуль MPI

При конфигурировании `mpich` в процессе инсталляции определяется местонахождение компилятора Фортрана 90, и если он найден, создается две различные версии модуля MPI. Один из модулей включает только те процедуры MPI, которые не принимают аргументы "по выбору"; друой включает все процедуры MPI. Аргумент "по выбору" может принимать значения любого типа данных; обычно это буферы в процедурах коммуникации MPI, таких, как `MPI_Send` и `MPI_Recv`. Два различных модуля доступны через опции `mpif90 -nochoice` и `-choice` соответственно. Версия модуля "по выбору" поддерживает ограниченное множество типов данных (числовые скаляры и числовые одно- и двумерные массивы). Это экспериментальная возможность; если у Вас возникли проблемы, присылайте почту на `mpi-bugs@mcs.anl.gov`. Ни один из этих модулей не предлагает полной "расширенной поддержки Фортрана", определенной в стандарте MPI-2.

2.2 Компиляция и компоновка без скриптов

В некоторых случаях использовать скрипты, поставляемые с `mpich` для компиляции и компоновки программ, невозможно. Например, другие средства могут иметь свои собственные скрипты компиляции. В

[2] Даже если компоновщик может предложить форму "добавить в путь поиска"

этом случае, Вы можете использовать `-compile_info` и `-link_info`, чтобы указать скриптам компиляции `mpich` флаги компиляции и библиотеки компоновки, необходимые для корректной работы процедур `mpich`. Например, при использовании устройства `ch_shmem` в системе Solaris, библиотека thread (`-lthread`) должна компоноваться вместе с приложением. Если этой библиотеки нет, приложение будет собрано, но необходимые процедуры будут заменены версиями-заглушками из C-библиотеки Solaris, вызывающими некорректное завершение приложения.

Например, чтобы определить флаги, используемые для компиляции и компоновки программ на C, Вы можете использовать эти команды, вывод которых показан для устройства `ch_p4` на рабочей станции Linux.

```
% mpicc -compile_info
cc -DUSE_STDARG -DHAVE_STDLIB_H=1 -DHAVE_STRING_H=1 -DHAVE_UNISTD_H=1
-DHAVE_STDARG_H=1 -DUSE_STDARG=1 -DMALLOC_RET_VOID=1
-I/usr/local/mpich/include -c
% mpicc -link_info
cc -DUSE_STDARG -DHAVE_STDLIB_H=1 -DHAVE_STRING_H=1 -DHAVE_UNISTD_H=1
-DHAVE_STDARG_H=1 -DUSE_STDARG=1 -DMALLOC_RET_VOID=1
-L/usr/local/mpich/lib -lmpich
```

2.3 Запуск через `mpirun`

Для запуска программы MPI используйте команду `mpirun`, находящуюся в каталоге `'/usr/local/mpich/bin'`. Практически для всех систем Вы можете использовать команду

```
mpirun −np 4 a.out
```

для запуска программы `'a.out'` на четырех процессорах. Команда `mpirun -help` выдаст Вам полный список опций, которые также можно найти в приложении B.

При выходе `mpirun` возвращает статус одного из процессов, обычно процесса с рангом 0 в `MPI_COMM_WORLD`.

3 Специальные возможности различных систем

MPI сделан относительно просто для написания переносимых параллельных программ. Единственной вещью, не стандартизованной в MPI, является среда, внутри которой выполняются параллельные программы. Существуют три базовых типа параллельных сред: параллельные компьютеры, кластеры рабочих станций и интегрированные распределенные среды, которые мы называем "вычислительными решетками", и которые включают в себя параллельные компьютеры и рабочие станции, а также могут охватывать несколько географических мест. Естественно, параллельный компьютер (обычный) предоставляет интегрированный, относительно простой способ выполнения параллельных программ. С другой стороны, кластеры рабочих станций и среды-решетки, обычно не имеют стандартного способа выполнения параллельных программ и требуют некоторой дополнительной настройки. Реализация `mpich` создана, чтобы скрыть эти различия внутри скрипта `mpirun`; однако, если Вам необходимы специальные возможности или опции, или если Вы столкнулись с проблемой при запуске Вашей программы, Вы должны понимать различия между этими системами. Далее мы описываем специальные возможности, которые применяются для кластеров рабочих станций, решеток (поддерживаемых через устройство `globus2`) и некоторых параллельных компьютеров.

Для связывания множества многопроцессорных систем выбор устройства globus2, описанного в разд. 3.6, может быть лучшим, чем устройства ch_p4.

3.1 Программы MPMD

Существует возможность запуска параллельной программы с различными исполняемыми файлами и использованием нескольких устройств, таких, как ch_p4, ch_mpl, и globus2. Этот стиль параллельного программирования часто называется MPMD (множество программ при множестве данных). Во многих случаях, программу MPMD легко преобразовать в единую программу, которая использует ранг процесса для вызова различных процедур; это облегчает старт параллельных программ и их отладку. Если преобразование программы MPMD в SPMD (одна программа при множестве данных, не путайте с SIMD - *один поток команд* при множестве данных) не возможно, то Вы можете запускать программы MPMD через mpich. Однако, Вы не можете использовать mpirun для запуска программ; вместо этого, Вам нужно следовать инструкциям для каждого устройства. Для устройства globus2 см. разд. 3.6. Для устройства ch_p4 см. разд. 3.2.6 и обсуждение файлов procgroup. Для устройства ch_mpl Вам нужно смотреть документацию POE для Вашей системы о деталях по запуску программ MPMD.

3.2 Кластеры рабочих станций и устройство ch_p4

Большинство процессоров массового параллелизма (MPP) предоставляют способ запуска программы на требуемом количестве процессоров; mpirun дает возможность использовать соответствующую команду. В противоположность этому, кластеры рабочих станций требуют, чтобы каждый процесс в параллельной задаче запускался индивидуально, даже если существуют программы, помогающие запуску этих процессов (см. ниже 3.2.3). Поскольку кластеры рабочих станций не всегда организованы как MPP, требуется дополнительная информация для их использования. mpich нужно инсталлировать со списком участвующих рабочих станций, находящимся в файле 'machines.<arch>' в каталоге '/usr/local/mpich/share'. Этот файл используется mpirun для выбора процессоров для выполнения. (Использование гетерогенных кластеров обсуждается в разд. 3.2.6.) Далее в этом разделе обсуждаются некоторые детали этого процесса и решение проблем.

3.2.1 Проверка Вашего списка машин

Используйте скрипт 'tstmachines' в каталоге '/usr/local/mpich/sbin', чтобы убедиться в возможности использования всех машин, указанных в списке. Этот скрипт выполняет rsh и краткий вывод каталогов; он проверяет, что у Вас есть доступ к узлу и что программа в текущем каталоге видна на удаленном узле. Если возникают проблемы, они будут указаны. Эти проблемы должны быть устранены до продолжения работы.

Единственным аргументом tstmachines является имя архитектуры; оно то же самое, что и расширение файла machines. Например,

```
/usr/local/mpich/sbin/tstmachines sun4
```

проверяет, что программа в текущем каталоге может исполняться на всех машинах sun4 в списке machines. Эта программа ничего не выводит, если все в порядке; если Вы хотите увидеть, что происходит, используйте аргумент -v (verbose - подробно):

```
/usr/local/mpich/sbin/tstmachines -v sun4
```

Вывод этой команды может выглядеть примерно так

```
Trying true on host1.uoffoo.edu ...
Trying true on host2.uoffoo.edu ...
Trying ls on host1.uoffoo.edu ...
Trying ls on host2.uoffoo.edu ...
Trying user program on host1.uoffoo.edu ...
Trying user program on host2.uoffoo.edu ...
```

Если `tstmachines` обнаружит проблему, он может предположить возможные причины и решения. Вкратце здесь проводятся три теста:

1. Могут ли процессы запускаться на удаленной машине? `tstmachines` пытается запустить команду `true` для shell на каждой машине из файла `'machines'`, используя команду удаленной shell. Отметьте, что устройства `ch_p4` не требуют команды удаленной shell и могут использовать альтернативные методы (см. разд. 3.2.3 и 3.2.2).

2. Доступен ли текущий рабочий каталог всем машинам? Это предполагает выполнение `ls` для файла, который создала `tstmachines` при запуске `ls` с использованием команды удаленной shell. Отметьте, что устройство `ch_p4` не требует, чтобы все процессоры имели доступ к одной и той же файловой системе (см. разд. 3.2.6), но этого требует команда `mpirun`.

3. Может ли программа пользователя выполняться на удаленной системе? Здесь проверяется, что разделяемые библиотеки и другие компоненты корректно установлены на всех машинах.

3.2.2 Использование безопасной shell

Руководство по инсталляции поясняет, как настроить вашу среду, чтобы устройство `ch_p4` в сети могло использовать безопасную shell `ssh` вместо `rsh`. Это полезно для сетей, в которых по причинам безопасности использование `rsh` не разрешено.

3.2.3 Использование безопасного сервера

Поскольку каждая рабочая станция в кластере (обычно) требует, чтобы новый пользователь регистрировался на ней, и поскольку этот процесс требует больших временных затрат, `mpich` предлагает программу, которую можно использовать для ускорения этого процесса. Это безопасный сервер `serv_p4` в каталоге `'/usr/local/mpich/bin'`. Скрипт `'chp4_servs'` в этом же каталоге можно использовать для запуска `serv_p4` на тех рабочих станциях, где Вы можете выполнить `rsh`. Вы можете также запустить сервер вручную и позволить ему работать в фоновом режиме; это подходит для машин, которые не имеют соединения по `rsh`, но на которых у Вас есть профиль.

Прежде чем Вы запустите этот сервер, убедитесь, что безопасный сервер установлен для общего использования; если это так, один и тот же сервер можно использовать для всех. В этом режиме для инсталляции сервера необходим доступ супервизора. Если сервер еще не инсталлирован, Вы можете установить его для собственного использования без специальных привилегий командой

```
chp4_servs -port=1234
```

Она запустит безопасный сервер на всех машинах, перечисленных в файле
`'/usr/local/mpich/share/machines.<arch>'`.

Номер порта, указанный опцией `-port=`, должен отличаться от любого уже используемого порта рабочих станций.

Чтобы разрешить использование безопасного сервера устройствами `ch_p4`, добавьте в Вашу среду следующие определения:

```
setenv MPI_USEP4SSPORT yes
setenv MPI_P4SSPORT 1234
```

Значением `MPI_P4SSPORT` должен быть порт, на котором Вы хотите запустить безопасный сервер. Когда эти переменные окружения установлены, `mpirun` пытается использовать безопасный сервер для запуска программ, использующих устройство `ch_p4`.(Аргумент командной строки `-p4ssport` для `mpirun` может использоваться вместо этих переменных окружения; `mpirun -help` даст Вам дополнительную информацию.)

3.2.4 Кластеры SMP

При использовании кластеров SMP-машин (с устройством `ch_p4`, сконфигурированным с опцией `-comm=shared`), Вы можете контролировать количество процессов на каждом узле, которые общаются через разделяемую память. Во-первых, Вы должны изменить файл `machines` (см. разд. 3.2), чтобы указать количество процессов, запускаемых на каждой машине. Обычно это число не больше количества процессоров; на SMP с большим количеством процессоров это число должно быть на единицу меньше, чтобы предоставить один процессор операционной системе. Формат очень прост: каждая строка файла `machines` определяет имя компьютера, возможно продолженное двоеточием (:) и количеством возможных процессов. Например, файл содержит строки

```
mercury
venus
earth
mars:2
jupiter:15
```

определяющие три однопроцессных машины (`mercury, venus, earth`), двухпроцессную машину `mars` и 15-процессную машину `jupiter`.

По умолчанию `mpirun` использует не более процессоров, чем указано в файле `machines` для каждого узла, до 16 процессов на каждой машине. При установке переменной окружения `MPI_MAX_CLUSTER_SIZE` с положительным целым значением, `mpirun` может запускать не более `MPI_MAX_CLUSTER_SIZE` процессов на машине, разделяя память для взаимодействия. Например, если `MPI_MAX_CLUSTER_SIZE` имеет значение 4, то команда `mpirun -np 9` для указанного выше файла `machines` создаст один процесс на машинах `mercury, venus, earth`, два процесса на `mars` (поскольку файл `machines` определяет, что `mars` может иметь два процесса, разделяющих память) и 4 на `jupiter` (поскольку `jupiter` может иметь 15 процессов, но необходимы только 4). Если необходимо 10 процессов, `mpirun` стартует заново с начала файла`machines`, создавая дополнительный процесс на `mercury`; значение `MPI_MAX_CLUSTER_SIZE` предотвращает запуск `mpirun` пятого процесса с разделяемой памятью на `jupiter`.

3.2.5 Гетерогенные сети и устройство `ch_p4`

Гетерогенная сеть рабочих станций - это сеть, в которую входят машины с различными архитектурами и/или операционными системами. Например, сеть может содержать три рабочих станции Sun SPARC

(sun4) и три рабочих станции SGI IRIX, взаимодействующих через протокол TCP/IP. mpirun может использовать все станции путем указания множества аргументов -arch и -np. Например, для запуска программы на 3-х sun4 и 2 SGI IRIX, используйте

```
mpirun -arch sun4 -np 3 -arch IRIX -np 2 program.%a
```

Специальное имя программы program.%a позволяет Вам определить различные исполняемые файлы программы, поскольку исполняемые файлы Sun не могут работать на рабочих станциях SGI и наоборот. %a заменяется на имя архитектуры; в данном примере program.sun4 работает на машинах Sun и program.IRIX работает на машинах SGI IRIX. Вы также можете поместить программы в различные каталоги; например,

```
mpirun -arch sun4 -np 3 -arch IRIX -np 3 /tmp/%a/program
```

Важно определить архитектуру через -arch *перед* определением количества процессоров. Первая команда arch должна относиться к процессору, на котором будет запущена задача. Если не указана опция -nolocal, то первая -arch должна относиться к процессору, с которого запущен mpirun.

3.2.6 Файл procgroup для P4

Для большего контроля над тем, как запускается задача, нам нужно увидеть, как mpirun запускает параллельную программу на кластере рабочих станций. Каждый раз при запуске mpirun он создает и использует новый файл имен машин только для этого запуска, используя в качестве ввода файл machines. (Новый файл называется PIyyyy, где yyyy - идентификатор процесса). Если Вы определяете -keep_pg для вызова mpirun, Вы можете использовать эту информацию, чтобы видеть, где mpirun запустил несколько Ваших последних задач. Вы можете создать этот файл самостоятельно и определить его в качестве аргумента mpirun. Чтобы сделать это для ch_p4, используйте

```
mpirun -p4pg pgfile myprog
```

где pgfile - имя файла. Формат файла определен ниже.

Это необходимо, если Вы хотите жестко контролировать машины, на которых Вы работаете, или mpirun не может создать файл автоматически. Это происходит, если

- Вы хотите иметь различные исполняемые файлы на различных машинах (Ваша программа не SPMD).
- Вы хотите работать в сети многопроцессорных машин с разделяемой памятью и определить количество процессов, которые разделяют память на каждой машине.

Формат файла procgroup для ch_p4 - это множество строк вида

```
<hostname>  <#procs>  <progname>  [<login>]
```

Примером такого файла, где команда будет запущена с машины sun1, может быть

```
sun1    0  /users/jones/myprog
sun2    1  /users/jones/myprog
sun3    1  /users/jones/myprog
hp1     1  /home/mbj/myprog  mbj
```

Указанный выше файл определяет четыре процесса, по одному на каждой из трех sun и один на другой станции, где имя профиля пользователя отличается. Отметьте 0 в первой строке. Он здесь, чтобы указать, что на машине `sun1` не должен запускаться никакой другой процесс, кроме того, который запускается командой пользователя. Вы можете пожелать запустить все процессы на Вашей собственной машине в качестве теста. Вы можете сделать это, повторив ее имя в файле

```
sun1    0  /users/jones/myprog
sun1    1  /users/jones/myprog
sun1    1  /users/jones/myprog
```

Эта команда запустит три процесса на sun1, взаимодействующих через сокеты. Для запуска на многопроцессорной системе с разделяемой памятью 10 процессов, вы можете использовать файл вида

```
sgimp    9   /u/me/prog
```

Заметьте, что это пример для 10 процессов, один из которых непосредственно запускается пользователем, а другие девять определены в этом файле. Он требует, чтобы `mpich` был сконфигурирован с опцией `-comm=shared`; для дополнительной информации см. *руководство по инсталляции.*

Если Вы находитесь на машине `gyrfalcon` и хотите запустить задачу с одним процессом на `gyrfalcon` и тремя процессами на `alaska`, причем процессы на `alaska` взаимодействуют через разделяемую память, Вы можете использовать

```
local    0  /home/jbg/main
alaska   3  /afs/u/graphics
```

Не существует возможности передать различным процессам MPI различные аргументы командной строки.

3.2.7 Настройка производительности P4

Существует несколько переменных окружения и опций командной строки, которые можно использовать для настройки производительности устройства `ch_p4`. Отметьте, что эти переменные окружения должны быть определены для всех процессов, которые создаются, а не только для того, который Вы запустите из программы MPI (т.е., установка этих переменных должна быть частью Ваших файлов `'.login'` или `'.cshrc'`). Переменными окружения являются:

`P4SOCBUFSIZE` определяет размер буфера сокета в байтах. Увеличение этого значения может увеличить производительность на некоторых системах.

`P4_WINSHIFT` Это еще один параметр сокета, поддерживаемый только на некоторых платформах. Мы советуем не использовать его.

`P4_GLOBMEMSIZE` Это количество памяти в байтах, зарезервированной для коммуникации через разделяемую память (когда `mpich` конфигурируется с `-comm=shared`). Увеличьте его, если Вы получаете сообщение об ошибке, что `p4_shmalloc` возвращает NULL.

Настройка TCP. Опция командной строки `-p4sctrl` принимает параметры, которые определяют различные опции сокетов. Они предоставляются в форме имя=значение, разделенные двоеточием. За исключением `bufsize`, пользователи обычно не изменяют их значения по умолчанию. Имена и их значения:

bufsize Размер буфера сокета, в килобайтах. Например, `bufsize=32` требует буферы для сокетов по 32K. По умолчанию значение - 16.

winsize Размер `winshift`. доступно только на системах, которые определяют `TCP_WINSHIFT` и игнорируется в других случаях.

netsendw Использовать `select` для ожидания завершения `write`. Значения `y` (по умолчанию) и `n`.

netreadw Использовать `select` для ожидания завершения `read`. Значения `y` (по умолчанию) и `n`.

writev Использовать `writev` для отправки заголовка (конверта MPI) и данных в одном сообщении. Значения `y` (по умолчанию) и `n`.

readb Переключить сокет в блокирующий режим ожидания по чтению, вместо ожидания по занятости или использования `select`. Значения `y` (по умолчанию) и `n`.

stat Вывести статистику операций `write` и `read`. Использовать только опытным пользователям!

