Институт теоретической и прикладной механики
им. С.А. Христиановича СО РАН

Инструкция пользователя вычислительного кластера Aero

Новосибирск 2020

Содержание

1 Общие сведения

1.1 Оснащение кластера

Кластер состоит из 1 управляющего и 6 вычислительных узлов

Каждый **вычислительный узел** оснащен следующим оборудованием:

- 2 20-ядерных процессора Intel(R) Xeon(R) Gold 6230 CPU @ 2.10GHz
- 2 графических процессора (GPU) NVidia Tesla V100
- 256 ГБ оперативной памяти DDR4-2933
- Коммуникационная сеть – FDR Infiniband (56 Гбит/с)

Суммарная пиковая (теоретическая) производительность вычислительных узлов – 84 Tflops

Управляющий узел (Узел доступа и управления задачами) – многопроцессорный сервер, который позволяет пользователю получить доступ к ресурсам кластера. На данном узле осуществляется подготовка заданий для кластера, работа с входными данными и результатами вычислений. Проведение расчетов на этом узле не предполагается.

1.2 Программное обеспечение

На кластере установлена операционная система CentOS Linux.

В системе установлены компиляторы Fortran, C/C++ из набора GCC-4.8.5. Кроме того, пользователь может подключить компиляторы Fortran и C/C++ версии 7.3.1 или 8.3.1. Для этого нужно дать соответствующую команду в командной строке:

```
scl enable devtoolset-7 bash
```

или

```
scl enable devtoolset-8 bash
```

Выбранные версии компиляторов будут действовать до перезапуска сессии.

Установлены библиотеки CUDA 10.1, OpenMPI-4.0.2.

Отладчик – gdb/cgdb.

В качестве консольных текстовых редакторов можно использовать `nano`, `vim`. Редактор с поддержкой GUI – geany.

Консольный файловый менеджер – mc.

Запуск задач на счет осуществляется под управлением системы очередей Slurm.

2 Работа пользователя с кластером

Цикл работы пользователя с кластером состоит из следующих этапов:

- Удаленный вход на кластер.
- Копирование данных между кластером и компьютером пользователя.
- Редактирование исходных текстов программ.
- Компиляция программ.
- Запуск задач и работа с очередью.
- Завершение сеанса.

Рассмотрим подробно основные этапы работы пользователя с кластером.

2.1 Удаленный вход на кластер

Физически кластер располагается в специально выделенном помещении, непосредственный доступ в которое ограничен. Пользователи получают доступ к ресурсам кластера посредством удаленного входа на узел доступа. Предварительно каждый пользователь должен пройти процедуру регистрации (получить логин и пароль для входа).

Доступ на кластер возможен по протоколу SSH.

В состав кластера включен узел доступа:

IP-адрес узла доступа: **192.168.64.100**, имя: **aero.lan**

Процедура входа на узел доступа зависит от операционной системы компьютера пользователя. Ниже описаны два основных варианта.

2.1.1 Для пользователей Linux

Для доступа к кластеру наберите в командной строке:

```
ssh username@aero.lan
```

где username – ваш логин, aero.lan – имя узла доступа. На соответствующий запрос системы введите пароль, полученный при регистрации. При первом входе на кластер система потребует сменить пароль.

Пример:

Для входа на ВК пользователь **testuser** должен набрать:

```
ssh testuser@aero.lan
```

2.1.2 Для пользователей Windows

Для удаленного подключения из Windows необходимо использовать специальную программу – SSH-клиент, например, PuTTy.

Для установки PuTTY скачайте на свой компьютер файл putty.exe и запустите его. В появившемся окне «Session» в строке «Host Name (or IP address)» укажите имя (или IP-адрес) узла доступа, выберите протокол SSH и начните сессию. В открывшемся окне с приглашением «login as:» наберите имя пользователя, нажмите Enter и далее в ответ на запрос системы введите пароль, полученный при регистрации.