Например, для использования буферов сокетов по 64K и отключения использования `writev`, Вы должны использовать

```
mpirun -np 2 mpptest -p4sctrl bufsize=64:writev=n
```

3.2.8 Использование специальных межсоединений

В некоторых инсталляциях определенные машины могут соединяться несколькими способами. Например, "обычный" Ethernet может поддерживаться высокоскоростным кольцом FDDI. Обычно для идентификации высокоскоростных соединений используются альтернативные имена машин. Все, что Вам необходимо сделать - это поместить эти альтернативные имена в файл `machines.xxxx`. В этом случае важно использовать не форму `local 0`, а имя локальной машины. Например, если машины `host1` и `host2` соединены ATM с именами `host1-atm` и `host2-atm` соответственно, верный файл `procgroup` для `ch_p4` (для запуска программы '`/home/me/a.out`') будет иметь вид

```
host1-atm   0   /home/me/a.out
host2-atm   1   /home/me/a.out
```

3.2.9 Использование разделяемых библиотек с устройством `ch_p4`

Как указано в конце раздела 2.1.1, иногда необходимо быть уверенным, что переменные окружения передаются удаленным машинам, прежде чем запустится программа, использующая разделяемые библиотеки. Различные команды удаленной shell (например, `rsh` или `ssh`) не делают этого. К счастью, безопасный сервер (разд. 3.2.3) может передать переменные окружения. Сервер создается и инсталлируется как часть устройства `ch_p4`, и может быть установлен на всех машинах из файла `machines` для текущей архитектуры (предполагая, что там существует работающая команда удаленной shell) с помощью

```
chp4_servs -port=1234
```

Безопасный сервер распространяет все переменные окружения на удаленные процессы, и обеспечивает, чтобы среда процесса (содержащего Вашу программу MPI) содержала все переменные окружения, начинающиеся с `LD_` (в таком случае система использует `LD_SEARCH_PATH` или некоторое другое имя для поиска разделяемых бибилиотек).

3.3 Быстрый запуск через многоцелевого демона и устройство ch_p4mpd

Это устройство является экспериментальным и версия mpirun для него немного отличается от версий для других устройств. В этом разделе мы описываем, как работает система демонов mpd и как с их использованием запустить программу MPI. Чтобы использовать эту систему, mpich должен конфигурироваться для устройства ch_p4mpd, и демоны должны быть запущены на машине, с которой Вы хотите выполнить задачу. Этот раздел описывает, как это сделать.

3.3.1 Цели

Целью многоцелевого демона (mpd и связанного с ним устройства ch_p4mpd) является обеспечить поведение mpirun как единой программы, даже если она создает множество процессов для выполнения задачи MPI. Далее мы будем ссылаться на процесс mpirun и процессы MPI. Это поведение включает в себя

- быстрый масштабируемый запуск процессов MPI (и даже не-MPI). Для тех, кто привык использовать устройство ch_p4 в сетях TCP, это станет сразу же заметным изменением. Запуск задачи станет намного быстрее.
- сбор вывода и ошибок от процессов MPI в файлы стандартного вывода ошибок процесса mpirun.
- доставка стандартного ввода mpirun на стандартный ввод процесса MPI с номером 0.
- доставка сигналов от процесса mpirun ко всем процессам MPI. Это означает легкое прекращение, приостановку и возобновление Вашей параллельной задачи, как будто она является отдельным процессом, с помощью команд cntl-C, cntl-Z, bg, fg.
- доставка аргументов командной строки ко всем процессам MPI.
- копирование переменной окружения PATH из среды, в которой работает mpirun, в среды, в которых работают процессы MPI.
- использование необязательных аргументов для других переменных окружения.
- использование необязательных аргументов для указания, где должны запускаться процессы MPI (см .ниже).

3.3.2 Введение

Устройство ch_p4 по умолчанию обращается к rsh для обработки запуска на удаленной машине. Необходимость идентификации во время запуска задачи, сочетающаяся с последовательным процессом, в котором информация собирается с каждой удаленной машины и отсылается широковещательно всем остальным, делает запуск задачи неприемлимо медленным, особенно для большого числа процессов.

Начиная с mpich версии 1.2.0 мы ввели новый медот обработки запуска, основанный на демонах. Этот механизм, требующий конфигурирования нового устройства, еще не достаточно широко протестирован, чтобы использоваться в кластерах по умолчанию, но мы надеемся, что в конечном итоге он станет таким. В этой версии mpich он был существенно улучшен и сейчас инсталлируется вместе с mpich по команде make install. В системах с gdb он поддерживает простой параллельный отладчик, который мы называем mpigdb.

Основной идеей является создание перед моментом запуска задачи сети демонов на машинах, которые будут исполнять процессоы MPI, а также на машине, где будет выполняться mpirun. После этого

команды запуска задачи (и другие команды) будут обращаться к локальному демону и использовать уже существующие демоны для запуска процессов. Большая часть начальной синхронизации, выполняемой устройством `ch_p4`, устраняется, поскольку демоны могут использоваться во время выполнения для поддержки установки соединений между процессами.

Для использования нового механизма запуска Вы должны

- выполнить конфигурирование с новым устройством:

  ```
  configure --with-device=ch_p4mpd
  ```

- соберите `mpich` как обычно:

  ```
  make
  ```

- перейдите в каталог `mpich/mpid/mpd`, где находится код демонов и создаются демоны, или поместите этот каталог в Ваш путь поиска `PATH`.

- запустите демонов:

  Демоны можно запустить вручную на удаленных машинах, используя номера портов, выделяемых демонам при их запуске:

  - На fire:

    ```
    fire% mpd &
    [2]23792
    [fire_55681]: MPD starting
    fire%
    ```

  - На soot:

    ```
    soot% mpd -h fire -p 55681 &
    [1]6629
    [soot_35836]: MPD starting
    soot%
    ```

  Демоны `mpd` идентифицируются по имени машины и номеру порта.

  Если демоны не могут назначить порты автоматически, можно найти машину и номер порта, используя команду `mpdtrace`:

  - На fire:

    ```
    fire% mpd &
    fire% mpdtrace mpdtrace: fire_55681:lhs=fire_55681 rhs=fire_55681 rhs2=fire_55681
    fire%
    ```

  - На soot:

    ```
    soot% mpd -h fire -p 55681 &
    soot% mpdtrace
    mpdtrace: fire_55681: lhs=soot_33239 rhs=soot_33239 rhs2=fire_55681
    mpdtrace: soot_33239: lhs=fire_55681 rhs=fire_55681 rhs2=soot_33239
    soot%
    ```

`mpdtrace` показывает кольцо из `mpd`, определяемых именем машины и номером порта, используемыми для включения следующего `mpd` в кольцо. Левый и правый соседи каждого `mpd` обозначаются соответственно как `lhs` и `rhs`. `rhs2` показывает демон, находящийся в двух позициях справа (в этом примере он указывает сам на себя).

Вы можете использовать команду `mpd -b` для запуска демонов как настоящих демонов, отсоединенных от терминала. Этот метод имеет свои преимущества и недостатки.

Существует пара скриптов в каталоге `mpich/mpid/mpd`, которые могут помочь:

```
localmpds <number>
```

запускает `<number>` экземпляров `mpd` на локальной машине. Это очень полезно для тестирования. Обычно Вы выполняете

```
mpd &
```

для запуска одного mpd на локальной машине. Другие `mpd` на удаленных машинах могут быть запущены через `rsh`, если возможно:

```
remotempds <hostfile>
```

где `<hostfile>` содержит имена других машин для запуска `mpd`. Это простой файл имен машин, не похожий по формату на файлы `MACHINES`, используемые устройством `ch_p4`, которые могут содержать комментарии и другие обозначения.

См. также скрипт `startdaemons`, который инсталлируется вместе с `mpich`.

- Наконец, запустите задачи командой `mpirun` как обычно

  ```
  mpirun -np 4 a.out
  ```

  Вы можете остановить демонов командой `mpichstop`.

3.3.3 Примеры

Здесь приведены несколько примеров использования `mpirun`, которые созданы при условии, что `mpich` конфигурировался и строился для устройства `ch_p4mpd`.

- Запустите пример `cpi`

  ```
  mpirun -np 16 cpi
  ```

- Вы можете получить пометки строк в `stdout` и `stderr` от Вашей программы, добавив опцию `-l`. Выводимые строки будут помечены по рангам процессоров.

- Запустите программу `fpi`, которая запрашивает количество используемых интервалов.

  ```
  mpirun -np 32 fpi
  ```

  Потоки `stdin, stdout` и `stderr` будут отображаться обратно в Ваш процесс `mpirun`, даже если процесс MPI ранга 0 будет выполняться на удаленной машине.

- Используйте аргументы и переменные окружения.

```
mpirun -np 32 myprog arg1 arg2 -MPDENV- MPE_LOG_FORMAT=SLOG \
GLOBMEMSIZE=16000000
```

Аргумент -MPDENV- - это *ограждение*. Все аргументы после него обрабатываются `mpirun`, а не приложением.

- Определите, где будет запущен первый процесс. По умолчанию, процессы MPI порождаются последовательно демонами `mpd` в кольце, начиная с демона *после* локального (демона, запущенного на той же машине, что и `mpirun`). Таким образом, если Вы входите на машину `dion` и существуют `mpd`, работающие на `dion` и машинах `belmont1, belmont2, ..., belmont64`, то после запуска

  ```
  mpirun -np 32 cpi
  ```

  Ваши процессы будут запущены на `belmont1, belmont2, ..., belmont32`. Вы можете заставить Ваши процессы MPI работать где-либо еще, задав `mpirun` аргументы оптимального размещения. Если Вы вводите

  ```
  mpirun -np 32 cpi -MPDLOC- belmont33 belmont34 ... belmont64
  ```

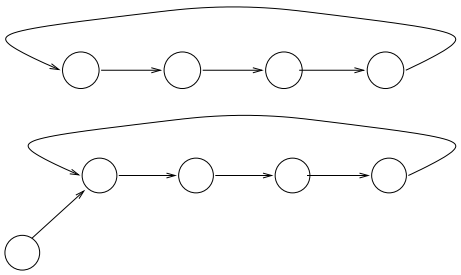
  то Ваша задача будет работать на `belmont33 belmont34 ... belmont64`. В общем, процессы будут запущены только на машинах из списка машин после `-MPDLOC-`.

  Это предварительный и чрезвычайно грубый способ выбрать размещение MPI-процессов для `mpirun`. В дальнейшем мы предполагаем использовать проект `mpd` как среду для исследования интерфейсов между расписаниями задач, менеджерами процессов, параллельными приложениями (особенно в динамических средах MPI-2) и командами пользователя.

- Определите, на каких машинах запущены Ваши mpd:

  ```
  mpirun -np 32 hostname | sort | uniq
  ```

  Эта команда запустит 32 экземпляра `hostname`, предполагая наличие `/bin` в Вашем пути поиска, независимо от того, сколько здесь имеется `mpd`. Другие процессы будут запускаться по кругу в кольце `mpd`.



3.3.4 Как работают демоны[3]

Как только демоны запускаются, они соединяются в кольцо: "консольный" процесс (`mpirun, mpdtrace, mpdallexit`, и т.д.) может соединиться с любым `mpd` через именованный сокет Unix, установленный в `/tmp` локальным `mpd`. Если это процесс `mpirun`, он требует, чтобы было запущено определенное число процессов, начиная с машины, заданной `-MPDLOC-`, как было указано выше. По умолчанию место размещения - следующий `mpd` в кольце после того, который был запрошен с консоли. После этого происходят следующие события.

- `mpd` порождает требуемое количество процессов `manager` (исполняемый файл называется `mpdman` и расположен в каталоге `mpich/mpid/mpd`). Менеджеры порождаются последовательно всеми `mpd` в кольце, делая полный круг, если необходимо.
- Менеджеры самостоятельно объединяются в кольцо, и порождают процессы приложения, называемые клиентами.
- Консоль отсоединяется от `mpd` и присоединяется к первому менеджеру. `stdin` от `mpirun` доставляется к клиенту менеджера 0.
- Менеджеры перехватывают стандартный ввод-вывод от клиентов, и доставляют им аргументы командной строки и переменные окружения, заданные в команде `mpirun`. Сокеты, содержащие `stdout` и `stderr`, формируют дерево с менеджером 0 в качестве корня.

С этого момента ситуация выглядит подобно показанной на рис. 1. Когда клиенту необходимо соединиться с другим клиентом, они используют менеджеры, чтобы найти подходящий процесс на машине-приемнике. Процесс `mpirun` может быть приостановлен - в этом случае останавливаются и его клиенты, однако `mpd` и менеджеры продолжают выполняться, чтобы они смогли разбудить клиентов после пробуждения `mpirun`. Уничтожение процесса `mpirun` уничтожает и клиентов, и менеджеров.

Одно и то же кольцо `mpd` может использоваться для запуска множества задач с множества консолей в одно и то же время. При обычных условиях необходимо, чтобы для каждого пользователя существовало отдельное кольцо `mpd`. Для целей безопасности каждый пользователь должен иметь в своем домашнем каталоге доступный для чтения только ему файл `'.mpdpasswd'`, содержащий пароль. Файл будет считываться при запуске `mpd`. Только `mpd`, знающие этот пароль, могут войти в кольцо существующих `mpd`.

Новой является возможность конфигурировать систему `mpd`, чтобы демоны могли загружаться как `root`. Чтобы сделать это после конфигурирования `mpich` Вам нужно повторно переконфигурировать ее в каталоге `mpid/mpd` с опцией `--enable-root` и пересобрать. Тогда `mpirun` будет инсталлирован как программа `setuid`. Несколько пользователей могут использовать одно и то же множество `mpd`, которые были запущены `root`, несмотря на то, что их `mpirun`, менеджеры и клиенты будут запущены от имени пользователя, вызвавшего `mpirun`.

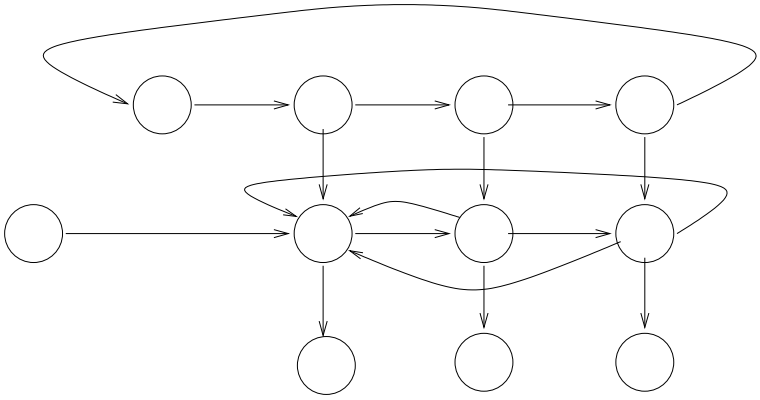


Рис. 1: Система `mpd` с консолью, менеджерами и клиентами

3.3.5 Запуск задач `mpich` под управлением MPD

Поскольку демоны MPD уже находятся в соединении друг с другом перед запуском задачи, то ее запуск происходит гораздо быстрее, чем при использовании устройства `ch_p4`. Команда `mpirun` для устройства `ch_p4mpd` имеет ряд специальных аргументов командной строки. Если Вы введете `mpirun` без аргументов, они будут выведены:

```
% mpirun
Использование: mpirun <аргументы> программа <аргументы программы>
Аргументы:
-np   число_процессов_для_запуска (обязательные 2 первых аргумента)
[-s]  (закрыть stdin; может запускаться в фоновом режиме без ввода с терминала)
[-g размер_группы]  (запускает размер_группы процессов на одном mpd)
[-m файл_machine]  (имя файла возможных машин)
[-l]  (метки строк; уникальные метки для вывода от каждого процесса)
[-1]  (не запускать первый процесс локально)
[-y]  (запускать как задачу для Myrinet)
```

Опция `-1` позволяет Вам, например, запускать `mpirun` на “входном узле” или “узле разработки” в Вашем кластере, но запускать все процессы приложения на “вычислительных узлах”.

Программа `mpirun` запускается в отдельном (не-MPI) процессе, который создает процессы MPI, выполняющие определенные исполняемые файлы. Он служит единым процессом, представляющим параллельные процессы MPI, чтобы сигналы, посылаемые ему, такие как $\hat{\mathrm{Z}}$ и $\hat{\mathrm{C}}$, передавались через систему демонов MPD ко всем процессам. Потоки вывода `stdout` и `stderr` от процессов MPI возвращаются в `stdout` и `stderr` для `mpirun`. Как и в большинстве реализаций MPI, `stdin` от `mpirun` направляется в `stdin` процесса MPI с рангом 0.

3.4 Отладка программ MPI

Отладка параллельных программ, как известно, сложный процесс. Параллельным программам присущи не только обычные виды ошибок, но и новые, связанные с временем и ошибками синхронизации. Часто

программы “зависают”, когда процесс ждет сообщения, которое никогда не было послано или послано с неверным идентификатором. Параллельные ошибки часто возникают, когда Вы добавляете код, чтобы попытаться определить ошибку, что особенно обидно. В этом разделе мы обсуждаем три подхода к параллельной отладке.

3.4.1 Подход printf

Как и в последовательной отладке, Вы часто желаете отследить интересующие события в программе выводом сообщений трассировки. Обычно Вы идентифицируете сообщение рангом процесса, пославшего его. Это делается непосредственным помещением ранга в сообщение трассировки. Как было указано выше, использование опции “метки строк” (`-l`) для `mpirun` в устройстве `ch_p4mpd` в `mpich` добавляет ранг автоматически.

3.4.2 Использование коммерческого отладчика

Отладчик TotalView© от Etnus, Ltd. [1] работает на множестве платформ и взаимодействует со многими фирменными реализациями MPI, включая `mpich` на кластерах Linux. Для устройства `ch_p4` Вы вызываете TotalView командой

```
mpirun -tv <другие аргументы>
```

а для устройства `ch_p4mpd` Вы используете

```
totalview mpirun <другие аргументы>
```

Здесь `mpirun` представляет параллельную задачу как целое. TotalView содержит специальные команды для вывода очередей сообщений в процессе MPI. Существует возможность присоединить TotalView к набору процессов, которые уже запущены параллельно; его также можно присоединить к одному из процессов.

3.4.3 Использование `mpigdb`

Устройство `ch_p4mpd` в `mpich` предоставляет “параллельный отладчик”, который состоит просто из нескольких копий отладчика `gdb`, и механизма перенаправления `stdin`. Команда `mpigdb` является версией `mpirun`, которая запускает каждый процесс под управлением `gdb` и управляет `stdin` для `gdb`. Команда ‘`z`’ позволяет Вам направит ввод с терминала в определенный процесс или разослать его всем процессам. Мы демонстрирем это запуском `cpi` под управлением простого отладчика:

```
donner% mpigdb -np 5 cpi                    # по умолчанию вывод от всех
(mpigdb) b 29                               # установить точку останова для
                                            # всех
0-4: Breakpoint 1 at 0x8049e93: file cpi.c, line 29.
(mpigdb) r                                  # запустить все 0-4:
Starting program:/home/lusk/mpich/examples/basic/cpi
0: Breakpoint 1, main (argc=1, argv=0xbffffa84) at cpi.c:29
1-4: Breakpoint 1, main (argc=1, argv=0xbffffa74) at cpi.c:29
0-4: 29   n = 0;                            # все достигли точки останова
(mpigdb) n                                  # пошаговый режим для всех
0: 38 if (n==0) n=100; else n=0;
```

```
1-4: 42 MPI_Bcast (&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
(mpigdb) z 0                                  # stdin только для процесса
                                              # ранга 0
(mpigdb) n                                    # пошаговый режим процесса
                                              # ранга 0
0: 40 startwtime = MPI_Wtime ();
(mpigdb) n                                    # до останова
0: 42 MPI_Bcast (&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
(mpigdb) z                                    # stdin возвращен для всех
                                              # процессов
(mpigdb) n                                    # пошаговый режим для всех
...                                           # до интересующего места
(mpigdb) n
0-4: 52     x = h * ( (double)i - 0.5);
(mpigdb) p x                                  # вывод от всех процессов
0: $1 = 0.0050000000000000001                 # значение x процесса 0
1: $1 = 0.014999999999999999                  # значение x процесса 1
2: $1 = 0.025000000000000001                  # значение x процесса 2
3: $1 = 0.035000000000000003                  # значение x процесса 3
4: $1 = 0.044999999999999998                  # значение x процесса 4
(mpigdb) c                                    # продолжить все
0: pi is approximately 3.141600986923, Error is 0.000008333333
0-4: Program exited normally.
(mpigdb) q                                    # выход
donner%
```

Если отлаживаемый процесс зависает (нет приглашения `mpigdb`) из-за ожидания текущим процессом действий другого процесса, ctl-C вызовет меню, позволяющее Вам переключать процессы. `mpigdb` не так развит, как TotalView, но зачастую полезнее и свободно распространяется с `mpich`.

3.5 Симметричные мультипроцессоры (SMP) и устройство ch_shmem

Во многих реализациях с разделяемой памятью `mpich` резервирует часть разделяемой памяти, через которую передаются сообщения. По умолчанию, `mpich` резервирует приблизительно 4 Мбайт разделяемой памяти. Вы можете изменить объем с помощью переменной окружения `MPI_GLOBMEMSIZE`. Например, чтобы предоставить 8 МБайт, введите

```
setenv MPI_GLOBMEMSIZE 8388608
```

Большие сообщения передаются частями, так что `MPI_GLOBMEMSIZE` не может ограничить максимальный размер сообщения, но его увеличение может улучшить производительность. Помните, что системы могут ограничивать количество доступной разделяемой памяти.

По умолчанию, `mpich` ограничивает количество процессов для устройства `ch_shmem` числом 32, несмотря на то, что во время конфигурирования было определено, что машина может иметь больше процессов. Вы можете переопределить этот предел, установив переменную окружения `PROCESSOR_COUNT` в максимальное число процессов, которое Вы хотите запустить, а затем переконфигурировать и пересобрать `mpich`.

3.6 Вычислительные решетки: устройство globus2

Устройство globus2[3] поддерживает выполнение программ MPI на "вычислительных решетках", которые могут включать параллельные компьютеры и рабочие станции, охватывающие несколько географических мест. В таких решетчатых средах различные машины могут поддерживать разные механизмы безопасности и механизмы создания процессов. Устройство globus2 скрывает от Вас низкоуровневые детали, позволяя Вам запускать программы через mpirun, как на многопроцессорных системах, так и на кластерах рабочих станций. Оно также предлагает другие удобные опции, такие, как удаленный доступ к файлам и вычислительную поддержку. Эти возможности предоставляются использованием служб, поддерживаемых пакетом Globus: см.http://www.globus.org о деталях.

Устройство globus2 требует, чтобы на компьютерах, где будут создаваться процессы, были запущены специальные серверы. В нашей дискуссии об использовании устройства globus2 мы предполагаем, что будем использовать globus2 на наборе машин, где уже установлены и запущены различные серверы Globus. Такой набор часто определяется как "вычислительная решетка", например NASA's Information Power Grid (IPG). Если это возможно, мы рекомендуем Вам использовать устройство globus2 для таких сред. Если Вы хотите использовать устройство globus2 в других ситуациях, пожалуйста, пришлите письмо на developers@globus.org. Детали о запуске программ MPI с использованием устройства globus2 в вычислительных решетках на основе Globus приведены в приложении C.

3.7 Многопроцессорные системы

Каждая многопроцессорная система отличается от других и даже системы одного производителя могут использовать различные способы запуска задач при различных инсталляциях. Программа mpirun пытается адаптироваться к этому, но Вы можете обнаружить, что она не работает с Вашей инсталляцией. Один из шагов, который Вы можете предпринять - использовать опцию -show или -t (тестирование) для mpirun. Она покажет, как mpirun пытается запустить Вашу программу без ее действительного запуска. Вы можете использовать эту информацию вместе с инструкциями по запуску программ на Вашей машине, чтобы узнать, как запустить программу. Пожалуйста, сообщите нам (mpi-bugs@mcs.anl.gov) о всех специальных требованиях.

IBM SP. Использование mpirun на компьютерах IBM SP может оказаться сложным, поскольку существует много различных (и часто взаимоисключающих) способов запуска программ на них. mpirun, распространяемый вместе с mpich, работает на системах, использующих планировщик Argonne (иногда называемый EASY) и в системах, использующих предустановленные величины для менеджера ресурсов (т.е. тех, которые не требуют от пользователя выбора RMPOOL). Если у Вас есть проблемы с запуском программ mpich, попробуйте следовать правилам инсталляции для запуска программы MPL или POE (при использовании устройства ch_mpl) или для запуска p4 (при использовании устройства ch_p4).

4 Примеры программ MPI

Пакет mpich содержит ряд примеров программ, находящихся в дереве исходных файлов. Большиство из них работают с любой реализацией MPI, а не только с mpich.

examples/basic содержит несколько коротких программ на Фортране, C и C++ для тестирования простейших возможностей MPI.

[3] globus2 заменило устройство globus, распространявшееся с предыдущими версиями mpich

examples/test содержит множество каталогов тестов для различных частей MPI. Введите "make testing" в этом каталоге для запуска нашего набора тестов функций.

examples/perftest - программы оценки производительности. См. скрипт `runmpptest` об информации по запуску тестов производительности. Они относительно сложны.

mpe/contrib/mandel - программа Mandelbrot, использующая графический пакет MPE, поставляемый вместе с `mpich`. Она должна работать с любой реализацией MPI, однако мы не тестировали ее. Это хорошая демонстрационная программа, если у Вас быстрый X сервер и не слишком много процессов.

mpe/contrib/mastermind - программа для параллельного решения головоломки Mastermind. Она может использовать графику (`gmm`) или не использовать ее (`mm`).

Дополнительные примеры из книги Using MPI [10] доступны через WWW на `ftp://info.mcs.anl.gov/pub/mpi/using`. На веб-сайте `ftp://info.mcs.anl.gov/pub/mpi/` можно также найти обучающий материал.

5 Библиотека полезных расширений MPE

Самая последняя версия этого материала может быть найдена в *руководстве пользователя MPE* и в файле `mpe/README`.

Ожидается, что `mpich` будет и далее накапливать процедуры расширения различных направлений. Мы собираем их в библиотеке, названной `mpe`, от MultiProcessor Environment (многопроцессорная среда).

В настоящее время главными компонентами `MPE` являются

- Набор процедур для создания log-файлов для анализа различными инструментами графической визуализации: `upshot, nupshot, Jumpshot-2` и `Jumpshot-3`.
- Параллельная графическая библиотека X с разделением дисплея.
- Процедуры для преобразования в последовательный код секции параллельно выполняемого кода.
- Процедуры настройки отладчика.

5.1 Создание log-файлов

MPE предлагает несколько способов генерации log-файлов, описывающих прогресс вычислений. Эти log-Файлы могут быть просмотрены одним из графических инструментов, распространяемым вместе с MPE. К тому же Вы можете настроить эти log-файлы, чтобы добавить информацию, специфическую для приложения.

Простейшим способом генерации log-файлов является связывание Вашей программы со специальной библиотекой MPE, которая использует возможности профилирования MPI перехватывать все вызовы MPI в приложении.

Вы можете создать настраиваемые log-файлы для просмотра с помощью вызовов различных процедур регистрации `MPE`. О деталях смотрите `man`-страницы MPE. Пример приведен в разд. 5.5.3.

Таблица 1: Графические процедуры MPE

Процедуры управления	
`MPE_Open_graphics`	(совместно) открывает дисплей X
`MPE_Close_graphics`	Закрывает графическое устройство X11
`MPE_Update`	Обновляет дисплей X11
Процедуры вывода	
`MPE_Draw_point`	рисует точку на дисплее X
`MPE_Draw_points`	рисует точки на дисплее X
`MPE_Draw_line`	Рисует линию на дисплее X11
`MPE_Draw_circle`	Рисует окружность
`MPE_Fill_rectangle`	Рисует закрашенный прямоугольник на дисплее X11
`MPE_Draw_logic`	Устанавливает логическую операцию для новых пикселей
`MPE_Line_thinkness`	Устанавливает толщину линий
`MPE_Make_color_array`	Создает массив индексов цветов
`MPE_Num_colors`	Возвращает число доступных цветов
`MPE_Add_RGB_color`	Добавляет новый цвет
Процедуры ввода	
`MPE_Get_mouse_press`	Возвращает текущие координаты мыши
`MPE_Get_drag_region`	Возвращает прямоугольную область

5.2 Формат log-файлов

В настоящее время MPE предлагает три различных формата log-файлов - ALOG, CLOG и SLOG. ALOG предоставляется для обратной совместимости и отмечает события текстом ASCII. CLOG похож на ALOG, но запоминает данные в двоичном формате (по существу “external32”) SLOG - это сокращение от Scalable LOGfile (расширяемый log-файл), запоминающий данные как состояния (а именно событие и его длительность) в специальном двоичном формате, выбранном для облегчения обработки программой визуализации очень больших (в несколько Гб) файлов.

Каждый из этих форматов log-файлов распознается одной или несколькими программами визуализации. Формат ALOG распознается программой nupshot. Формат CLOG распознается программой `nupshot` и `jumpshot`, базирующейся на Java. SLOG и `Jumpshot-3` способны обрабатывать log-файлы, содержащие гигабайты данных.

5.3 Параллельная графика для X

MPE предлагает ряд процедур, поволяющих Вам выводить простую графику через систему X Window. К тому же здесь есть процедуры для ввода, такие, как определение области с помощью мыши. Примеры доступных графических процедур показаны в табл. 1. Об аргументах смотрите страницы `man`. Вы можете найти пример использования графической библиотеки MPE в каталоге `mpich/mpe/contrib/mandel`. Введите

```
make
mpirun -np 4 pmandel
```

чтобы увидеть параллельный алгоритм вычисления Mandelbrot, который демонстрирует некоторые возможности графической библиотеки MPE.

5.4 Другие процедуры MPE

Иногда, во время выполнения параллельной программы Вам необходимо убедиться, что лишь несколько (зачастую один) процессор в данное время чем-то занят. Процедуры `MPE_Seq_begin` и `MPE_Seq_end` позволяют Вам создать "последовательную секцию" в параллельной программе.

Стандарт MPI упрощает для пользователей определение процедур, вызываемых при обнаружении ошибки MPI. Зачастую то, что Вы хотите получить, это чтобы программа вызывала отладчик, чтобы Вы могли диагностировать проблему немедленно. В некоторых средах сделать это позволяет Вам обработчик ошибок в `MPE_Errors_call_dbx_in_xterm`. В дополнение к этому Вы можете скомпилировать библиотеку MPE с вложенным отладочным кодом. (См. опцию `configure` с названием `-mpedbg`).

5.5 Профилирующие библиотеки

Профилирующий интерфейс MPI предлагает Вам удобный способ добавить инструменты анализа производительности к любой реализации MPI. Мы демонстрируем этот механизм в `mpich` и даем Вам возможность начать, поддерживая три профилирующих библиотеки в поставке `mpich`. Пользователи MPE могут создать и использовать эти библиотеки с любой реализацией MPI.

5.5.1 Сбор информации о затратах времени в процедурах MPI

Первая профилирующая библиотека очень простая. Профилирующая версия каждой из процедур `MPI_Xxx` вызывает `PMPI_Wtime` (создающую временную отметку) до и после каждого вызова соответствующей процедуры `PMPI_Xxx`. Времена накапливаются в каждом процессе и выводятся в отдельные файлы для каждого процесса в профилирующей версии `MPI_Finalize`. Файлы затем могут использоваться либо в общем отчете, либо в отчете процесс за процессом. Эта версия не принимает во внимание вложенные вызовы, возникающие при реализации, например, `MPI_Bcast` в рамках `MPI_Send` и `MPI_Recv`. Файл '`mpe/src/trc_wrappers.c`' реализует этот интерфейс, а опция `-mpitrace` в любом скрипте компиляции (например, `mpicc`) автоматически подключает эту библиотеку.

5.5.2 Автоматическая регистрация

Вторая профилирующая библиотека называется библиотекой *регистрации* MPE для генерации log-файлов, которые являются файлами временных отметок событий для CLOG или временных отметок состояний для SLOG. Во время выполнения делаются вызовы `MPE_Log_event` для запоминания определенных типов событий в памяти, после чего эти буферы памяти собираются и сливаются параллельно во время `MPI_Finalize`. Во время выполнения можно использовать `MPI_Pcontrol` для приостановления или перезапуска операций регистрации. (По умолчанию регистрация включена. Вызов `MPI_Pcontrol (0)` выключает регистрацию; `MPI_Pcontrol (1)` снова включает ее.) Вызовы `MPE_Log_event` делаются автоматически при каждом вызове MPI. Вы можете анализировать конечные log-файлы множеством инструментов; они описаны в разд. 5.6.1 и 5.6.2.

5.5.3 Настраиваемая регистрация

В дополнение к использованию предопределенных библиотек *регистрации* MPE для регистрации всех вызовов MPI, вызовы регистрации MPE могут встраиваться в пользовательские программы MPI для определения и регистрации состояний. Эти состояния называются состояниями, *определяемыми пользователем*. Состояния могут вкладываться друг в друга, позволяя определять состояния, описывающие процедуру

пользователя, содержащую несколько вызовов MPI, и показывать как состояние, определенное пользователем, так и содержащиеся в нем операции MPI. Процедура `MPE_Log_get_event_number` может использоваться для назначения уникальных номеров событий[4] из системы MPE. Процедуры `MPE_Describe_state` и `MPE_Log_event` могут затем использоваться для описания пользовательских состояний. Следующий пример иллюстрирует использование этих процедур.

```
int eventID_begin, eventID_end;
...
eventID_begin = MPE_Log_get_event_number ();
eventID_end = MPE_Log_get_event_number ();
...
MPE_Describe_state ( eventID_begin, eventID_end, "Amult", "bluegreen" );
...
MyAmult ( Matrix m, Vector v )
/* Записывает стартовое событие вместе с размером матрицы */
MPE_Log_event ( eventID_begin, m->n, (char *)0 );
... Код Amult, включая вызовы MPI ...
MPE_Log_event ( eventID_end, 0, (char *)0 );
```

Log-файл, генерируемый этим кодом, будет отмечать процедуры MPI внутри процедуры MyAmult зеленоголубым прямоугольником. Цвет, использованный в этом коде выбран из файла rgb.txt, предоставленного инсталляцией сервера X, т.е. '`rgb.txt`' находится в каталоге '`/usr/X11R6/lib/X11`' в Linux.

Если бибилотека регистрации MPE '`liblmpe.a`' не связаны с программой пользователя, необходимо использовать перед и после всех вызовов MPE процедуры `MPE_Init_log` и `MPE_Finish_log`. Программы-примеры '`cpilog.c`' и '`fpi.f`' для иллюстрации использования этих процедур MPE находятся в каталоге исходных кодов MPE '`contrib/test`' или в установленном каталоге '`share/examples`'.

5.5.4 Анимация в реальном времени

Третья библиотека реализует простую программную анимацию в реальном времени. Графическая библиотека MPE содержит процедуры, позволяющие множеству процессов разделять дисплей X, который не ассоциирован ни с одним определенным процессом. Наш прототип использует эти возможности для рисования стрелок, которые представляют трафик сообщений при выполнении программ.

5.6 Просмотрщики log-файлов

Существует четыре графических средства визуализации, распространяемых вместе с MPE - это `upshot`, `nupshot`, `Jumpshot-2` и `Jumpshot-3`. Из этих 4 просмотрщиков log-файлов только три построены с помощью MPE. Это `upshot`, `Jumpshot-2` и `Jumpshot-3`.

5.6.1 Upshot и Nupshot

Один из используемых нами инструментов называется `upshot`, написанный на Tcl/Tk, который происходит от Upshot [13]. Вид экрана `Upshot` в работе показан на рис. 2.

Он показывает параллельные линии времени с состояниями процессов, подобно paraGraph [12]. Вид может увеличиваться или уменьшаться по горизонтали или вертикали, центрироваться на любой точке

[4] Это важно, если Вы пишете библиотеку, которая использует процедуры регистрации MPE

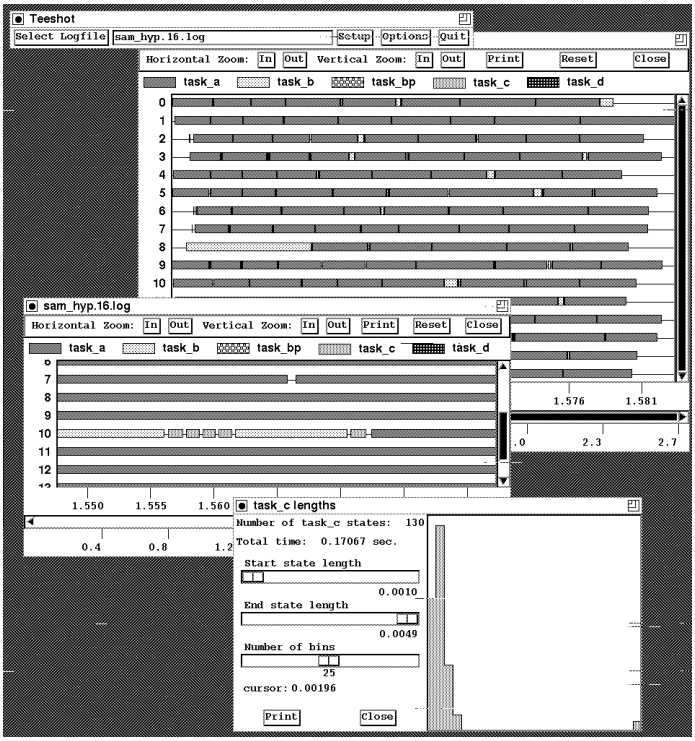


Рис. 2: Снимок экрана `upshot`

дисплея, выбранной мышью. На рис. 2 среднее окно происходит от увеличения верхнего окна в выбранной точке, чтобы увидеть больше деталей. Нижнее окно показывает гистограмму продолжительности состояний, с несколькими настраиваемыми параметрами.

`Nupshot` - это версия `upshot`, которая быстрее, но требует устаревшей версии Tcl/Tk. Из-за этих ограничения `Nupshot` не создается по умолчанию в текущей версии MPE.

5.6.2 Jumpshot-2 и Jumpshot-3

Существует две версии Jumpshot, поставляемых вместе с MPE. Это `Jumpshot-2` и `Jumpshot-3`, развившиеся из `Upshot` и `Nupshot`. Обе написаны на Java и являются графическими средствами визуализации для интерпретации двоичных файлов трасс, которые показывают их на дисплее, так, как показано на рис. 3.

О Jumpshot-2 см. [17] для дополнительной информации. Для Jumpshot-3 см. файл `'mpe/viewers/jumpshot-3/doc/TourStepByStep.pdf'` для краткого введения.

По мере увеличения размера log-файла уменьшается производительность `Jumpshot-2`, что может привести к зависанию `Jumpshot-2` во время чтения файла. Трудно определить, в какой момент происходит зависание, но мы наблюдали это при файлах менее 10Mb. Если размер файла CLOG около 4 Mb, производительность `Jumpshot-2` начинает заметно снижаться. Это текущие исследовательский результат, который получается из-за попытки сделать программу, основанную на Java, значительно более расширяемой. Результаты первой итерации этих попыток - это SLOG, поддерживающий расширяемую регистрацию

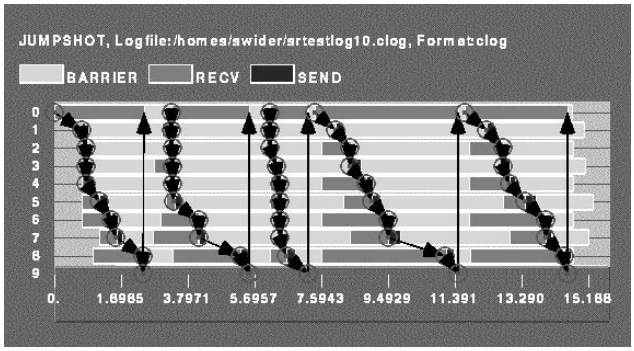


Рис. 3: Дисплей `Jumpshot-1`

данных и `Jumpshot-3`, который читает SLOG.