Если аутентификация прошла успешно, на экране появится приглашение командной строки, например:

```
testuser@headnode ~ $
```

Это означает, что вы подключились к кластеру и находитесь в своем домашнем каталоге **/mnt/data/home/<username>** на головном (управляющем) узле кластера (headnode).

Примечание: При первой попытке входа на кластер операционная система вашего компьютера выдаст предупреждение о том, что ей неизвестен хост с адресом **192.168.64.100**, и спросит, согласны ли вы продолжить подключение.

Для подключения наберите слово **Yes** и нажмите **Enter** (при использовании PuTTy – нажмите кнопку **Да**), хост **192.168.64.100** будет добавлен к списку известных хостов:

```
Warning: Permanently added '192.168.64.100' (RSA) to the list
of known hosts.
```

2.2 Копирование файлов между кластером и компьютером пользователя

Работая в режиме удаленного доступа, пользователь не имеете возможности выполнять прямое копирование файлов на кластер с внешних носителей и в обратном направлении. Для передачи файлов необходимо использовать команды **scp** или **sftp** при подключении с Unix-системы, либо специальные утилиты, предназначенные для удаленного копирования файлов (например, WinSCP), при подключении с компьютера на базе Windows.

Рассмотрим действия, необходимые для копирования файлов между двумя компьютерами с использованием протокола SCP.

2.2.1 Для пользователей Linux

Упрощенный формат команды **scp** или **rsync**:

```
scp [[user@]host1:]file1 ... [[user@]host2:]file2
```

Команда **scp** выполняет копирование файлов между двумя компьютерами сети. Первый параметр – копируемый файл на компьютере host1, второй параметр – путь его размещения на компьютере host2. Если для копирования требуется подключение к удаленному компьютеру, необходимо указать логин и пароль пользователя, имеющего к нему доступ.

Примечание: При подключении к удаленному хосту команда **scp** использует протокол SSH, что требует от пользователя указания пароля для удаленного доступа.

Примеры:

Для копирования файла **myprog.cpp** из домашнего каталога **~/data/** на кластере в каталог **/home/user/test** рабочего компьютера пользователю **testuser** потребуется набрать команду:

```
scp testuser@aero.lan:~/data/myprog.cpp /home/user/test/
```

В данном примере не используются необязательные части второго параметра, так как предполагается, что пользователь **testuser** выполняет копирование, находясь за своим рабочим компьютером (нет необходимости указывать его сетевой адрес и логин для подключения).

Для копирования файла **myprog.cpp** в обратном направлении (с рабочего компьютера в каталог ~/data/ на кластере) пользователь **testuser** должен набрать команду:

```
scp /home/user/test/myprog.cpp testuser@aero.lan:~/data/
```

Для копирования директории вместе со всем ее содержимым необходимо использовать параметр **-r**:

```
scp -r /home/user/test/myprog.cpp testuser@aero.lan:~/data/
```

2.2.2 Для пользователей Windows

Рассмотрим процедуру копирования файлов на примере программы WinSCP. После установки и запуска программа WinSCP запросит у вас (по аналогии с PuTTY) данные для подключения: символьное имя (IP-адрес) узла доступа, логин и пароль. После успешной аутентификации откроется окно, подобное Windows Explorer, либо двухпанельное окно в стиле Norton Commander (в зависимости от варианта, выбранного при установке программы). После этого вы получите

возможность копировать файлы между рабочим компьютером и кластером привычным для вас способом.

Кроме копирования программа WinSCP позволяет выполнять другие операции с файлами: редактирование, переименование, удаление, изменение прав доступа и прочие.

2.3 Редактирование файлов на кластере

Процесс разработки программного обеспечения содержит этап редактирования файлов с исходными данными и текстами программ. Можно вносить необходимые изменения в файлах на своем рабочем компьютере, а затем копировать их на кластер, однако в ряде случаев удобнее выполнять редактирование прямо на кластере. Для этого можно воспользоваться любым текстовым редактором. Одним из наиболее простых в освоении является редактор `mcedit`, встроенный в оболочку Midnight Commander.