5.7 Автоматическая генерация профилирующих библиотек

Для каждой из этих библиотек процессы построения очень подобны. Во-первых, должны быть написаны профилирующие версии `MPI_Init` и `MPI_Finalize`. Профилирующие версии других процедур MPI подобны по стилю. Код каждой из них выглядит подобно

```
int MPI_Xxx (...)
{
сделать что-либо для профилирующей библиотеки
retcode = PMPI_Xxx ( . . . );
сделать что-либо еще для профилирующей библиотеки
return retcode;
}
```

Мы создаем эти процедуры только написанием частей “сделать что-либо”, схематически, а затем обрамляем их автоматически вызовами `PMPI_`. Поэтому генерация профилирующих библиотек очень проста. См. также файл `README` в каталоге `'mpe/profiling/wrappergen'` или приложение A.

Примеры написания заготовок обрамления находятся в подкаталоге `'mpe/profiling/lib'`. Здесь Вы найдете исходные коды (файлы `.w`) для создания трех профилирующих библиотек, описанных выше. Пример make-файла для их испытания находится в каталоге `'mpe/profiling/examples'`.

5.8 Инструменты управления профилирующей библиотекой

Простые профилирующие оболочки для `mpich` распространяются как код определения оболочки. Этот код пропускается через утилиту `wrappergen` для генерации кода C (см. разд. 5.7). Любое количество определений оболочки можно использовать совместно, так что возможен любой уровень вложенности профилирующих обрамлений при использовании `wrappergen`.

Несколько примеров определений обрамления представлены в `mpich`:

timing Использует `MPI_Wtime ()` для определения общего количества вызовов каждой функции MPI, и времени, затраченного на эту функцию. Она просто проверяет таймер перед и после вызова функции.

Она не отнимает время, потраченное на вызов других функций.

logging Создает log-файл для всех вызовов функций pt2pt.

vismess Создает окно X, которое выдает простую визуализацию всех прошедших сообщений.

allprof Все вышеперечисленное. Этот пример показывает, как можно комбинировать профилирующие библиотеки.

Замечание: Эти обрамления не используют никаких специальных возможностей `mpich`, кроме графики и регистрации MPE в `'vismess'` и `'logging'` соответственно. Они должны работать с любой реализацией MPI.

Вы можете встраивать эти обрамления вручную в Ваше приложение, что потребует трех этапов в построении Вашего приложения:

- Создание исходного кода для требуемого обрамления с помощью `wrappergen`. Это разовая задача.
- Компиляция кода обрамления. Убедитесь в наличии необходимых параметров строки компиляции. `'vismess'` и `'logging'` требуют библиотеку MPE (`'-lmpe'`), а определение оболочки `'vismess'` требует наличия `DMPE_GRAPHICS` во флагах компилятора C.
- Компоновка скомпилированного кода оболочки, профилирующей версии библиотеки mpi, и любых других необходимых библиотек (`'vismess'` требует X) с Вашим приложением. Нужной командой будет:

```
$ (CLINKER) <объектные файлы приложения...> \
<объектный код обрамления> \
<другие необходимые библиотеки (-lmpe)> \
<профилирующая бибилотека mpi (-lpmpich)> \
<стандартная библиотека mpi (-lmpi)>
```

Чтобы упростить ее, были созданы несколько примеров разделов make-файлов в каталоге `'mpe/profiling/lib'`:

```
Makefile.timing - оболочка timing
Makefile.logging - оболочка logging
Makefile.vismess - оболочка animated messages
Makefile.allprof - оболочка timing, logging, and vismess
```

Для использования этих фрагментов make-файлов:

1. (Необязательно) Добавьте в список зависимостей Вашего приложения

```
$(PROF_OBJ)
myapp: myapp.o $(PROF_OBJ)
```

2. Добавьте $(PROF_FLG) в строку компиляции CFLAGS:

```
CFLAGS = -O $(PROF_FLG)
```

3. Добавьте $(PROF_LIB) в Ваш путь компоновки, после объектного кода Вашего приложения, но перед главной библиотекой MPI:

```
$(CLINKER) myapp.o -L$(MPIR_HOME)/lib $(PROF_LIB) -lmpich
```

4. (Необязательно) Добавьте $(PROF_CLN) в Вашу мишень clean:

```
rm -f *.o *  myapp $(PROF_CLN)
```

5. Включите нужный фрагмент make-файла в Ваш make-файл:

```
include $(MPIR_HOME)/mpe/profiling/lib/Makefile.logging
```

6 Отладка программ MPI встроенными средствами

Отладка параллельных программ, как известно, трудна, и у нас нет волшебного решения этой проблемы. Тем не менее, мы встроили в `mpich` некоторые возможности, которые можно использовать в отладке программ MPI.

6.1 Обработчики ошибок

Стандарт MPI определяет механизм установки своих собственных обработчиколв ошибок, и определяет поведение двух предопределенных обработчиков `MPI_ERRORS_RETURN` и `MPI_ERRORS_ARE_FATAL`. Мы включили как часть библиотеки MPE еще два обработчика ошибок для облегчения использования `dbx` в отладке программ MPI.

```
MPE_Errors_call_dbx_in_xterm
MPE_Signals_call_debugger
```

Эти обработчики ошибок находятся в каталоге MPE. Опция `configure` (`-mpedbg`) включает эти обработчики ошибок в стандартные библиотеки MPI, и разрешает аргумент командной строки `-mpedbg`, чтобы сделать `MPE_Errors_call_dbx_in_xterm` обработчиком ошибок по умолчанию (вместо `MPI_ERRORS_ARE_FATAL`).

6.2 Аргументы командной строки для `mpirun`

`mpirun` предлагает некоторую помощь в запуске программ с отладчиком.

```
mpirun -dbg=<name of debugger> -np 2 program
```

запускает `program` на двух машинах с запущенным выбранным отладчиком на локальной машине. Существует пять скриптов отладки, вложенных в `mpich`, которые будут находиться в каталоге '`mpich/bin`' после выполнения `make`. Они называются `mpirun_dbg.%d`, где `%d` может заменяться на `dbx, ddd, gdb, totalview, xxgdb`. Соответствующий скрипт вызывается при использовании опции `-dbg` вместе с `mpirun`.

6.3 Аргументы MPI для приложений

Они в настоящее время недокументированы и некоторые требуют опций `configure` для своего определения (как `-mpipktsize` и `-chmemdebug`). Опция `-mpiversion` полезна для определения того, как сконфигурирована Ваша версия `mpich` и какая это версия.

-mpedbg Если возникает ошибка, запускается `xterm`, соединяющийся с процессом, который сгенерировал ошибку. Требует, чтобы `mpich` конфигурировался с `-mpedbg` и работает только на некоторых системах рабочих станций.

-mpiversion Выводит версию и аргументы конфигурирования для используемой реализации `mpich`.

Эти аргументы предназначены для программы, а не для `mpirun`. То есть,

```
mpirun -np 2 a.out -mpiversion
```

6.4 Аргументы p4 для устройства `ch_p4`

При использовании устройства `ch_p4` можно использовать для контроля за поведением программы ряд аргументов командной строки.

6.4.1 Отладка p4

Если Ваша конфигурация `mpich` использует `-device=ch_p4`, то Вам доступны некоторые из возможностей отладки p4. Наиболее полезными из них являются аргументы командной строки для приложения. Таким образом

```
mpirun -np 10 myprog -p4dbg 20 -p4rdbg 20
```

приведет к выводу во время выполнения в стандартный вывод информации о трассировке программы на уровне 20. Для получения информации о том, что выводится на конкретном уровне, см. руководство пользователя p4 [2].

Если в командной строке указано `-p4norem`, `mpirun` не будет реально запускать процессы. Главный процесс выводит сообщение, подсказывающее пользователю, что он может сделать. Целью этой опции является возможность позволить пользователю, например, запустить удаленные процессы под его любимым отладчиком. Опция имеет смысл лишь тогда, когда процессы запускаются удаленно, например, в сети рабочих станций. Отметьте, что это аргумент программы, а не `mpirun`. Например, для запуска `myprog` таким способом, используйте

```
mpirun -np 4 myprog -p4norem
```

Для запуска `cpi` двумя процессами, причем второй процесс выполняется под отладчиком, Вам нужно ввести команду вида

```
mpirun -np 2 cpi -p4norem
ожидание процесса с машины shakey.mcs.anl.gov:
/home/me/mpich/examples/basic/cpi sys2.foo.edu 38357 −p4amslave
```

на первой машине и

```
% gdb cpi
GNU gdb 5.0
Copyright 2000 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type “show copying” to see the conditions.
```

```
There is absolutely no warranty for GDB. Type “show warranty” for details.
This GDB was configured as “i586-mandrake-linux”...
(gdb) run sys2.foo.edu 38357 -p4amslave
Запуск пргаммы: /home/me/mpich/examples/basic/cpi sys2.foo.edu 38357 -p4amslave
```

6.4.2 Установка рабочего каталога для устройства p4

По умолчанию рабочим каталогом для процессов, запущенных удаленно на устройстве `ch_p4` будет тот же каталог, что и для двоичных файлов. Для определения рабочего каталога используйте `-p4wdir`, как указано ниже:

```
mpirun -np 4 myprog -p4wdir myrundir
```

6.5 Аргументы командной строки для приложений

Аргументы в командной строке, котрые следуют за именем приложения и не относятся к системе `mpich` (не начинаются с `-mpi` или `-p4`) пропускаются ко всем процессам приложения. Например, если Вы выполните

```
mpirun -echo -np 4 myprog -mpiversion -p4dbg 10 x y z
```

то `-echo -np 4` интерпретируется `mpirun` (отображение для `mpirun` и запуск 4 процессов), `-mpiversion` интерпретируется `mpich` (каждый процесс выводит информацию о конфигурации), `-p4dbg 10` интерпретируется устройством p4, если Ваша версия сконфигурирована с `-device=ch_p4` (устанавливает уровень отладки p4 в 10), a x, y, z пропускаются к приложению. К тому же, `MPI_Init` обрезает все аргументы, не принадлежащие приложению, так что после вызова `MPI_Init` в Вашей программе на C вектор аргументов `argv` содержит лишь

```
myprog x y z
```

и Ваша программа может обрабатывать свои собственные аргументы командной строки. Отметьте, что массив аргументов для программ Фортрана и Фортрана 77 будет содержать команды `mpich`, поскольку в Фортране не определен стандартный механизм для доступа или модификации командной строки. обычным способом.

Не существует возможности предоставить различным процессам различные аргументы командной строки.

6.6 Запуск задач через отладчик

Опция `-dbg=<name of debugger>` для `mpirun` заставляет процессы запускаться под управлением выбранного отладчика. Например, вызов

```
mpirun -dbg=gdb
```

или

```
mpirun -dbg=gdb a.out
```

вызывает скрипт `mpirun_dbg.gdb`, находящийся в каталоге ‘`mpich/bin`’. Этот скрипт захватывает правильные аргументы, вызывает отладчик `gdb`, и запускает первый процесс под `gdb`, если это возможно.

Существует четыре скрипта отладчика; `gdb, xxgdb, ddd` и `totalview`. Их иногда необходимо исправлять, что зависит от Вашей системы. Существует также скрипт отладчика `dbx`, который должен быть отредактирован, поскольку команды `dbx` различаются для версий. Вы можете также использовать эту опцию для вызова другого отладчика; например, `-dbg=mydebug`. Все, что Вам необходимо, это написать скрипт `'mpirun_dbg.mydebug'`, который совпадает по формату с поставляемыми файлами скриптов, и поместить его в каталог `'mpich/bin'`. Дополнительная информация по использованию отладчика Totalview с `mpich` приведена в разделе 7.

6.7 Запуск отладчика при возникновении ошибки

Часто очень удобно запускать отладчик, когда в программе возникает ошибка. Если `mpich` сконфигурирован с опцией `--enable-mpedbg`, то добавление опции командной строки `mpedbg` к программе вызовет попытку `mpich` запустить отладчик (обычно `dbx` или `gdb`), когда возникает ошибка, которая генерирует сигнал (такой как `SIGSEGV`). Например

```
mpirun -np 4 a.out -mpedbg
```

Если Вы не уверены, что Ваш `mpich` поддерживает эту возможность, Вы можете использовать `-mpiversion` и убедится, что `mpich` создан с опцией `--enable-mpedbg`.

6.8 Присоединение отладчика к запущенной программе

На кластере рабочих станций Вы часто можете присоединить отладчик к запущенному процессу. Например, отладчик `dbx` может принять идентификатор процесса (pid), который Вы можете получить с помощью команды `ps`. Форма вызова

```
dbx a.out 1234
```

или

```
dbx -pid 1234 a.out
```

где 1234 - идентификатор процесса. Можно также присоединить к работающей программе отладчик TotalView (см. разд. 7.3 ниже).

6.9 Сигналы

Вообще, пользователю рекомендуется избегать использования сигналов в программах MPI. Страница `man` для `MPI_Init` описывает сигналы, котрые испоьзуются реализацией MPI; они не должны изменяться пользователем.

Поскольку Unix не связывает сигналы в цепь, существует возможность, что различные пакеты могут использовать те же самые сигналы, вызывая ошибки в программе. Например, по умолчанию, устройство `ch_p4` использует `SIGUSR1`; некоторые пакеты потоков также используют `SIGUSR1`. Если у Вас возникла такая ситуация, см. руководство по инсталляции `mpich` об информации, как выбрать другие сигналы для использования в `mpich`.

В некоторых случаях, Вы можете изменить сигнал перед вызовом `MPI_Init`. В этих случаях Ваш обработчик сигналов будет вызван после того, как реализация `mpich` отреагирует на сигнал. Например, если Вы хотите изменить поведение `SIGSEGV` для вывода сообщения, Вы можете установить этот обработчик сообщения перед вызовом `MPI_Init`. Для устройств типа `ch_p4`, которые обрабатывают `SIGSEGV`, это вызовет реакцию на Ваш сигнал после того, как его обработает `mpich`.

6.10 Инструменты отладки

Расширяемые инструменты Unix - это набор для управления сетью рабочих станций, такой, как мультипроцессорная система. Они включают в себя программы для поиска процессов в кластере и выполнения над ними операций (таких, как присоединение отладчика к каждому процессу, которым Вы управляете, и который запускает отдельную программу). Они не являются частью MPI, но могут быть полезны при работе с кластерами рабочих станций. Версия MPI для этих инструментов находится на `www.mcs.anl.gov/sut`; эта реализация хорошо работает с устройством `ch_p4mpd` для `mpich`.

7 Отладка программ MPI с помощью TotalView

TotalView© является мощным коммерческим переносимым отладчиком для параллельных и многопоточных программ, доступным с `http://www.etnus.com`. TotalView воспринимает множество реализаций MPI, включая `mpich`. Это означает, что при наличии установленного в Вашей системе TotalView очень легко запустить Вашу программу для `mpich` под управлением TotalView, даже если Вы работаете на нескольких машинах, легко управлять Вашими процессами вместе или по отдельности через удобный графический интерфейс TotalView, и даже контролировать внутренние структуры данных `mpich` в очередях сообщений [3]. Общая модель работы TotalView будет знакома пользователям отладчиков, базирующихся на командной строке, таких, как `gdb` или `dbx`.

7.1 Подготовка `mpich` для отладки в TotalView

См. *руководство по инсталляции* об инструкциях по конфигурированию `mpich`, чтобы TotalView мог показать очереди сообщений.

7.2 Запуск программы `mpich` под управлением TotalView

Для запуска параллельной программы под управлением TotalView, просто добавьте `'-dbg=totalview'` к Вашим аргументам для `mpirun`:

```
mpirun -dbg=totalview -np 4 cpi
```

TotalView запустится и Вы можете начать выполнение программы, нажав `'G'`. Появится окно с вопросом, хотите ли Вы остановить процессы после выполнения `MPI_Init`. Вы можете посчитать более удобным ответить "нет" и установить свою собственную точку останова после `MPI_Init` (см. разд. 7.4). Таким образом, после остановки процесс будет находиться в некоторой строке программы вместо неопределенного места внутри `MPI_Init`.

7.3 Присоединение к работающей программе

TotalView можно присоединить к работающей программе, что особенно полезно, если Вы подозреваете, что Ваш код имеет блокировки. Чтобы сделать это, запустите TotalView без аргументов, а затем нажмите `'N'` в главном окне. Это приведет к появлению списка процессов, к которым Вы можете присоединиться. Если Вы углубляетесь в начальный процесс `mpich` в этом окне, TotalView также собирает все другие процессы `mpich` (даже не являющиеся локальными). (См. руководство по TotalView о деталях этого процесса).

7.4 Отладка с помощью TotalView

Вы можете установить точки останова, отметив мышью поле слева от номера строки. Большая часть графического интерфейса TotalView интуитивно понятна. Вы выбираете элементы левой кнопкой мыши, вызываете меню действий средней кнопкой и "углубляетесь" в функции, переменные, структуры, процессы и т.д. правой кнопкой. Нажатие `cntl-?` в любом окне TotalView вызывает справочную информацию по этому окну. В исходном окне TotalView это приводит к появлению общей справки. Полная документация (*Руководство пользователя* TotalView) доступно на веб-сайте Etnus.

Вы переключаетесь от просмотра одного процесса к другому с помощью стрелок в верхнем правом углу главного окна, или непосредственным выбором (левой кнопкой) процесса в корневом окне для фокусировки существующего окна на этом процессе, или углублением (правой кнопкой) через процесс в корневом окне, чтобы открыть новое окно для выбранного процесса. Все "быстрые клавиши" указаны в меню, вызываемом средней кнопкой. Команды в основном знакомые. Специфической для MPI является команда `'m'`, которая показывает очереди сообщений, ассоциированые с процессом.

Заметьте также, что при использовании функции MPI-2 `MPI_Comm_set_name` для коммуникатора, TotalView будет показывать это имя, всякий раз показывая информацию о коммуникаторе, позволяя легко определить каждый коммуникатор.

8 Другая документация MPI

Информация о MPI доступна из множества источников. Некоторые из них, особенно Web-страницы имеют указатели на другие ресурсы.

- Сам стандарт:
  - Технический отчет [4]
  - Postscript и HTML на `www.mpi-forum.org`
  - Статья в журнале: Выпуск 1994 г. Journal of Supercomputing Applications [14]
- Дискуссии на форумах MPI
  - Email-дискуссии, а также текущая и ранние версии стандарта доступны на `www.netlib.org`. Дискуссии по MPI-2 доступны на `www.mpi-forum.org`.
- Книги:
  - *Using MPI: Portable Parallel Programming with the Message-Passing Interface, Second Edition*, by Gropp, Lusk, and Skjellum [10]
  - *Using MPI-2: Advanced Features of the Message-Passing Interface*, by Gropp, Lusk, and Thakur [11]
  - *MPI - The Complete Reference: Volume 1, The MPI Core*, by Snir, et al. [16]
  - *MPI - The Complete Reference: Volume 2, The MPI-2 Extensions*, by Gropp, et al. [5]
  - *Parallel Programming with MPI*, by Peter S. Pacheco. [15]
- Конференция:
  - `comp.parallel.mpi`

- Почтовые рассылки:
  - `mpi-comments@cs.utk.edu`: Список для дискуссий на форуме MPI.
  - `mpi-impl@mcs.anl.gov`: Список дискуссий для разработчиков.
  - `mpi-bugs@mcs.anl.gov`: Адрес для сообщения об ошибках `mpich`.
- Реализации, доступные через Web:
  - `mpich` доступен с `http://www.mcs.anl.gov/mpi/mpich` или с анонимного `ftp.mcs.anl.gov` в каталоге '`pub/mpi/mpich`', файл '`mpich.tar.gz`'.
  - LAM доступен с `http://www.lam-mpi.org`.
- Репозитарий тестового кода:
  - `ftp://ftp.mcs.anl.gov/pub/mpi/mpi-test`

9 В случае неудачи

Этот раздел описывает некоторые общие возникающие проблемы и их решения. Он также описывает машинно-зависимые случаи. Присылайте любые проблемы, с которыми Вы не можете справиться после чтения этого раздела на `mpi-bugs@mcs.anl.gov`.

Пожалуйста, укажите:

- Версию `mpich` (т.е., 1.2.2)
- Вывод после запуска Вашей программы с аргументом -mpiversion (т.е., `mpirun -np 1 a.out -mpiversion`)
- Вывод

  ```
  uname -a
  ```

  для Вашей системы. Если Вы работаете на системе SGI, укажите также вывод

  ```
  hinv
  ```

- Если проблема возникает со скриптом типа `mpirun` или `configure`, запустите скрипт с аргументом `-echo` (т.е., `mpirun -echo -np 4 a.out`)
- Если Вы используете сеть рабочих станций, пришлите также вывод `bin/tstmachines`

Каждый раздел организован в формате вопрос-ответ, вначале с вопросами, относящимися к нескольким средам (рабочая станция, операционная система и т.д.), а затем с вопросами, относящимися к отдельным средам. Проблемы с кластерами рабочих станций собраны вместе.

9.1 Проблемы компиляции или компоновки программ на Фортране

9.1.1 Общие

1. В: При компоновке тестовой программы генерируется следующее сообщение:

   ```
   f77 -g -o secondf secondf.o -L/usr/local/mpich/lib/sun4/ch_p4 -lmpich
   invalid option -L/usr/local/mpich/lib/sun4/ch_p4
   ld: -lmpich: No such file or directory
   ```

   О: Эта программа `f77` не принимает команду `-L` для установки пути поиска библиотек. Некоторые системы предлагают скрипт shell для `f77`, очень ограниченный по своим возможностям. Чтобы обойти это, используйте полный путь к библиотекам вместо опции `-L`:

   ```
   f77 -g -o secondf secondf.o /usr/local/mpich/lib/sun4/ch_p4/libmpich.a
   ```

   Начиная с версии `mpich` 1.2.0, `configure` для `mpich` пытается определить корректную опцию для определения путей библиотек для компилятора Фортрана. Если Вы обнаружили, что `configure` для `mpich` делает ошибки, пожалуйста сообщите о них по адресу `mpi-bugs@mcs.anl.gov`.

2. В: При компоновке программ на Фортране выводятся непонятные обозначения вида:

   ```
   f77 -c secondf.f
   secondf.f:
   MAIN main:
   f77 -o secondf secondf.o -L/home/mpich/lib/solaris/ch_shmem -lmpich
   Undefined first referenced
   symbol in file
   getdomainname /home/mpich/lib/solaris/ch_shmem/libmpi.a (shmempriv.o)
   ld: fatal: Symbol referencing errors. No output written to secondf
   ```

   С программами на С этого не происходит.

   О: Это означает, что Ваш компилятор С предоставляет Вам библиотеки, которые не может предоставить компилятор Фортрана. Найдите опции для компиляторов С и Фортрана, указывающие, какие из бибилотек используются (или же Вы можете обнаружить такие опции, как `-dryrun`, которая покажет команды, используемые компилятором). Создайте простую программу на С или Фортране и сравните использованные библиотеки (обычно в командной строке `ld`). Попробуйте те, которые представлены для компилятора С, но отсутствуют для компилятора Фортран.

3. В: При попытке компиляции кода Фортрана компилятором Фортран 90 или Фортран 95 получены сообщения об ошибке

   ```
   Error: foo.f, line 30: Inconsistent datatype for argument 1 in MPI_SEND
   ```

   О: Фортран требует, чтобы в двух вызовах одной и той же процедуры типы аргументов совпадали. Т.е., если Вы вызываете `MPI_SEND` с буфером `REAL` в качестве первого аргумента, то вызов его с буфером `INTEGER` в качестве первого аргумента компилятор Фортрана воспринимает как ошибку. Некоторые компиляторы Фортрана 77 могут пропустить это; большинство компиляторов Фортран 90 или Фортран 95 проверяют это. Существует два решения. Одно из них - использование модуля MPI (в версии с "аргументами по выбору": используйте опцию `-choicemod` для `mpif90`); другим

является указание опции компилятору Фортрана 90, допускающей несовпадение аргументов. Использование модуля MPI более предпочтительно. Пользователи Фортрана 77 могут иногда увидеть подобные сообщения, особенно в последних версиях g77. Опция `-Wno-globals` запрещает появление этих предупреждений.

9.2 Проблемы компоновки программ на C

9.2.1 Общие

1. В: При компоновке программ выводятся сообщения о неопределенных `__builtin_saveregs`.

   О: Возможно в Вашей системе процедуры C и Фортрана несовместимы (например, при использовании `gcc` и фирменного компилятора Фортрана). Если Вы не планируете использование Фортрана, простейший способ исправить это - перестроить все с опцией `-nof77` для `configure`.

   Вы также должны попытаться сделать Ваш компилятор на C совместимым с компилятором Фортрана.

   Одной из простых, но неэлегантных возможностей является использование `f2c` для преобразования Фортрана в C с последующей компиляцией компилятором C. Если Вы выбрали этот способ, помните, что каждая процедура Фортрана должна компилироваться с помощью `f2c` и компилятора C.

   С другой стороны, Вы можете использовать различные опции (см. страницы `man` для Ваших компиляторов), чтобы увидеть, какие библиотеки они добавляют при компоновке. Добавьте эти библиотеки в строку компоновки другого компилятора. Если Вы обнаружили работающий набор библиотек, отредактируйте соответствующие скрипты (например, `mpicc`). `mpich` пытается найти все необходимые библиотеки, но это не всегда получается.

9.2.2 Sun Solaris

1. В: При компоновке на Solaris выводятся сообщения об ошибке:

```
cc -g -o testtypes testtypes.o -L/usr/local/mpich/lib/solaris/ch_p4 -lmpich
    -lsocket -lnsl -lthread
ld: warning: symbol `_defaultstkcache' has differing sizes:
    (file /usr/lib/libthread.so value=0x20; file /usr/lib/libaio.so
    value=0x8);
    /usr/lib/libthread.so definition taken
```

   О: Это ошибка в Solaris 2.3, исправленная в Solaris 2.4. Должна существовать заплатка для Solaris 2.3; свяжитесь с Sun для более полной информации.

9.2.3 HPUX

1. В: При компоновке на HPUX выводятся сообщения об ошибке:

```
cc -o pgm pgm.o -L/usr/local/mpich/lib/hpux/ch_p4 -lmpich -lm
/bin/ld: Unsatisfied symbols:
sigrelse   (code)
sigset   (code)
```

```
sighold   (code)
** Error code 1
```

О: Вам нужна опция компоновки `-lV3`. Устройство p4 на HP использует сигналы System V; они реализуются библиотекой `'V3'`.

9.2.4 LINUX

1. В: При компоновке программы на Фортране получается

```
Linking:
foo.o (.data+0x0): undefined reference to `pmpi_wtime_'
```

О: Это ошибка в компиляторе `pgf77` (вызванная ошибкой в команде `ld` для Linux). Вы можете исправить это либо добавлением `-lpmpich` к строке компоновки, либо изменением `'mpif.h'`, чтобы удалить выражения `external pmpi_wtime, pmpi_wtick`.

`configure` для `mpich` пытается определить, могут ли `pmpi_wtime` и `pmpi_wtick` быть определены в `'mpif.h'` и удалает их, если возникает проблема. Если это произошло и Вы используете `pmpi_wtime` или `pmpi_wtick` в Вашей программе, Вы должны определить их как функции, возвращающие значения двойной точности.

9.3 Проблемы при запуске программ

9.3.1 Общие

1. В: При попытке запуска программы с помощью

```
mpirun -np 2 cpi
```

появляется сообщение об ошибке или программа зависает.

О: В некоторых системах, таких, как IBM SP, существует много взаимно исключающих способов запуска параллельных программ; каждая система выбирает способ, подходящий для нее. Скрипт `mpirun` пробует наиболее общие методы, но может сделать неправильный выбор. Используйте опции `-v` или `-t` для `mpirun`, чтобы увидеть, как он пытается запустить программу, а затем сравните это со специфическими инструкциями для Вашей системы. Вам может понадобиться адаптировать код `mpirun` для собственных нужд. См. также следующий вопрос.

2. В: При попытке запуска программы с помощью, например, `mpirun -np 4 cpi`, появляется

```
Используйте : mpirun [опции]<программа> [<узлы выполнения>][-- <аргументы>]
```

или

```
mpirun [опции]<схема>
```

О: В Вашем пути поиска присутствует команда `mpirun`, не принадлежащая `mpich`. Выполните команду

```
which mpirun
```

чтобы увидеть, какая команда `mpirun` была действительно найдена. Решением будет либо смена порядка каталогов в пути поиска, чтобы поместить версию `mpirun` для `mpich` первой, лиоб определить псевдоним `mpirun`, использующий абсолютный путь. Например, в `csh` shell, Вы можете выполнить

```
alias mpirun /usr/local/mpich/bin/mpirun
```

чтобы установить `mpirun` на версию для `mpich`.

3. В: При попытке запуска большого количества процессов в сети рабочих станций выводится сообщение

   ```
   p4_error: latest msg from perror: Слишком много открытых файлов
   ```

   О: Существует ограничение на количество открытых дескрипторов файла. На некоторых системах Вы можете увеличить это ограничение самостоятельно; на других Вам может помочь системный администратор. Вы можете экспериментировать с безопасным сервером, но это не полное решение. Сейчас мы работаем над более расширяемым механизмом запуска для новой реализации.

4. В: При попытке запуска программы на более чем одном процессоре выводится сообщение об ошибке

   ```
   mpirun −np 2 cpi
   /home/me/cpi: ошибка загрузки разделяемых библиотек: libcxa.so.1: невозможно
   открыть разделяемый объектный файл : Файл или каталог не существует
   ```

   Проблем при запуске одного процесса нет.

   О: Это означает, что некоторые разделяемые библиотеки, используемые системой, не обнаруживаются удаленными процессорами. Существует две возможности:

   (a) Разделяемые библиотеки не установлены на удаленном процессоре. Чтобы исправить это, попросите Вашего системного администратора установить библиотеки.

   (b) Разделяемые библиотеки не упомянуты в пути поиска по умолчанию. Это может случиться, если путь содержится в переменной окружения, установленной в Вашем текущей shell, но который не входит в путь на удаленной системе. Исправления в этом случае тяжелее, поскольку Вы должны связать размещение разделяемых библиотек и исполняемых файлов (главным недостатком в большинстве разделяемых библиотек Unix является то, что исполняемые файлы по умолчанию не запоминают, где были найдены разделяемые библиотеки при компоновке). Простейшей поправкой будет попросить Вашего системного администратора поместить необходимые разделяемые библиотеки в один из каталогов, который просматривается по умолчанию. Если это невозможно, то Вы должны сами помочь компилятору и компоновщику.

   Многие компоновщики предоставляют способ определения пути поиска для разделяемых библиотек. Трудность состоит в (a) передаче этой команды программе-компоновщику и (b) определении всех необходимых библиотек.

   Например, в системах на Linux, команда компоновщика для определения пути поиска разделяемых библиотек - это `-rpath path`, т.е. `-rpath /usr/lib:/usr/local/lib`. Чтобы передать эту команду компоновщику посредством компилятора Intel C `icc`, используется команда `-Qoption,link,-rpath,path`. По умолчанию, компоновщик Linux проверяет каталог '`/usr/lib`' и каталоги, определенные в переменной окружения `LD_LIBRARY_PATH`. Поэтому, чтобы заставить компоновщик включить путь для разделяемых библиотек, Вы можете использовать

```
mpicc -o cpi cpi -Qoption,link,-rpath,$LD_LIBRARY_PATH:/usr/lib
```

Если это сработает, то попробуйте изменить значение `LDFLAGS` в скрипте `mpicc`, чтобы включить эту опцию.

К сожалению, каждый компилятор использует различные способы передачи этих аргументов компоновщику, а каждый компоновщик использует различный набор аргументов для определения пути поиска разделяемых библиотек. Вам может понадобиться просмотреть документацию Вашей системы об этих опциях.

5. В: При попытке запуска `cpilog` получено следующее сообщение:

```
ld.so.1: cpilog: fatal: libX11.so.4: невозможно открыть файл: errno 2
```

О: Версия X11, обнаруженная `configure`, не установлена корректно. Это обычная проблема для систем Sun/Solaris. Возможно, что Ваши машины на Solaris выполняют слегка различающиеся версии. Вы можете попробовать статическую компоновку (`-Bstatic` на Solaris).

С другой стороны, можно попробовать добавить следующие строки в Ваш `'.login'` (предполагая C shell):

```
setenv OPENWINHOME /usr/openwin
setenv LD_LIBRARY_PATH /opt/SUNWspro/lib:/usr/openwin/lib
```

(Вы можете прежде проконсультироваться с Вашим системным администратором, чтобы убедиться, что пути в Вашей системе правильны). Убедитесь, что Вы добавили их *перед* строками вида

```
if ($?USER == 0 || $?prompt == 0) exit
```

6. В: Программа завершается с ошибкой при попытке записи в файл.

О: Если Вы открыли файл *перед* вызовом `MPI_INIT`, поведение MPI (не только `mpich`-реализации MPI) неизвестно. На устройстве `ch_p4` только нулевой процесс (в `MPI_COMM_WORLD`) может иметь открытые файлы; другие процессы не могут открывать файлы. Переместите операции, которые открывают файлы и взаимодействуют с внешним миром после `MPI_INIT` (и перед `MPI_FINALIZE`).

7. В: Программа пытается стартовать бесконечно долго.

О: Это может быть вызвано несколькими проблемами. На системах с динамически компонуемыми исполняемыми файлами это может быть вызвано проблемами файловой системы, одновременно получившей запросы от многих процессоров о динамически компонуемых частях исполняемых файлов (это отмечено как проблема с некоторыми реализациями DFS). Вы можете попробовать использовать статическую компоновку Вашего приложения.

В сетях рабочих станций долгие времена запуска могут быть из-за времени, используемого для запуска удаленных процессов; см. обсуждение безопасного сервера в разд. 3.2.3 для устройства `ch_p4` или рассмотрите использование устройства `ch_p4mpd`.

9.3.2 Сети рабочих станций

1. В: При использовании `mpirun` выводится сообщение `Permission denied`.

О: Если Вы видите что-нибудь подобное

```
% mpirun -np 2 cpi
Нет прав доступа.
```

или

```
% mpirun -np 2 cpi
сокет: ошибка протокола при настройке линии
```

при использовании устройства `ch_p4`, это возможно означает, что у Вас нет прав на использование `rsh` для запуска процессов. Для проверки можно использовать скрипт `tstmachines`. Попробуйте выполнить

```
tstmachines
```

Если это окончилось неудачей, Вам могут понадобиться либо файлы `'.rhosts'` или `'/etc/hosts.equiv'` (Вам нужно поговорить с системным администратором), либо использование сервера p4 (см. разд. 3.2.3). Другой возможной проблемой является выбор программы удаленной shell; в некоторых системах их несколько. Обсудите с Вашим системным администратором, какую версию `rsh` или `remsh` Вам нужно использовать. Если Вы должны использовать `ssh`, см. разд. об этом в *руководстве по инсталляции*.

Если Ваши системные правила позволяют использовать файл `'.rhosts'`, выполните следующее:

(a) Создайте файл `'.rhosts'` в Вашем домашнем каталоге

(b) Смените защиту в нем на права чтения/записи только для пользователя:
`chmod og-rwx .rhosts`.

(c) Добавьте строки в файл `'.rhosts'` для каждого процессора, который Вы желаете использовать. Формат

```
host username
```

Например, если Ваше имя пользователя `doe` и Вы хотите использовать машины `a.our.org` и `b.our.org`, то Ваш файл `'.rhosts'` будет содержать

```
a.our.org doe
b.our.org doe
```

Помните об использовании полных имен машин (некоторые системы этого требуют).

В сетях, где использование файлов `'.rhosts'` не допускается, Вам нужно использовать безопасный сервер для запуска задачи на машинах, которые не являются доверенными к машине, с которой Вы инициируете задачу.

Наконец, Вам может понадобиться использовать нестандартную команду `rsh` внутри `mpich`. `mpich` должен быть переконфигурирован с опцией `-rsh=command_name`, а также, возможно, с `-rshnol`, если команда удаленной shell не поддерживает аргумент `-l`. В системах, использующих Kerberos и/или AFS это может понадобиться. См. разд. в *руководстве по инсталляции* об использовании безопасной shell `ssh`.

Другим источником сообщения "Permission denied." является то, что Вы использовали команду `su` для смены Вашего идентификатора пользователя. На некоторых системах устройство `ch_p4` не будет работать в такой ситуации. Зарегистрируйтесь обычным способом и попробуйте еще раз.

2. В: При использовании `mpirun` выводится сообщение `Try again.`

   О: Если Вы видите что-либо подобное

   ```
   % mpirun -np 2 cpi
   Попробуйте еще раз.
   ```

   это означает, что Вы не можете запустить удаленную задачу командой удаленной shell на определенной машине, даже если Вы можете сделать это обычным способом. Это может означать, что назначенная машина занята, не имеет свободной памяти или не может создать новый процесс. Страница `man` для `rshd` даст Вам подробную информацию. Единственным способом исправить это является просьба к Вашему системному администратору проверить состояние машины, которая выдает это сообщение.

3. В: При запуске устройства `ch_p4` выводятся сообщения об ошибке вида

   ```
   stty: TCGETS: Операция не поддерживается сокетом
   ```

   или

   ```
   stty: tcgetattr: Нет прав доступа
   ```

   или

   ```
   stty: Невозможно присвоить требуемый адрес
   ```

   О: Это означает, что один из Ваших скриптов загрузки (т.е. `'.login'` или `'.cshrc'` или `'.profile'`) имеет незащищенное использование программ `stty` или `tset`. Для пользователей C shell типичным исправлением является проверка инициализации переменных `TERM` или `PROMPT`. Например,

   ```
   if ($?TERM) then
   eval `tset -s -e^\? -k^U -Q -I $TERM`
   endif
   ```

   Другим решением будет добавление

   ```
   if ($?USER == 0 || $?prompt == 0) exit
   ```

   в начало Вашего файла `'.cshrc'` (но после любого кода, который настраивает среду выполнения, такую, как пути к библиотекам (т.е., `LD_LIBRARY_PATH`)).

4. В: При запуске устройства `ch_p4` и запуске либо скрипта `tstmachines` для проверки файла машин, либо тестов `mpich`, появляются сообщения о неожиданном выводе или различиях с ожидаемым выводом. Также появляется дополнительный вывод при запуске программ. Однако, программы MPI выглядят работающими.