Midnight Commander – файловый менеджер с текстовым интерфейсом, функционирующий в соответствии с теми же принципами, что и Norton Commander, Far Manager, Total Commander.

Midnight Commander поддерживает все основные операции с файлами и каталогами: создание, удаление, копирование, перемещение, изменение прав доступа и другие. Для запуска Midnight Commander используется команда `mc`.

Экран Midnight Commander состоит из двух панелей, в которых отображаются списки каталогов и файлов. Панель, в которой находится курсор, является активной. Выбор активной панели осуществляется с помощью клавиши **Tab**.

В нижней части экрана расположена командная строка и панель горячих клавиш **F1-F10:**

- F1 – помощь;
- F2 –меню пользователя;
- F3 – просмотр файла (каталога), на который указывает курсор;
- F4 – редактирование выбранного файла с помощью встроенного редактора **mcedit**;
- F5 – копирование выбранного файла (каталога);
- F6 – переименование/перемещение выбранного файла (каталога);
- F7 – создание каталога;
- F8 – удаление выбранного файла (каталога);
- F9 – вызов верхнего меню;

- F10 – выход из Midnight Commander.

Для выполнения групповых операций с несколькими файлами и каталогами активной панели предварительно отметьте их с помощью клавиши **Ins**. Для создания файла используйте сочетание клавиш **Shift+F4** (вызов редактора **mcedit**). Для сохранения изменений нажмите клавишу **F2**. Для выхода из редактора используйте клавишу **F10** или **Esc**.

2.4 Компиляция программ

Для гибкого использования различных версий компиляторов и библиотек используется система модулей. Каждая версия компилятора или подключаемой библиотеки выделена в свой модуль, и при загрузке модуля производится перенастройка путей поиска запускаемых файлов и путей к библиотекам. Для компиляции программ с использованием CUDA или MPI необходимо подключить соответствующие модули. Список доступных модулей можно увидеть, набрав команду

```
module avail
```

Появится список доступных модулей. Загрузить необходимый модуль можно командой **module add**. или **module load** Например, для загрузки модулей cuda/10.1 и openmpi-4.0.2 нужно дать команду

```
module add cuda/10.1 openmpi-4.0.2
```

Просмотреть список загруженным модулей можно командой

```
module list
```

Для компиляции программы на языке C++ с использованием MPI нужно набрать команду

```
mpicxx program.cc -o program
```

Для программы на языке Fortran:

```
mpifort program.f -o program
```

Для программ с использованием CUDA:

```
nvcc program.cu -o program
```

Для сборки больших программ, состоящих из многих файлов, на кластере доступны средства сборки **make** и **cmake**.

2.5 Запуск задач и работа с очередью

Для запуска задач (прикладных программ) на счет и последующего контроля за их выполнением на кластере используется система управления заданиями Slurm – свободно распространяемая версия системы пакетной обработки. Эта система автоматически распределяет задания пользователей по свободным вычислительным ресурсам кластера, выполняет планирование очередей заданий, осуществляет контроль за исполнением заданий. Для запуска задачи необходимо заказать ресурсы, и указать какие исполняемы файлы необходимо выполнить, когда заказанные ресурсы станут доступными (освободятся после окончания ранее запущенных вычислений).

На кластере организовано 3 очереди: **cpu**, **gpu** и **hybrid**.

- Очередь **cpu** предназначена для выполнения расчетов с использованием только центральных процессоров. Максимальное количество ресурсов: 6 узлов по 38 процессов на каждом узле.
- Очередь **gpu** – для использования графических процессоров. Максимальное количество ресурсов: 6 узлов по 2 ГПУ и 2 ЦПУ на каждом узле.
- Очередь **hybrid** – для запуска задач, использующих как ЦПУ, так и ГПУ одновременно. Максимальное количество ресурсов: 6 узлов по 2 ГПУ и 40 ЦПУ на каждом узле.

ПРЕДУПРЕЖДЕНИЕ!

*Если задача запущена в неправильной очереди, она зарезервирует ресурсы, но не будет их использовать и не позволит другим пользователям сделать это. Например, задача, использующая ГПУ, но запущенная в очереди **cpu**, блокирует 20 процессов, которые могли бы быть использованы другими пользователями. В случае поступления жалобы на подобную ситуацию, эта задача будет снята со счета принудительно без предупреждения.*

Внимание! Следите, чтобы заказанные ресурсы не превышали реальные возможности кластера, так как такая задача никогда не запустится. Например, бесполезно заказывать 7 вычислительных узлов, так как их физически всего 6. Задача поставится в очередь, но на счет не попадет.

Справочные man-руководства существуют для всех сервисов SLURM, команд, и API функций. Параметр `--help` выдает краткий перечень возможных параметров команды.

Основные команды Slurm приведены ниже.

`sbatch` — используется, чтобы запустить скрипт пакетной обработки для дальнейшего выполнения. Скрипт обычно содержит одну или больше srun-команд для запуска параллельных задач.

`scancel` — используется, чтобы отменить случайно запущенные или зависшие задачи и подзадачи. Она может также быть использована, чтобы отправить управляющий сигнал всем процессам, связанным с выполняющейся задачей или подзадачей.

`scontrol` — инструмент администрирования для просмотра и/или изменения состояния SLURM. Отметим, что многие из управляющих команд могут выполняются только суперпользователем.

`sinfo` — сообщает состояние разделов и узлов, управляемых SLURM. Она имеет много способов фильтрации, сортировки, и форматирования выдаваемых результатов.

`squeue` — докладывает о состоянии задач и подзадач. Она имеет много способов фильтрации, сортировки, и форматирования выдаваемых результатов. По умолчанию, она в первую очередь сообщает о запущенных задачах, зависшие задачи идут следующими по ранжированию.

`srun` — используется вместо более знакомого многим mpirun, чтобы запустить задачу или подзадачу в реальном времени. `srun` имеет много различных параметров, чтобы конкретизировать требования к ресурсам, в том числе: минимальное и максимальное количество узлов, количество процессоров, указать какие узлы использовать или не использовать, характеристики узлов (сколько памяти, дискового пространства, определенные требуемые особенности, и т.п.). Задача может содержать множество job шагов, выполняющихся последовательно или в параллельно на независимых или общих узлах в пределах выделенных задаче узлов.

2.6 Примеры

Для начала мы определяем, какие разделы существуют в системе, какие узлы они включают, и какое состояние системы в целом. Эта информация предоставляется командой `sinfo`.

```
$ sinfo
```

```
PARTITION   AVAIL       TIMELIMIT   NODES       STATE       NODELIST
cpu         up          14-00:00:0  6           mix         node[01-06]
gpu         up          14-00:00:0  6           mix         node[01-06]
hybrid      up          14-00:00:0  6           mix         node[01-06]
```

Видим, что есть три раздела (очереди): **cpu**, **gpu** и **hybrid**. Мы видим, что все разделы находятся в состоянии **up** (доступна). Статус **mix** обозначает, что ресурсы этого раздела частично заняты.

В каждой строке приводится информация о максимальном времени работы задачи в разделе, количестве узлов, их состоянии и список узлов.

Запуск задачи на счет осуществляется с помощью скрипта с описанием требуемых параметров и указанием, какую именно задачу необходимо запустить. Рассмотрим пример, когда требуется запустить задачу, совмещающую MPI и OpenMP параллелизацию.

Предположим, требуется запустить задачу на 2 узлах, на каждом узле запустить 2 MPI-потока, причем каждый поток может разделяться на 20 OpenMP-нитей.