   О: Это означает, что один из Ваших скриптов загрузки (т.е. `'.login'`, или `'.cshrc'`, или `'.profile'`, или `'.bashrc'`) имеет незащищенное использование некоторых программ, которые генерируют вывод, таких, как `fortune` или даже `echo`. Для пользователей C shell типичным исправлением является проверка инициализации переменных `TERM` или `PROMPT`. Например,

```
if ($?TERM) then
fortune
endif
```

Другим решением будет добавление

```
if ($?USER == 0 || $?prompt == 0) exit
```

в начало Вашего файла `'.cshrc'` (но после любого кода, который настраивает среду выполнения, такую, как пути к библиотекам (т.е., `LD_LIBRARY_PATH`)).

5. В: Иногда программы завершаются с сообщением

   ```
   poll: ошибка протокола во время создания связи
   ```

   О: Вы можете увидеть это сообщение, если Вы пытаетесь запустить слишком много программ MPI за короткий период времени. Например, в Linux при использовании устройства `ch_p4` (без безопасного сервера или `ssh`), `mpich` может использовать для запуска процессов MPI `rsh`. В зависимости от определенного дистрибутива Linux и его версии, может быть установлен лимит на минимум 40 процессов в минуту. При запуске набора тестов `mpich` или запуске коротких параллельных задач через скрипт возможно превышение этого лимита. Чтобы исправить это, Вы можете сделать следующее:

   (a) Обождать несколько секунд между запусками параллельных задач. Вам может понадобиться ждать до минуты.

   (b) Изменить `'/etc/inetd.conf'`, чтобы разрешить большее количество процессов в минуту для `rsh`. Например, измените

   ```
   shell stream tcp nowait root /etc/tcpd2 in.rshd
   ```

   на

   ```
   shell stream tcp nowait.200 root /etc/tcpd2 in.rshd
   ```

   (c) Использовать устройство `ch_p4mpd` или опцию безопасного сервера для устройства `ch_p4`. Ни одно из них не зависит от inetd.

6. В: При использовании `mpirun` появляется странный вывод вида

   ```
   arch: Файл или каталог не существует
   ```

   О: Обычно это проблема в Вашем файле `'.cshrc'`. Попробуйте команду shell

   ```
   which hostname
   ```

   Если Вы видите тот же самый странный вывод, то Ваша проблема в Вашем файле `'.cshrc'`. У Вас имеется некоторый код в Вашем файле `'.cshrc'`, который предполагает, что Ваша shell присоединена к терминалу.

7. В: При попытке запуска программы выводится

   ```
   p0_4652: p4_error: ошибка открытия в файле procgroup (procgroup): 0
   ```

О: Это указывает, что программа `mpirun` не может создать требуемый входной файл, необходимый для запуска. Наиболее возможной причиной является то, что команда `mpirun` пытается запустить программу, построенную на устройстве `ch_p4` для устройства с разделяемой памятью (`ch_shmem`) или другого.

Попробуйте следующее:

Запустите программу с использованием `mpirun` и аргумента `-t`:

```
mpirun -t -np 1 foo
```

Это покажет, что `mpirun` должен сделать (`-t` - это тестирование). Вы также можете использовать аргумент `-echo`, чтобы точно знать, что выполняет `mpirun`:

```
mpirun -echo -np 1 foo
```

В зависимости от выбора, сделанного при инсталляции `mpich`, Вы должны выбрать версию `mpirun` для определенного устройства вместо "общей" версии. Мы рекомендуем, чтобы префикс инсталляции включал имя устройства, например, `'/usr/local/mpich/solaris/ch_p4'`.

8. В: При попытке запуска программы выводится сообщение:

```
icy% mpirun -np 2 cpi -mpiversion
icy: icy: Файл или каталог не существует
```

О: Ваша проблема в том, что программа удаленной shell - это не `'/usr/lib/rsh'`. Попробуйте следующее:

```
which rsh
ls /usr/*/rsh
```

Возможно, в Вашем пути `'/usr/lib'` указан раньше `'/usr/ucb'` или `'/usr/bin'`. Это указывает на "ограниченную" shell вместо "удаленной" shell. Простейшим исправлением является удаление `'/usr/lib'` из Вашего пути (некоторым людям это нужно); иначе, Вы можете переместить его на место после каталога, содержащего "удаленную" shell `rsh`.

Другим способом будет добавление ссылки на удаленную shell в каталог, более ранний в пути поиска. Например, у меня `'/home/gropp/bin/solaris'` стоит раньше в пути поиска, тогда я могу использовать здесь

```
cd /home/gropp/bin/solaris
ln -s /usr/bin/rsh rsh
```

(предполагая, что удаленная shell - это `'/usr/bin/rsh'`).

9. В: При попытке запуска программы выводится сообщение:

```
пробуем обычную rsh
```

О: Вы используете версию удаленной shell, которая не поддерживает аргумент `-l`. Переконфигурируйте `mpich` с `-rshnol` и пересоберите. Вы можете почувствовать некоторые неудобства при попытке запуска на системах, где у Вас другое имя пользователя. Вы можете также попробовать использовать `ssh`.

10. В: При запуске программы выводятся сообщения

    ```
    ld.so: warning: /usr/lib/libc.so.1.8 имеет более раннюю версию,
    хотя ожидалось 9
    ```

    О: Вы пытаетесь работать на машине с устаревшей версией базовой библиотеки C. К сожалению, некоторые производители не выпускают совместимых разделяемых библиотек между мелкими (или исправленными) версиями их программного обеспечения. Вы должны попросить системного администратора привести все машины к одному и тому же уровню установленного программного обеспечения.

    Временным исправлением, которое Вы можете использовать, является добавление опции времени компоновки, заменяющей динамическую компоновку системных библиотек статической. На некоторых рабочих станциях Sun это опция `-Bstatic`.

11. В: Программа вообще не запускается. Зависает даже `tstmachines`.

    О: Вначале проверьте, работает ли `rsh` вообще. Например, если у Вас есть рабочие станции `w1` и `w2`, и Вы работаете на `w1`, попробуйте

    ```
    rsh w2 true
    ```

    Это должно завершиться очень быстро. Если этого не происходит, попробуйте

    ```
    rsh w1 true
    ```

    (т.е. используйте `rsh` для запуска на той же системе, где Вы работаете). Если Вы получили `Permission denied`, см. соответствующую справку. Если Вы получили

    ```
    krcmd: No ticket file (tf_util)
    rsh: внимание, используется стандартная rsh: невозможно получить данные
    аутентификации Kerberos.
    ```

    то в Вашей системе неправильно установлена `rsh`. Эта проблема наблюдается в некоторых системах FreeBSD. Попросите системного администратора исправить проблему (часто неправильный набор программ `rsh/rshd`).

12. В: При запуске устройства `ch_p4` выводятся сообщения об ошибке

    ```
    больше пользователей, чем очередей сообщений
    ```

    О: Это означает, что Вы пробуете запустить `mpich` в одном режиме, а он сконфигурирован в для работы в другом. В частности, Вы определяете в Вашем файле p4 `procgroup`, что некоторые процессы будут находиться в разделяемой памяти на определенной машине, помещая либо число, большее 0, в первую строку (где оно означает число локальных процессов вместе с оригинальным), либо число, большее 1, в любую ииз следующих строк (где оно указывает общее число процессов, разделяющих память на этой машине). Вы должны либо изменить Ваш файл procgroup, чтобы указать только один процесс в строке, или переконфигурировать `mpich` с помощью

    ```
    configure −−with-device=ch_p4 -comm=shared
    ```

что перенастроит устройство p4 так, чтобы несколько процессов могли разделять память на каждой машине. Причина здесь не только подразумевает то, что в такой конфигурации Вы увидите ожидание занятости на каждой станции, но и что устройство будет переключаться между выбором сокета и проверкой внутренних очередей разделяемой памяти.

13. В: Программы зависают при выполнении `MPI_Init`. О: Существует ряд причин, вызывающих это:

    (a) Одна из рабочих станций, выбранных Вами для запуска не работает (попробуйте `'tstmachines'`, если Вы используете устройство `ch_p4`).

    (b) Вы компонуете программу с использованием пакета потоков FSU; известно, что это вызывает проблемы, особенно в системных вызовах `select`, который является частью Unix и используется `mpich`.

    Другая причина может состоять в том, что Вы используете библиотеку `'-ldxml'` (расширенная математическая библиотека) в системе Compaq Alpha. Известно, что она вызывает зависание `MPI_Init`. На данный момент неизвестен способ обхода этой проблемы; свяжитесь с Compaq для получения исправлений, если Вам необходимо использовать совместно MPI and `'-ldxml'`.

    Причины этой проблемы в том, что устройство `ch_p4` использует `SIG_USR1`, и любая библиотека, которая также использует этот сигнал может перекрываться с дейтсвиями `mpich`, когда используется устройство `ch_p4`. Вы можете перестроить `mpich`, чтобы использовать другой сигнал, с помощью аргумента `--with-device=ch_p4:-listener_sig=SIGNAL_NAME` для `configure` и пересобрать `mpich`.

14. В: Программа (использующая устройство `ch_p4`) завершается с ошибкой

    ```
    p0_2005: p4_error: fork_p4: fork failed: -1
    устройство ch_p4 p4_error: latest msg from perror: Error 0
    ```

    О: Размер исполняемого файла Вашей программы может быть слишком велик. Когда запускается программа для устройства `ch_p4` или `ch_tcp`, она может создавать свою копию для обработки некоторых коммуникационных задач. Из-за способа организации кода, это (хотя и временно) полная копия Вашей оригинальной программы, занимающая то же место. Поэтому, если Ваша программа больше, чем половина всего доступного пространства, Вы получите эту ошибку. В системах SGI Вы можете использовать команду `size`, чтобы определить размер исполняемого файла и `swap -l`, чтобы узнать размер доступного пространства. Заметьте, что `size` возвращает размер в байтах, а `swap -l` возвращает размер в 512-байтовых блоках. Другие системы могут предложить подобные команды. Такая же проблема может возникнуть в IBM SP при использовании устройства `ch_mpl`; причина та же, но происходит из библиотеки IBM MPL.

15. В: Иногда возникает ошибка

    ```
    Неверный формат exec. Неверная архитектура.
    ```

    О: Возможно, Вы используете NFS (Network File System). NFS может не поддерживать файлы, обновляемые временным способом; эта проблема может вызываться созданием исполняемого файла на одной машине и попыткой использования его с другой. Обычно NFS определяет существование нового файла в течение нескольких минут. Вы можете попробовать использовать команду `sync`. `mpirun`, как правило, пытается выполнить команду `sync`, но во многих системах `sync` лишь информативна и не гарантирует, что файловая система стала непротиворечивой.

16. В: Похоже, что существует две копии моей программы, запущенные на каждом узле. Это удваивает потребность в памяти моего приложения. Нормально ли это?

    О: Да, это нормально. В реализации `ch_p4` второй процесс используется для динамической установки связей с другими процессами. Начиная с `mpich` версии 1.1.1, эта возможность может помещаться в отдельный поток на многих архитектурах, и этот второй процесс будет невидимым. Чтобы разрешить это, используйте опцию `-p4_opts=-threaded_listener` в командной строке `configure` для `mpich`.

17. В: `MPI_Abort` иногда не работает в устройстве `ch_p4`. Почему?

    О: Сейчас (в версии 1.2.2) процесс определяет, что другой процесс прерван, толко когда он пытается принимать или посылать сообщение, а прерванный процесс является тем, с которым он соединялся в прошлом. Поэтому, процесс, занятый вычислениями, может не заметить, что один из его коллег выполнил `MPI_Abort`, несмотря на то, что для многих общих примеров коммуникации это не представляет проблемы. Это будет исправлено в дальнейшей реализации.

9.3.3 IBM RS6000

1. В: При попытке запуска на IBM RS6000 с устройством `ch_p4` получается

```
% mpirun -np 2 cpi
Невозможно загрузить программу /home/me/mpich/examples/basic/cpi
Невозможно загрузить библиотеку libC.a[shr.o]
Ошибка: Файл или каталог не найден
```

    О: Это означает, что `mpich` был создан компилятором `xlC`, но некоторые машины в Вашем файле `'util/machines/machines.rs6000'` не имеют установленного `xlC`. Либо инсталлируйте `xlC`, либо пересоберите `mpich` с использованием другого компилятора (`xlc` или `gcc`; преимущество `gcc` в отсутствии лицензионных ограничений).

9.3.4 IBM SP

1. В: При запуске программы на IBM SP выводится:

```
$ mpirun -np 2 hello
ERROR: 0031-124 Невозможно распределить узлы для параллельного исполнения.
Выход ...
ERROR: 0031-603 Распределение менеждера ресурсов для задач: 0, узел:
me1.myuniv.edu, rc = JM_PARTIONCREATIONFAILURE
ERROR: 0031-635 Возвращено ненулевое состояние -1 из pm_mgr_init
```

    О: Это значит, что либо `mpirun` пытается запустить задачи на Вашей SP отличным от поддерживаемого Вашей инсталляцией способом, или существует неисправность в программном обеспечении IBM, управляющем параллельными задачами (все эти сообщения об ошибках происходят от команды IBM `poe`, которую `mpirun` использует для запуска задачи MPI). Свяжитесь с Вашим системным администратором для помощи в исправлении этой ситуации. Ваш системный администратор может использовать

```
dsh -av "ps aux | egrep -i 'poe|pmd|jmd"'
```

с управляющей рабочей станции для поиска случайных задач IBM POE, что может вызывать такое поведение. Файлы `/tmp/jmd_err` на отдельных узлах могут также содержать полезную диагностическую информацию.

2. В: При попытке запуска программы на IBM SP получено сообщение от `mpirun`:

   ```
   ERROR: 0031-214 pmd: chdir </a/user/gamma/home/mpich/examples/basic>
   ERROR: 0031-214 pmd: chdir </a/user/gamma/home/mpich/examples/basic>
   ```

   О: Это сообщения от системы IBM `tbe`, а не от `mpirun`. Они могут быть вызваны несовместимостью между POE, программой автомонтирования (особенно AMD) и shell, особенно если Вы используете не `ksh`. Не существует хорошего решения; IBM часто рекомендует сменить Вашу shell на `ksh`!

3. В: При попытке запуска программы на IBM SP получено

   ```
   ERROR : Невозможно определить каталог сообщений (pepoe.cat) используя текущий
   NLSPATH
   INFO : Если NLSPATH установлен правильно и каталог существует, проверьте
   переменные LANG или LC_MESSAGES
   (C) Открытие каталога сообщений "pepoe.cat" невозможно
   ```

   (и другие вариации, упоминающие `NLSPATH` и "каталог сообщений").

   О: Это проблема в Вашей системе; свяжитесь с Вашей командой поддержки. Обратите внимание на (a) значение `NLSPATH`, (b) ссылки из `'/usr/lib/nls/msg/prime'` к соответствущему языковому каталогу. Сообщения происходят не от `mpich`; они от кода IBM POE/MPL, который использует реализация `mpich`.

4. В: При запуске программы на IBM SP выводится:

   ```
   ERROR: 0031-124 Менее 2 узлов доступно из pool 0
   ```

   О: Это означает, что система IBM POE/MPL не может распределить запрошенные узлы, когда Вы пытаетесь запустить программу; вероятно, систему использует кто-то еще. Вы можете попытаться использовать переменные окружения `MP_RETRY` и `MP_RETRYCOUNT`, чтобы заставить задачу ожидать, пока узлы не станут доступны. Используйте `man poe` для дополнительной информации.

5. В: При запуске программы на IBM SP задача генерирует сообщение

   ```
   Сообщение номер 0031−254 не найдено в каталоге сообщений.
   ```

   а затем завершается.

   О: Если Ваше имя пользователя состоит из 8 символов, Вы можете обнаружить ошибку в среде IBM POE. Единственным исправлением, описанным на данный момент, является испоьзование профиля, в котором имя пользователя состоит из 7 символов или менее. Спросите Вашего представителя IBM о PMR 4017X (poe с идентификаторами пользователя длиной 8 символов не работает) и связанным с ним APAR IX56566.

9.4 Программы завершаются с ошибкой при старте

9.4.1 Общие

1. В: В некоторых системах Вы можете увидеть

```
/lib/dld.sl: Вызов по ссылке неудачен
/lib/dld.sl: Неверный аргумент
```

(Это пример из HP−UX), или

```
ld.so: libc.so.2: не найдена
```

(Это пример из SunOS 4.1; подобные вещи происходят и в других системах).

О: Проблема в том, что Ваша программа использует разделяемые библиотеки, и библиотеки не доступны на некоторых машинах, с которыми Вы работаете. Чтобы исправить это, перекомпонуйте Вашу программу без разделяемых библиотек. Чтобы сделать это, добавьте соответствующие опции командной строки на этапе компоновки. Например, для системы HP и указанной выше ошибки, исправление состоит в использовании на этапе компоновки `-Wl,-Bimmediate`. Для Solaris соответствующей опцией будет `-Bstatic`.

9.4.2 Сети рабочих станций

1. В: Я могу запустить программу, используя небольшое число процессов, однако, когда я запрашиваю больше, чем 4-8 процессов, я не получаю ввод от всех моих процессов и программы никогда не завершаются.

О: Мы наблюдали такую проблему при инсталляциях, использующих AFS. Программа удаленной shell `rsh`, поставляемая с некоторыми системами AFS, ограничивает количество задач, которые могут использовать стандартный вывод. Это предотвращает выход для некоторых процессов, вызывая зависание задачи. Существует четыре возможных пути решения:

   (a) Использовать другую команду `rsh`. Вероятно, Вы можете сделать это, поместив каталог, содержащий не-AFS версию, раньше в Вашем `PATH`. Эта опция может быть недоступна для Вас, что зависит от Вашей системы. Версия не-AFS может находиться в том же месте в '`/bin/rsh`'.

   (b) Использовать безопасный сервер (`serv_p4`). См. его обсуждение в руководстве пользователя.

   (c) Перенаправить весь стандартный вывод в файл. Для этого может использоваться процедура `MPE MPE_IO_Stdout_to_file`.

   (d) Использовать исправленную версию команды `rsh`. Возможным источкником проблемы является неправильное использование системного вызова `select` в команде `rsh`. Если код выполняет нечто вроде

```
int mask;
mask |= 1 « fd;
select ( fd+1, &mask, ... );
```

   вместо

```
fd_set mask;
FD_SET (fd,&mask);
select ( fd+1, &mask, ... );
```

то этот код неправильный (вызов `select` изменился, чтобы разрешить более 32 десткипрторов файлов очень давно, а программа `rsh` (или программист!) не изменились со временем).

Четвертой возможностью является использование AFS-версии `rsh`, которая исправляет эту ошибку. Поскольку мы сами на пользуемся AFS, нам неизвестно, есть ли где нибудь исправленная версия.

2. В: Запускаются не все процессы.

   О: Это может произойти при использовании устройства `ch_p4` и системы, имеющей исключительно малые ограничения на число удаленных shell, принадлежащих Вам. Некоторые системы используют "Kerberos" (пакет безопасности сети), позволяющий иметь только три или четыре удаленных shell; на этих системах размер `MPI_COMM_WORLD` будет ограничен этим же числом (плюс один, если Вы используете и локальную машину).

   Единственный способ обойти это - попытаться использовать безопасный сервер; это описано в руководстве по инсталляции `mpich`. Отметьте, что Вы должны запускать серверы "вручную", поскольку скрипт `chp4_servs` использует удаленную shell для запуска серверов.

9.5 Программы завершаются с ошибкой после запуска

9.5.1 Общие

1. В: Я использую `MPI_Allreduce`, и получаю различные результаты в зависимости от числа используемых процессов.