Пример скрипта

```
#!/bin/bash                   Интерпретатор скрипта
#SBATCH --partition=cpu       Используемая очередь
#SBATCH --job-name=Test       Имя задачи
#SBATCH --nodes=2             Требуемое количество вычислительных узлов
#SBATCH --ntasks-per-node=2   Количество MPI-процессов на каждом узле
#SBATCH --cpus-per-task=20    Количество OpenMP-процессов на каждый MPI-поток
#SBATCH --time=10:00          Максимальное время счета в формате HH:MM:SS. Если
                              задача не закончится раньше, по истечении этого
                              времени она будет снята системой. Если этот
                              параметр не задан, задаче задается время по
                              умолчанию – 4 часа.
#SBATCH --output=stdout.txt   Стандартный вывод перенаправить в файл stdout.txt
#SBATCH --error=stderr.txt    Вывод ошибок перенаправить в файл stderr.txt
#SBATCH --export=ALL          Передать переменные среды вычислительным узлам.

module load openmpi-4.0.2     Загрузить модули (если стоит опция #SBATCH —
                              export=ALL, эту module можно не вызывать)
# Run the parallel MPI executable
srun ./program                Отправить программу program на исполнение
```

Сохраним содержимое этого скрипта в файле **sbatch.cmd**

Запуск задачи осуществляется командой

```
sbatch sbatch.cmd
```

Запущена ли задача и в каком состоянии она находится, можно увидеть командой squeue. Поля показывают ID задачи, очередь, название, пользователя, состояние, время работы, количество и список узлов. Для более детальной информации используйте команду *man slurm*.

```
$ squeue
```

```
JOBID   PARTITION   NAME      USER     ST    TIME      NODES   NODELIST(REASON)
1075    gpu         ns_shock  vashen   PD    0:00      6       (Resources)
1074    cpu         Slider    vashen   R     2:25:53   6        node[01-06]
```

Видим, что в очереди стоят две задачи с идентификаторами 1074 и 1075. Одна из них запущена в очереди **cpu**, другая – в очереди **gpu**. Статус задач показывает состояние: «R» (Running) – расчет выполняется, «PD» (Pending) – задача ожидает старта. Последнее поле показывает, на каких узлах задача считается или причину простоя. В данном случае задача **1074** считается на узлах **node[01-06]**, а задача **1075** стоит в очереди из-за нехватки ресурсов.

Снять задачу со счета или из очереди можно командой scancel:

```
$ scancel 1075
```

По этой команде задача с идентификатором 1075 снимется со счета.

3 Основы работы в Linux

3.1 Операции с файлами и каталогами

Файловая система – это средство организации, хранения и именования данных в виде файлов. Файловая система представляет собой древовидную структуру, начинающуюся от корневого каталога. В каждом каталоге могут находиться другие каталоги и файлы. Местоположение файла задается в виде последовательности вложенных каталогов (пути) к файлу. Путь может быть относительным и абсолютным. Абсолютный путь к файлу начинается от корневого каталога, обозначаемого символом «/», относительный путь – от текущего каталога. Например, абсолютный путь к файлу `test.cpp`, находящемуся в каталоге `/home/testuser/bin`, будет выглядеть следующим образом:

```
/home/tester/bin/test.cpp
```

Относительный путь из каталога `/home/testuser` будет таким:

```
../bin/test.cpp
```

Примечание: Символом **«..»** (две точки) обозначается предыдущий по отношению к текущему каталог. Если требуется явно указать системе, что она должна использовать файл из текущего каталога, то можно воспользоваться символом **«.»** (одна точка). Например, запись `./test.cpp` эквивалентна указанию абсолютного пути к файлу `test.cpp`, находящемуся в текущем каталоге.

3.1.1 Определение текущего каталога

Для определения текущего каталога командной строки используется команда pwd.

3.1.2 Смена каталога

Для перехода в произвольный каталог используется команда cd. Например, команда

```
cd /home/tester/
```

сделает каталог `/home/tester` текущим каталогом командной строки. Команда «`cd /`» сменит каталог на корневой.

Каждому пользователю соответствует специальный каталог, называемый домашним каталогом. Этот каталог расположен в директории **/home**, и его название совпадает с именем пользователя. Пользователь располагает полными

правами доступа к домашнему каталогу и его подкаталогам: может создавать файлы и каталоги, редактировать и удалять их, переименовывать, перемещать, а также запускать программы на выполнение.

Примечание: Для обозначения домашнего каталога используется символ **«~»** («тильда»). Т.е. команду `cd /home/testuser/bin` пользователь `testuser` может заменить командой

```
cd ~/bin
```

3.1.3 Список объектов в каталоге

Для просмотра списка объектов в каталоге используется команда `ls`. Команда, вызванная без параметров, выведет список файлов и каталогов в текущем каталоге. Команда

```
ls /home/testuser
```

выдаст на экран содержимое каталоге /home/`testuser`. Команда `ls` имеет множество параметров. Например, вызов команды с ключом `-l` позволит получить полную информацию об объектах. Вызов ls -lrth выдаст список файлов, отсортированных по времени в обратном порядке с указанием длины файлов в удобочитаемом виде (кБ, МБ).

3.1.4 Создание и удаление каталога

Для создания каталога используется команда

```
mkdir <name>
```

Если указано только имя каталога, он будет создан в текущем каталоге. Можно указать как абсолютный, так и относительный путь к создаваемому каталогу. Например, находясь в корневом каталоге, пользователь `testuser` может создать каталог `data` в своем домашнем каталоге, выполнив команду:

```
mkdir /home/testuser/data
```

Для удаления каталога используется команда:

```
rmdir <name>
```

Эта команда удаляет только пустые каталоги, поэтому перед удалением любого каталога необходимо очистить его содержимое. Для рекурсивного удаления каталога и всего содержимого используется команда:

```
rm –rf <name>
```

3.1.5 Создание файлов

Текстовый файл можно создать в любом текстовом редакторе, указав имя создаваемого файла при запуске редактора. Для создания файла в Midnight Commander нажмите **Shift+F4**. Другой способ создания файла – скопировать уже существующий файл.

Для создания нового пустого файла можно использовать команду `touch`. Например,

```
touch ~/test
```

создаст пустой файл `test` в домашнем каталоге.

3.1.6 Копирование, перемещение и удаление файлов и каталогов

Команда `cp` позволяет скопировать файл или каталог. Формат команды следующий:

```
cp <source> <destination>
```

cp копирует файлы (или, если попросить, каталоги). Вы можете либо копировать один файл в другой, заданный файл, либо копировать сколько угодно файлов в заданный каталог. Если последний аргумент является существующим каталогом, то cp копирует каждый исходный файл в этот каталог (сохраняя имена). В противном случае, если задано только два файла, то cp копирует первый файл во второй.

Для копирования каталога к команде `cp` добавляется ключ `-r`**:**

```
cp -r <source_dir>  <detination_dir>
```

Команда `mv` перемещает либо переименовывает файл или каталог. Исходные объекты либо получают новое имя, либо перемещаются в новое место.

```
mv <source> <destination>
```

Для удаления файлов используется команда `rm`.

```
rm <name>
```

Для удаления каталогов добавляется опция `-r`:

```
rm -r <dir>
```

3.1.7 Вывод текстового файла на экран

Команда **less** выводит содержимое текстового файла на экран. Эта команда удобна тем, что позволяет просматривать содержимое файла постранично, умеет искать данные в файле.

```
less test.cpp
```

Используйте для перемещения по тексту следующие клавиши:

f либо **пробел** – следующая страница; **b** либо **Esc-v** – предыдущая страница; **/шаблон** – поиск шаблона вперед по тексту; **n** – повторить поиск вперед; **? шаблон** – поиск шаблона назад по тексту; **N** – повторить поиск назад; **q** – выход.