   О: Коллективные процедуры MPI могут использовать ассоциативность для достижения лучшего параллелизма. Например,

   ```
   MPI_Allreduce ( &in, &out, MPI_DOUBLE, 1, ... );
   ```

   может вычислять

   $$(((((((a+b)+c)+d)+e)+f)+g)+h)$$

   или может вычислять

   $$((a+b)+(c+d))+((e+f)+(g+h))$$

   где a, b, ... - значения in в каждом из восьми процессов. Эти выражения эквивалентны для целых, вещественных и других привычных объектов из математики, но *не* эквивалентны для типов данных, таких, как данные с плавающей точкой, используемых в компьютерах. Ассоциация, которую использует MPI, зависит от количества процессов, таким образом, Вы можете не получить тот же самый результат при использовании различного числа процессов. Отметьте, что Вы не получите неверный результат, а просто другой (большинство программ предполагают, что арифметические операции ассоциативны).

2. В: Компилятор Фортрана завершается с ошибкой `BUS`.

   О: Компилятор C, с помощью которого был построен `mpich` и компилятор Фортрана, который Вы используете, имеют различные правила выравнивания для таких вещей, как `DOUBLE PRECISION` (двойная точность). Например, GNU C компилятор `gcc` может предположить, что все значения `double`

выравниваются по 8-байтовой границе, а язык Фортран требует только такого выравнивания DOUBLE PRECISION для INTEGER, которое делается по 4-байтовой границе.

Хорошего решения не существует. Рассмотрим перестроение mpich с компилятором C, который поддерживает слабые правила выравнивания данных. Некоторые компиляторы Фортрана позволяют Вам задать 8-байтовое выравнивание для DOUBLE PRECISION (например, опции -dalign в -f некоторых компиляторах Фортрана для Sun); заметьте, однако, что это может испортить некоторые корректные программы Фортрана, взломав правила ассоциации хранения в Фортране.

Некоторые версии gcc могут поддерживать -munaligned-doubles; mpich должен быть перестроен с этой опцией, если Вы используете gcc версии 2.7 или выше. mpich пытается определить и использовать опцию, если она доступна.

3. В: Я использую fork для создания нового процесса или создаю новый поток, а программа завершается с ошибкой.

   О: Реализация mpich не полностью совместима с потоками и не не поддерживает ни fork, ни создание нового процесса. Заметьте, что спецификация MPI полностью совместима с потоками, но от реализаций этого не требуется. На данный момент некоторые реализации поддерживают потоки, хотя это уменьшшает производительность реализации MPI (сейчас Вам нужно определить, нужен ли Вам захват потока, даже если захват и освобождение более дорогостоящие).

   Реализация mpich поддерживает вызов MPI_Init_thread; посредством этого вызова, нового для MPI-2, Вы можете определить, какой уровень поддержки потоков позволяет реализация MPI. Начиная с mpich версии 1.2.0 поддерживается только MPI_THREAD_SINGLE. Мы уверены, что версия 1.2.0 и выше поддерживает MPI_THREAD_FUNNELED, и некоторые пользователи используют mpich в этом режиме (особенно вместе с OpenMP), но мы недостаточно протестировали mpich в этом режиме. Будущие версии mpich будут поддерживать MPI_THREAD_MULTIPLE.

4. В: Программы C++ выполняют глобальне деструкторы (или конструкторы) больше раз, чем нужно. Например:

```
class Z {
public:
Z () { cerr « "*Z" « endl; }
Z () { cerr « "+Z" « endl; }
};
Z z;
int main (int argc, char **argv) {
MPI_Init (&argc, &argv);
MPI_Finalize ();
}
```

   при запуске с устройством ch_p4 на два процесса выполняет деструктор дважды для каждого процесса.

   О: Ряд процессов, запущенных перед MPI_Init или после MPI_Finalize не определяются ; Вы не можете полагаться на определенное поведение. В случае ch_p4 порождается новый процесс для обработки запросов соединения; он прекращается после окончания программы.

Вы можете использовать контролирующий процесс в потоке с устройством `ch_p4` или использовать вместо этого устройство `ch_p4mpd`. Заметьте, однако, что этот код непереносим, поскольку он полагается на поведение, которое не определяет стандарт MPI.

9.5.2 HPUX

1. В: Программы Фортрана завершаются с `SIGSEGV` при запуске на рабочих станциях HP.

   О: Попробуйте компилировать и компоновать программы Фортрана с опцией `+T`. Это может быть необходимым, чтобы среда Фортрана корректно обрабатывала прерывания, используемые `mpich` для создания соединений с другими процессами.

9.5.3 Устройство `ch_shmem`

1. В: Программы иногда зависают при использовании устройства `ch_shmem`.

   О: Убедитесь, что Вы компонуете программу со *всеми* корректными библиотеками. Если Вы не используете `mpicc`, попробуйте использовать `mpicc` для компоновки Вашего приложения. Причиной этого является то, что правильная работа версии с разделяемой памятью может зависеть от дополнительных библиотек, предоставляемых системой. Например, в Solaris, должна использоваться библиотека потоков, иначе критичные для корректного функционирования реализации MPI нефункциональные версии процедур mutex будут взяты из `'libc'`.

9.5.4 LINUX

1. В: Процессы завершаются с сообщением об ошибке

   ```
   p0_1835: p4_error: Найдено неработающее соединение при ожидании сообщений: 1
   ```

   О: Дело в том, что реализация TCP на этой платформе решает, что соединение "оборвалось", когда его в действительности еще нет. Текущая реализация `mpich` предполагает, что реализация TCP не закрывает соединений и не имеет процедур для восстановления оборванных соединений. Будущие версии `mpich` будут обходить эту проблему.

   К тому же, некоторые пользователи обнаружили, что ядро Linux для одного процессора более стабильно, чем ядро SMP.

9.5.5 Сети рабочих станций

1. В: Задача работает до окончания, но завершается с сообщением

   ```
   Таймаут при ожидании завершения процессов. Это может вызываться дефектной
   программой rsh (Некоторые версии Kerberos rsh имеют эту проблему).
   Это не проблема P4 или mpich, а проблема операционной среды. Для многих
   приложений эта проблема замедляет прекращение процессов.
   ```

   Что это означает?

   О: Если что-либо вызывает завершение `MPI_Finalize` за время более, чем 5 минут, в этом подозревается реализация `rsh`. `rsh`, используемая в некоторых инсталляциях Kerberos предполагает, что `sizeof (FD_SET) == sizeof (int)`. Это означает, что программа `rsh` предполагает, что наибольшее

значение FD это 32. Если программа использует `fork` для создания процессов, которые запускают `rsh` с поддержкой `stdin, stdout`, и `stderr` порожденных процессов, это предположение неверно, поскольку FD, которые `rsh` создает для сокета, могут быть > 31, если запущено достаточно много процессов. При использовании такой неисправной реализации `rsh` симптомом является то, что задача никогда не завершается, поскольку задачи `rsh` ждут (благодаря `select`) закрытия сокета.

Устройство `ch_p4mpd` исключает эту проблему.

9.6 Проблемы ввода-вывода

9.6.1 Общие

1. В: Я хотел бы, чтобы вывод `printf` появлялся немедленно.

   О: Это зависит от возможностей Вашей системы выполнения C и/или Фортрана. Для C, попробуйте

   ```
   setbuf ( stdout, (char *)0 );
   ```

   или

   ```
   setvbuf ( stdout, NULL, _IONBF, 0 );
   ```

9.6.2 IBM SP

1. В: У меня есть код, запрашивающий данные от пользователя и читающий из стандартного ввода. На системах IBM SP приглашение не появляется, пока пользователь не ответит на него!

   О: Это свойство системы IBM POE. Существует процедура POE, `mpc_flush (1)`, которую Вы можете использовать для очистки вывода. Прочтите страницу man для этой процедуры; она синхронизирована во всей задаче и не может использоваться, пока все процессы в `MPI_COMM_WORLD` не вызовут ее. С другой стороны, Вы всегда можете оканчивать Вывод символом новой строки (\n); это вызовет очистку вывода и также поместит ввод от пользователя на новую строку.

9.6.3 Сети рабочих станций

1. В: Я хотел бы, чтобы стандартный вывод (`stdout`) от каждого процесса выводился на разные строки.

   О: `mpich` не имеет для этого встроенного способа. Кстати, сам по себе он пытается собрать стандартный вывод для Вас. Вы можете сделать следующее:

   (a) Использовать встроенные команды Unix для перенаправления `stdout` из Вашей программы (`dup2`, и т.д.). Процедура `MPE_IO_Stdout_to_file` в `'mpe/src/mpe_io.c'` указывает способ делать это. Помните, что в Фортране подход с использованием `dup2` работает, только если в `stdout` выводит `PRINT` из Фортрана. Это обычный способ, но не универсальный.

   (b) Выводите непосредственно в файлы вместо `stdout` (используйте `fprintf` вместо `printf`, и т.д.). Вы можете создать имя файла, используя ранг процесса. Это наиболее переносимый способ.

9.7 Upshot и Nupshot

Программы `upshot` и `nupshot` требуют определенных версий языков `tcl` и `tk`. Этот раздел описывает только проблему, которые возникают, если эти инструменты успешно созданы.

9.7.1 Общие

1. В: При попытке запуска `upshot` или `nupshot` выдается сообщение

   ```
   Нет имени дисплея и переменой окружения $DISPLAY
   ```

   О: Это проблема с Вашей средой X. `Upshot` - это программа для X. Если имя Вашей рабочей станции `'foobar.kscg.gov.tw'`, то перед запуском программы X, Вы должны выполнить

   ```
   setenv DISPLAY foobar.kscg.gov.tw:0
   ```

   Если Вы работаете на какой-либо другой машине и выводите результат на `foobar`, Вам может понадобиться выполнить

   ```
   xhost +othermachine
   ```

   на `foobar`; это даст `othermachine` разрешение для записи на дисплей `foobar`.

   Если у Вас нет дисплея X (Вы входите в систему с машины под Windows без возможностей X), Вы не можете использовать `upshot`.

2. В: При попытке запуска `upshot` выдается

   ```
   upshot: Команда не найдена.
   ```

   О: Вначале проверьте, находится ли `upshot` в Вашем пути. Вы можете использовать команду

   ```
   which upshot
   ```

   чтобы сделать это.

   Если он находится в Вашем пути, проблема может быть в том, что имя интерпретатора `wish` (используемого `upshot`) слишком длинное для Вашей системы Unix. Проверьте первую строку файла `'upshot'`. Она должна выглядеть примерно так

   ```
   #! /usr/local/bin/wish -f
   ```

   Если она выглядит как

   ```
   #! /usr/local/tcl7.4-tk4.2/bin/wish -f
   ```

   это может быть слишком длинным именем (некоторые системы Unix ограничивают первую строку 32 символами). Чтобы исправить это, Вам нужно поместить ссылку на `'wish'` где-нибудь еще, чтобы имя было достаточно коротким. Также, Вы можете запустить `upshot` через

   ```
   /usr/local/tcl7.4-tk4.2/bin/wish -f /usr/local/mpich/bin/upshot
   ```

9.7.2 HP-UX

1. В: При попытке запуска `upshot` в HP-UX выдаются сообщения об ошибке

   ```
   set: Имя переменной должно начинаться с буквы.
   ```

   или

   ```
   upshot: ошибка синтаксиса в строке 35: ’ (’ неопределена
   ```

   О: Ваша версия HP-UX ограничивает имена shell до очень коротких строк. `upshot` является программой, выполняемой `wish` shell, и по некоторым причинам HP-UX отказывает в исполнении в этой shell, а затем пытается выполнить программу `upshot` в Вашей текущей shell (т.е., ‘`sh`’ или ‘`csh`’), вместо выдачи разумного сообщения об ошибке, что имя команды слишком велико. Существует два способа исправления:

   (a) Добавьте ссылку с более коротким именем, например

   ```
   ln -s /usr/local/tk3.6/bin/wish /usr/local/bin/wish
   ```

   Затем отредактируйте скрипт `upshot` для использования этого имени. Это может потребовать прав доступа `root`, в зависимости от того, где Вы поместили ссылку.

   (b) Создайте обычную программу shell, содержащую строки

   ```
   #! /bin/sh
   /usr/local/tk3.6/bin/wish -f /usr/local/mpi/bin/upshot
   ```

   (с соответствующими именами для исполняемых файлов ‘`wish`’ и ‘`upshot`’). Поместите сообщение об ошибке HP в файл. В настоящее время, сообщение об ошибке здесь неверно; нет причины ограничивать выбор общей shell (в противоположность shell загрузки).

A Автоматическая генерация профилирующих библиотек

Генератор профилирующих обрамлений (`wrappergen`) создан для дополнения профилирующего интерфейса MPI. Он позволяет пользователю создавать любое количество мета-обрамлений, котороые могут применяться к любому количеству функций MPI. Обрамления могут находиться в отдельных файлах, и могут корректно вкладываться друг в друга, так что в отдельной функции может существовать более одного уровня профилирования.

`wrappergen` использует три источника ввода:

1. Список функций, для которых генерируются обрамления.
2. Описания профилируемых функций. Для достижения скорости и простоты разбора используется специальный формат. См. файл '`proto`'. Функции MPI-1 находятся в '`mpi_proto`'. Функции ввода-вывода из MPI-2 находятся в '`mpiio_proto`'.
3. Определение обрамления.

Список функций представляет собой просто файл имен функций, разделенных пробелами. Если он опущен, любые макросы `forallfn` или `fnall` будут расширены на каждую функцию в файле определений.

Если описания функций отсутствуют, используются те, которые находятся в '`mpi_proto`' (это множество с определением `PROTO_FILE` в '`Makefile`').

Опции `wrappergen`:

-w file Добавить файл для использования в списке файлов обрамлений.

-f file Файл содержит список имен функций для профилирования, разделенный пробелами.

-p file Файл содержит определения прототипов специальных функций.

-o file Направить вывод в файл.

Например, для оценки времени каждой операции ввода-вывода, используйте

```
cd mpe/profiling/lib
../wrappergen/wrappergen -p ../wrappergen/mpiio_proto \
-w time_wrappers.w > time_io.c
```

Результирующий код требует только версию `MPI_Finalize` для вывода временных величин. Она может быть написана либо добавлением `MPI_Finalize` и `MPI_Init` к '`mpiio_proto`', либо через простое редактирование версии, полученной при использовании '`mpi_proto`' вместо '`mpiio_proto`'.

A.1 Написание определения обрамления

Определения обрамления сами по себе состоят из кода на C со специальными макросами. Каждый макрос окружен последовательностью {{}}. Следующий макрос распознается `wrappergen`:
{{ `fileno`}}

> Целочисленный индекс представляет, из какого файла обрамления происходит макрос. Это полезно при описании глобальных для файла переменных, чтобы избежать пересечения имен. Рекомендуется, чтобы все идентификаторы, описанные вне функций, оканчивались на _{{ `fileno`}}. Например:

```
    static double overhead_time_{{ fileno}};
```

может расширяться до:

```
    static double overhead_time_0;
```

(конец примера).

```
{{ forallfn <function name escape> <функция A> <функция B> ... }}
...
{{endforallfn}}
```

Код между {{ forallfn}} и {{endforallfn}} копируется один раз для каждой профилируемой функции, исключая перечисленные функции, заменяя <function name escape> именем каждой из функций. Например:

```
    {{forallfn fn_name}} static int {{fn_name}}_ncalls_{{fileno}};
    {{endforallfn}}
```

может расширяться до:

```
    static int MPI_Send_ncalls_1;
    static int MPI_Recv_ncalls_1;
    static int MPI_Bcast_ncalls_1;
```

(конец примера).

```
{{foreachfn <function name escape> <функция A> <функция B> ... }}
...
{{endforeachfn}}
```

{{foreachfn}} - это то же самое, что и {{forallfn}}, за исключением того, что обрамления будут написаны только для точно названных функций. Например:

```
    {{forallfn fn_name mpi_send mpi_recv}}static int {{fn_name}}_ncalls_{{fileno}};
    {{endforallfn}}
```

может расширяться до:

```
    static int MPI_Send_ncalls_2;
    static int MPI_Recv_ncalls_2;
```

(конец примера).

```
{{fnall <function name escape> <функция A> <функция B> ... }}
...
{{callfn}} ...{{endfnall}}
```

{{fnall}} определяет обрамление, используемое для всех функций, исключая названные. wrappergen расширяет их в полное определение функции в традиционном формате C. Макрос

{{callfn}} сообщает wrappergen, где вставить вызов функции, которая будет профилироваться. Должен быть только один экземпляр макроса {{callfn}} в каждом определении оболочки. Макрос, определенный <function name escape> заменяется именем каждой из функций.

Внутри определения обрамления распознаются другие макросы.

{{vardecl <type> <arg> <arg> ... }}

Используйте vardecl для объявления переменных внутри определения обрамления. Если вложенный макрос требует переменных, заданных посредством vardecl с теми же именами, wrappergen создаст уникальные имена добавлением последовательных целых в конец требуемого имени (var, var1, var2, ...), пока не будет создано уникальное имя. Неразумно объявлять переменные в определении обрамления вручную, поскольку имена переменных могут конфликтовать с другими обрамлениями, а определения переменных могут встретиться в коде позже, чем выражения из других обрамлений, что запрещено в классическом и ANSI C.

{{<varname>}}

Если переменная описана через vardecl, то требуемое имя для этой переменной (которое может отличаться от унифицированной формы, которая появится в финальном коде) становится временным макросом, который будет расширен в унифицированную форму. Например,

```
{{vardecl int id}}
```

может расширяться до:

```
int i, d3;
```

(конец примера)

{{<argname>}}

Рекомендуемый, но не необходимый макрос, состоящий из имени одного из аргументов профилируемой функции, расширяется до имени соответствующего аргумента. Этот макрос служит небольшой цели, иной чем вставка, чтобы профилируемая функция имела аргумент с заданным именем.

{{<argnum>}}

Аргументы профилируемой функции могут также адресоваться увеливающимся номером, начиная с 0.

{{returnVal}}

returnVal расширяется до переменной, используемой для хранения возвращаемого результата профилируемой функции.

{{callfn}}

callfn расширяется до вызова профилируемой функции. При вложенных определениях обрамления, это также представляет точку для вставки кода любой вложенной внутренней функции. Порядок вложенности определяется порядком, в котором встречаются обрамления программе wrappergen. Например, если два файла 'prof1.w' и 'prof2.w' каждый содержат два обрамления для MPI_Send, профилированный код, созданный с использованием обоих файлов, имеет вид:

```
int MPI_Send ( аргументы...)
объявление аргументов...
{
/*код перед вызовом функции из обрамления 1 из prof1.w */
/*код перед вызовом функции из обрамления 2 из prof1.w */
/*код перед вызовом функции из обрамления 1 из prof2.w */
/*код перед вызовом функции из обрамления 2 из prof2.w */
returnVal = MPI_Send ( аргументы... );
/*код после вызова функции из обрамления 2 из prof2.w */
/*код после вызова функции из обрамления 1 из prof2.w */
/*код после вызова функции из обрамления 2 из prof1.w */
/*код после вызова функции из обрамления 1 из prof1.w */
return returnVal;
}
```

```
{{fn <function name escape> <function A> <function B> ... }}
...
{{callfn}}
...
{{endfnall}}
```

fn идентично fnall, за исключением генерации обрамлений только для точно названных функций. Например:

```
{{fn this_fn MPI_Send}}
{{vardecl int i}}
{{callfn}}
printf ( "Вызов {{this_fn}}.\n" );
printf ( "{{i}} не используется.\n" );
printf ( "Первый аргумент для {{this_fn}} это {{0}}\n" );
{{endfn}}
```

будет расширен до:

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

```
            int returnVal;
            int i;
            returnVal = PMPI_Send ( buf, count, datatype, dest, tag, comm );
            printf ( "Вызов MPI_Send.\n" );
            printf ( "i не используется.\n" );
            printf ( "Первый аргумент для MPI_Send это buf\n" );
            return returnVal;
        }
```

{{fn_num}}

Это номер, начинающийся с 0. Он наращивается при каждом использовании.