3.1.8 Перенаправление вывода

Если результат работы программы или команды не помещается полностью на экране, существует несколько способов, позволяющих прочитать вывод полностью. Первый способ – нажать клавишу **Shift** и клавиши **PageUp** или **PageDown**.

Второй способ – перенаправление результатов работы программы (команды) в текстовый файл. Для этого используется специальный символ перенаправления вывода **«>»** («больше»). Так, после выполнения команды

```
cmd1 > file
```

в текущем каталоге появится файл с именем `file`, который будет содержать весь текстовый вывод программы `cmd1`. Если такой файл существовал, он будет перезаписан заново. Если указать два символа перенаправления вывода подряд:

```
cmd1 >> file
```

то весь вывод будет дописан к концу файла.

Если содержимое вывода команды нужно не сохранять в файл, а просто просмотреть, можно использовать конвейеры. Конвейеры — это возможность нескольких программ работать совместно, когда выход одной программы непосредственно идет на вход другой без использования промежуточных временных файлов. Синтаксис: `команда1 | команда2`, выполняет `команду1` используя её поток вывода как поток ввода при выполнении `команды2`, что равносильно использованию двух перенаправлений и временного файла:

```
команда1 > ВременныйФайл
команда2 < ВременныйФайл
rm ВременныйФайл
```

Для просмотра вывода команды можно использовать cmd | less.

Возможный пример использования конвейеров – поиск фразы в большом количестве файлов. Например, нужно найти все вызовы команд MPI во всех файлах, приведенных в разделе примеров. Эти файлы у каждого пользователя находятся в директории $HOME/Examples/ . Для этого можно составить конвейер

find ~/Examples/ -type f | xargs grep MPI_

Алгоритм работы этого конвейера следующий:

find ~/Examples/ -type f – находит все файлы в директории ~/Examples. Этот список нужно отдать в качестве аргумента команде grep. Для этого используется команда xargs. Эта команда читает данные, приходящие на стандартный вход и передает, их следующей команде как аргументы.

grep MPI_ – ищет все вхождения слов «MPI_» в полученном списке файлов

В результате получим следующий список:

```
/mnt/data/home/vashen/Examples/MPI/mpiProg.cc: MPI_Init(&argc, &argv);
/mnt/data/home/vashen/Examples/MPI/mpiProg.cc: MPI_Comm_size(MPI_COMM_WORLD, &size);
/mnt/data/home/vashen/Examples/MPI/mpiProg.cc: MPI_Comm_rank(MPI_COMM_WORLD, &rank);
/mnt/data/home/vashen/Examples/MPI/mpiProg.cc: MPI_Finalize();
/mnt/data/home/vashen/Examples/MPI/sbatch.cmd:#SBATCH --job-name=MPI_test
Двоичный файл /mnt/data/home/vashen/Examples/MPI/mpiProg совпадает
/mnt/data/home/vashen/Examples/MPI+OpenMP/mpi_omp_Prog.cc: MPI_Init(&argc, &argv);
/mnt/data/home/vashen/Examples/MPI+OpenMP/mpi_omp_Prog.cc:  MPI_Comm_size(MPI_COMM_WORLD,
&size);
/mnt/data/home/vashen/Examples/MPI+OpenMP/mpi_omp_Prog.cc:  MPI_Comm_rank(MPI_COMM_WORLD,
&rank);
/mnt/data/home/vashen/Examples/MPI+OpenMP/mpi_omp_Prog.cc: MPI_Finalize();
/mnt/data/home/vashen/Examples/MPI+OpenMP/sbatch.cmd:#SBATCH --job-name=MPI_OMP_test
```

3.1.9 Работа с системой документации

Основной способ получения информации в Linux – чтение, так называемых, **man pages** (manual pages – страницы руководства). Большинство объектов Linux, начиная с основных понятий, заканчивая командами операционной системы, описаны в таком руководстве. Для того чтобы посмотреть справку по той или иной команде, наберите в командной строке

```
man <object>
```

Команда

```
man man
```

выведет на экран руководство по использованию команды `man`.