Простой файл обрамления находится в ‘sample.w’, а соответствующий вывод в ‘sample.out’.

B Опции mpirun

Опции mpirun можно увидеть при запуске mpirun -help (отметьте, что не все опции поддерживаются всеми устройствами). В зависимости от определенного устройства, вывод mpirun -help может различаться; следующая информация относится к устройству globus2.

```
   mpirun [опции mpirun ...]<имя программы> [опции...]
опции mpirun:

  -arch <архитектура>
           Определяет архитектуру (должен быть соответствующий
           файл machines.<arch> в /usr/local/mpich/bin/machines)
           если используется execer
  -h       Эта справка
  -machine <имя машины>
           Использовать стартовую процедуру для <имя машины>
           В настоящее время поддерживаются:
               paragon
               p4
               sp1
               ibmspx
               anlspx
               sgi_mp
               ipsc860
               inteldelta
               cray_t3d
               execer
               smp
               symm_ptx
  -machinefile <имя файла machine>
           Брать список возможных машин для запуска из файла <machine-filename>.
           Это список всех доступных машин; используйте -np <np> для
```

```
        указания определенного числа машин.
-np <np>
        Определить количество процессоров для запуска
-nodes <nodes>
        Определить количество узлов для запуска (для систем SMP,
        сейчас поддерживается только устройством ch_mpl)
-nolocal
        не запускать на локальной машине (работает только для задач
        ch_p4)
-all-cpus, -allcpus
        Использовать все доступные CPU на всех узлах.
-all-local
        Запускать все процессы на главном узле.
-exclude <list>
        Исключить узлы из списка, разделенного двоеточием.
-map <list>
        Использовать список, разделенный двоеточием для определения,
        какой ранг работает на каком узле.
-stdin filename
        Использовать filename как стандартный ввод программы. Это
        необходимо для программ, запускаемых как пакетные задачи, как на
        некоторых системах IBM SP и Intel Paragon с использованием NQS
        (см. ниже -paragontype).
        Используйте
            -stdin /dev/null
        если ввода нет и Вы предполагаете запустить программу в фоновом
        режиме. Альтернативой является перенаправление стандарного ввода
        из /dev/null, как в
            mpirun -np 4 a.out < /dev/null
-t      Тестирование - не запускать, а выводить, что должно было произойти
-v      Выводить подробные комментарии
-dbg    Опция `-dbg' сожет использоваться для выбора отладчика.
        Например, -dbg=gdb вызывает скрипт mpirun_dbg.gdb, находящийся в
        каталоге `mpich/bin'. Этот скрипт захватывает нужные аргументы вызывает
        отладчик gdb, и запускает первый процесс под управлением gdb, если это
        возможно. Существует 4 отладочных скрипта; gdb, xxgdb, ddd, totalview. Они
        могут потребовать редактирования, что зависит от Вашей системы. Существует
        еще скрипт для dbx, но он всегда должен редактироваться, поскольку команды
        для dbx варьируются от версии к версии. Вы можете также использовать эту
        опцию для вызова другого отладчика; например, -dbg=mydebug. Вам нужно
        написать скрипт `mpirun_dbg.mydebug', который следует формату
        встроенных скриптов отладки и поместить его в каталог mpich/bin.
-ksq    Хранить очереди передачи. Это полезно, если Вы предполагаете
        позже добавить totalview к запущенной (или блокированной) задаче, и хотите
        просмотреть очереди сообщений. (Обычно они не поддерживают
        просмотр отладчиком).
```

Опции для устройства globus2: За исключением -h, устройством globus поддерживаются только опции mpirun.

```
-machinefile <machine-file name>
        Взять список возможных машин для запуска из
        файла <machine-file name>
-np <np>
        Определить количество процессоров для запуска
-dumprsl
        Показывает строку RSL, используемую для отсылки задачи.
        Использование этой опции не запускает задачу.
-globusrsl <globus-rsl-file name>
        <globus-rsl-file name> содержит строку Globus RSL.
        При использовании этой опции все другие опции mpirun игнорируются.
```

Специальные опции пакетных сред:

```
-mvhome
         Перемещает исполняемые файлы в домашний каталог. Это необходимо,
         если все файловые системы не кросс-монтированы. Сейчас используется только anlspx.
-mvback files
         Перемещает указанные файлы обратно в текущий каталог. Нужно
         только при использовании -mvhome; в ином случае не имеет эффекта.
-maxtime min
         Максимальное время выполнения задачи в минутах. Сейчас используется только
         anlspx. Значение по умолчанию $max_time минут.
-nopoll
         Не использовать коммуникацию в режиме опроса. Доступны только для IBM SP.
```

Специальные опции для IBM SP2:

```
-cac name
          CAC для планировщика ANL. Сейчас используется только anlspx.
          Если не указано, используется любой приемлемый CAC.
```

При выходе mpirun возвращает состояние 0, если не обнаружена проблема, иначе возвращается ненулевое состояние.

При использовании устройства ch_p4 несколько архитектур может обрабатываться при указании нескольких аргументов -arch и -np. Например, для запуска программы на 2-х sun4 и 3-х rs6000, и если локальная машина - sun4, используйте

```
mpirun -arch sun4 -np 2 -arch rs6000 -np 3 program
```

Это предполагает, что программа работает для обеих архитектур. Если нужны различные исполняемые файлы, строка '%a' может заменяться именем архитектуры. Например, если программы - это program.sun4 и program.rs6000, то команда будет

```
mpirun -arch sun4 -np 2 -arch rs6000 -np 3 program.%a
```

Если вместо этого исполняемые файлы находятся в различных каталогах, например, '/tmp/me/sun4' и '/tmp/me/rs6000', команда будет

```
mpirun -arch sun4 -np 2 -arch rs6000 -np 3 /tmp/me/%a/program
```

Важно определить архитектуру через `-arch` *перед* определением числа процессоров. *Первая* команда `arch` должна относиться к процессору, на котором будет запущена задача.Если не определена опция `-nolocal`, то первая `-arch` должна относиться к процессору, с которого запускается `mpirun`.

C `mpirun` и Globus

В этом разделе мы описываем, как запускать программы MPI, используя устройство `globus2` в распределенной среде вычислений, ориентированной на Globus. Это предполагает, что (a) Globus установлен и на машинах, где Вы хотите запустить Ваше приложение MPI, запущены соответствующие демоны Globus; (b) Вы уже получили свой идентификатор Globus и сертификаты безопасности; (c) Ваш идентификатор Globus зарегистрирован на всех машинах; (d) у Вас действующий (с лицензией) прокси-сервер Globus. См. для более подробной информации `http://www.globus.org`.

Каждая команда `mpirun` для устройства `globus2` передает на сеть компьютеров, ориентированную на Globus, скрипт на языке определения ресурсов Globus, или *RSL скрипт*. Каждый скрипт RSL состоит из одной или нескольких подзадач RSL, обычно по одной на каждую машину в вычислениях. Вы можете создать Ваш собственный скрипт RSL[5] точно для `mpirun` (используя ощию `-globusrsl <rslfilename>`), в случае чего Вы не должны определять больше никаких опций `mpirun`, или Вы можете заставить `mpirun` создать скрипт RSL для Вас, основанный на аргументах, которые Вы передаете `mpirun`, и на содержимом Вашего файла `machines` (обсуждается ниже). В этом случае важно помнить, *что коммуникация между узлами в различных подзадачах всегда производится по TCP/IP, а более эффективный фирменный MPI поставщика используется только среди узлов одной и той же подзадачи.*

C.1 Использование `mpirun` для создания скрипта RSL для Вас

Вы можете использовать этот метод, если Вы хотите запустить *отдельный* исполняемый файл, который охватывает множество из нескольких двоично-совместимых машин, разделяющих единую файловую систему (т.е., они все имеют доступ к исполняемому файлу).

Использование `mpirun` для создания скрипта RSL для Вас требует наличия файла *machines*. Команда `mpirun` определяет, какой из файлов machines будет использоваться, следующим образом:

1. Если определен аргумент `-machinefile <имя файла machines>` для `mpirun`, именно он и используется, иначе
2. `mpirun` ищет файл 'machines' в каталоге, в котором Вы ввели `mpirun`, и наконец
3. он проверяет '/usr/local/mpich/bin/machines', где '/usr/local/mpich' является каталогом инсталляции `mpich`.

Если файл machines не найден ни в одном из этих мест, `mpirun` завершается с ошибкой.

Файл machines используется для указания компьютеров, на которых Вы можете пожелать запустить свое приложение. Компьютеры перечисляются указанием "сервиса" Globus на данной машине. Для большинства приложений используется сервис по умолчанию, требующий только указания полного доменного

[5] О синтаксисе и семантике языка определения ресурсов Globus см. `http://www.globus.org`

имени. Проконсультируйтесь с Вашим администратором Globus или веб-сайтом Globus `www.globus.org` для дополнительной информации относительно специальных сервисов Globus. Например, рассмотрим следующую пару двоично-совместимых машин {`m1,m2`}`.utech.edu`, имеющих доступ к единой файловой системе. Здесь приведен файл `machines`, использующий сервисы Globus по умолчанию.

```
“m1.utech.edu” 10
“m2.utech.edu” 5
```

Число, появляющееся на конце каждой строки необязательно (по умолчанию 1). Оно определяет максимальное число узлов, которые могут быть созданы в одной подзадаче RSL на каждой машине. `mpirun` использует определение `-np` для “обхода” файла `machines`. Например, используя файл, указанный выше, `mpirun -np 8` создает RSL с одной подзадачей с 8-ю узлами на `m1.utech.edu`, в то время, как `mpirun -np 12` создает две подзадачи, где первая имеет 10 узлов на `m1.utech.edu`, а вторая имеет 2 узла на `m2.utech.edu`, и наконец, `mpirun -np 17` создает три подзадачи с 10 узлами на `m1.utech.edu`, 5 узлами на `m2.utech.edu`, и заканчивая третьей подзадачей с 2 узлами снова на `m1.utech.edu`. Заметьте, что межподзадачные сообщения *всегда* передаются через TCP, даже если две отдельные подзадачи находятся на той же машине.

C.1.1 Использование `mpirun` для поддержки Вашего собственного скрипта RSL

Вы можете использовать `mpirun`, создав Ваш собственный скрипт RSL, если у Вас есть множество машин, которые не имеют доступа или не выполняют один и тот же исполняемый файл (т.е., машин, которые не совместимы на уровне двоичных файлов или не разделяют файловую систему). В этой ситуации, мы должны использовать нечто, называемое запросом на *языке определения ресурсов (RSL)* для определения имени исполняемого файла для каждой машины. Эта техника очень гибкая, но гораздо сложнее; сейчас идет работа над упрощением способа, которым организованы эти средства.

Простейший способ научиться писать свои собственные запросы RSL - изучить запросы, созданные для Вас `mpirun`. Рассмотрим пример, в котором мы хотим запустить приложение на кластере рабочих станций. Вспомним, что наш файл `machines` выглядит так:

```
“m1.utech.edu” 10
“m2.utech.edu” 5
```

Чтобы увидеть запрос RSL, сгенерированный в этом случае, *без* запуска программы, мы набираем следующую команду `mpirun`:

```
% mpirun -dumprsl -np 12 myapp 123 456
```

что приведет к выводу:

```
+
( &(resourceManagerContact=“m1.utech.edu”)
     (количество=10)
     (тип задачи=mpi)
     (метка=“subjob 0”)
     (среда=(GLOBUS_DUROC_SUBJOB_INDEX 0))
     (аргументы=“ 123 456”)
     (каталог=/homes/karonis/MPI/mpich.yukon/mpich/lib/IRIX64/globus)
     (исполняемый файл=/homes/karonis/MPI/mpich.yukon/mpich/lib/IRIX64/globus/myapp)
```

```
)
( &(resourceManagerContact="m2.utech.edu")
     (количество=2)
     (тип задачи=mpi)
     (метка="subjob 1")
     (среда= (GLOBUS_DUROC_SUBJOB_INDEX 1))
     (аргументы=" 123 456")
     (каталог=/homes/karonis/MPI/mpich.yukon/mpich/lib/IRIX64/globus)
     (исполняемый файл=/homes/karonis/MPI/mpich.yukon/mpich/lib/IRIX64/globus/myapp)
)
```

Отметьте, что (`тип задачи=mpi`) может встретиться только в тех подзадачах, машины которых имеют фирменные реализации MPI. Дополнительные переменные окружения могут быть добавлены так, как в примере ниже:

```
+
( &(resourceManagerContact="m1.utech.edu")
     (количество=10)
     (тип задачи=mpi)
     (метка="subjob 0")
     (среда=(GLOBUS_DUROC_SUBJOB_INDEX 0))
           (MY_ENV 246)
     (аргументы=" 123 456")
     (каталог=/homes/karonis/MPI/mpich.yukon/mpich/lib/IRIX64/globus)
     (исполняемый файл=/homes/karonis/MPI/mpich.yukon/mpich/lib/IRIX64/globus/myapp)
)
( &(resourceManagerContact="m2.utech.edu")
     (количество=2)
     (тип задачи=mpi)
     (метка="subjob 1")
     (среда= (GLOBUS_DUROC_SUBJOB_INDEX 1))
     (аргументы=" 123 456")
     (каталог=/homes/karonis/MPI/mpich.yukon/mpich/lib/IRIX64/globus)
     (исполняемый файл=/homes/karonis/MPI/mpich.yukon/mpich/lib/IRIX64/globus/myapp)
)
```

После редактирования Вашего собственного файла RSL Вы можете передать его непосредственно `mpirun` следующим образом:

```
% mpirun -globusrsl <ваш файл запроса RSL>
```

Отметьте, что при поддержке Вашего собственного файла RSL он должен быть единственным аргументом, определяемым для `mpirun`.

RSL является гибким языком, способным выполнять гораздо больше, чем представлено здесь. Например, он может использоваться для установки исполняемых файлов и переменных окружения на удаленных компьютерах перед началом выполнения. Полное описание языка можно найти на `http://www.globus.org`.

Благодарности

Выполнению работы, описанной в этом документе, способствовало много людей. Мы также благодарны тем, кто помогал в реализации `mpich`, особенно Patrick Bridges и Edward Karrels. Особенная благодарность Nathan Doss и Anthony Skjellum за ценную помощь в реализации и разработке `mpich`. Debbie Swider, работавшая в группе по `mpich` несколько лет помогала в поддержке и улучшении реализации `mpich`. David Ashton, разрабатывал версию `mpich` для Windows NT, поддержаваемую грантом корпорации Microsoft. Устройство `globus2` разработано Nick Karonis из Northern Illinois University и Brian Toonen из Argonne National Laboratory. Привязки к C++ были выполнены Andrew Lumsdaine и Jeff Squyres из Notre Dame. Параллельная система ввода-вывода ROMIO MPI-2 была разработана Rajeev Thakur из Argonne.

D Устаревшие возможности

В процессе разработки `mpich` были разработаны различные возможности для его инсталляции и использования. Некоторые из них были заменены затем новыми возможностями, описанными выше. Этот раздел хранит информацию об устаревших возможностях.

D.1 Более детальный контроль за компиляцией и компоновкой

Для большего контроля над процессом компиляции и компоновки `mpich` Вы можете использовать `'Makefile'`. Однако, чем изменять свой make-файл для каждой системы, Вам лучше использовать заготовки make-файлов и использовать команду `'mpireconfig'` для их преобразования в действительный make-файл. Чтобы сделать это, начните с файла `'Makefile.in'` в каталоге `'/usr/local/mpich/examples'`. Измените этот `'Makefile.in'` для Вашей программы и наберите

```
mpireconfig Makefile
```

(но не `mpireconfig Makefile.in`). Эта команда создаст `'Makefile'` из `'Makefile.in'`. Затем введите:

```
make
```

Список литературы

[1] TotalView Multiprocess Debugger/Analyzer, 2000. `http://www.etnus.com/Products/TotalView`.

[2] Ralph Butler and Ewing Lusk. User's guide to the p4 parallel programming system. Technical Report ANL-92/17, Argonne National Laboratory, Argonne, IL, 1992.

[3] James Cownie and William Gropp. A standard interface for debugger access to message queue information in MPI. In Jack Dongarra, Emilio Luque, and Tomas Margalef, editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, volume 1697 of *Lecture Notes in Computer Science*, pages 51-58. Springer Verlag, 1999. 6th European PVM/MPI Users' Group Meeting, Barcelona, Spain, September 1999.

[4] Message Passing Interface Forum. MPI: A message-passing interface standard. Computer Science Dept. Technical Report CS-94-230, University of Tennessee, Knoxville, TN, 1994.

[5] William Gropp, Steven Huss-Lederman, Andrew Lumsdaine, Ewing Lusk, Bill Nitzberg, William Saphir, and Marc Snir. *MPI-The Complete Reference: Volume 2, The MPI-2 Extensions*. MIT Press, Cambridge, MA, 1998.

[6] William Gropp and Ewing Lusk. Installation guide for `mpich`, a portable implementation of MPI. Technical Report ANL-96/5, Argonne National Laboratory, 1996.

[7] William Gropp and Ewing Lusk. A high-performance MPI implementation on a shared-memory vector supercomputer. *Parallel Computing*, 22 (11):1513-1526, January 1997.

[8] William Gropp and Ewing Lusk. Sowing `mpich`: A case study in the dissemination of a portable environment for parallel scienti computing. *IJSA*, 11 (2):103-114, Summer 1997.

[9] William Gropp, Ewing Lusk, Nathan Doss, and Anthony Skjellum. A high-performance, portable implementation of the MPI Message-Passing Interface standard. *Parallel Computing*, 22 (6):789-828, 1996.

[10] William Gropp, Ewing Lusk, and Anthony Skjellum. *Using MPI: Portable Parallel Programming with the Message Passing Interface*, 2nd edition. MIT Press, Cambridge, MA, 1999.

[11] William Gropp, Ewing Lusk, and Rajeev Thakur. *Using MPI-2: Advanced Features of the Message-Passing Interface*. MIT Press, Cambridge, MA, 1999.

[12] M. T. Heath. Recent developments and case studies in performance visualization using ParaGraph. In G. Haring and G. Kotsis, editors, *Performance Measurement and Visualization of Parallel Systems*, pages 175-200. Elsevier Science Publishers, 1993.

[13] Virginia Herrarte and Ewing Lusk. Studying parallel program behavior with upshot. Technical Report ANL-91/15, Argonne National Laboratory, 1991.

[14] Message Passing Interface Forum. MPI: A Message-Passing Interface standard. *International Journal of Supercomputer Applications*, 8 (3/4):165-414, 1994.

[15] Peter S. Pacheco. *Parallel Programming with MPI*. Morgan Kaufman, 1997.

[16] Marc Snir, Steve W. Otto, Steven Huss-Lederman, David W. Walker, and Jack Dongarra. *MPI-The Complete Reference: Volume 1, The MPI Core*, 2nd edition. MIT Press, Cambridge, MA, 1998.

[17] Omer Zaki, Ewing Lusk, William Gropp, and Deborah Swider. Toward scalable performance visualization with Jumpshot. *High Performance Computing Applications*, 13 (2):277-288, Fall 1999.