Для перелистывания страниц используются клавиши управления курсором **PgUp**, **PgDn**, **Home**, **End**, для завершения просмотра – **q**.

Кроме того, многие программы могут запускаться в режиме выдачи помощи. Для этого надо запустить их с ключом `--help` (два знака «минус» `help`) . В результате на экран будет выдана краткая справка по использованию программы. Например, частичный результат выполнения команды `man --help`:

```
 usage: man [-c|-f|-k|-w|-tZT device] [-adlhu7V] [-Mpath] [-
Ppager] [Slist] [-msystem] [-pstring] [-Llocale] [-eextension]
[section] page … -a, --all find all matching manual pages. -d,
--debug emit debugging messages.
```

…

```
-h, --help show this usage message.
```

Здесь видно, что команда `man` имеет множество параметров, большинство из которых необязательны, кроме параметра `page`.

Примечание: В большинстве случаев помощь выводится на английском языке. Для получения русскоязычной справки воспользуйтесь интернетом.

3.1.10 Права доступа к файлам и каталогам

Каждый пользователь Linux входит в одну основную и, возможно, некоторые дополнительные группы. Узнать подробную информацию о пользователе и его членстве в группах можно с помощью команды `id`.

Любой файл или каталог в Linux имеет пользователя-владельца и группу-владельца. Права доступа к файлам и каталогам определяются для трех категорий: пользователя-владельца, группы-владельца и прочих. Права могут быть следующие: чтение (**r**), запись (**w**), выполнение (**x**), и отсутствие соответствующих прав (**-**). Таким образом, права на конкретный файл обычно выглядят так:

```
rwxr-xr-x
```

Первые 3 символа – права владельца, следующие 3 символа – права группы, последние 3 символа – права остальных пользователей. В данном примере видно, что владелец может читать, изменять и запускать файл на выполнение, а все прочие не могут изменять файл, но могут читать и выполнять его.

Для изменения прав доступа нужно дать команду chmod. Например, команда

```
chmod a+r
```

добавит (**+**) всем (**a**) права на чтение (**r**). Команда

```
chmod o-rw
```

запретит (**-**) всем остальным (**o**), кроме владельца и группы читать (**r**) и писать (**w**) в файл.

3.1.11 Основные команды Linux

Еще раз перечислим команды, речь о которых шла выше.

`mc`	запуск файлового менеджера Midnight Commander
`man <command>`	руководство по использованию `<command>`
`pwd`	получение имени текущего каталога
`cd <dir>`	переход в каталог `dir`
`ls <dir>`	получение списка объектов в каталоге `dir`
`mkdir <dir>`	создание каталога `dir`
`rm <file>`	удаление файла `file`
`rm -r <dir>`	удаление каталога `dir` с подкаталогами
`cp <source> <destination>`	копирование файла (каталога)
`mv <source> <destination>`	перемещение либо переименование файла (каталога)
`passwd`	смена пароля
`id`	получение информации о пользователе
`ssh <user>@<host>`	удаленный доступ пользователя `user` на машину с адресом `host`
`exit`	завершение сеанса

`scp <user1>@<host1>:<file1> <user2>@<host2>:<file2>`	удаленное копирование файлов
`rsync <user1>@<host1>:<file1> <user2>@<host2>:<file2>`	Удаленное копирование файлов с докачкой
`find`	поиск файлов в заданной директории
`grep`	поиск шаблона в файле
`locate`	поиск файла в системе по части его имени
`meld`	Средство для сравнения нескольких файлов или директорий с графической оболочкой
`diff`	Консольное средство сравнения файлов
`rsync <user1>@<host1>:<file1> <user2>@<host2>:<file2>`	Удаленное копирование файлов с докачкой
`cat`	выдача содержимого файла на экран
`less`	выдача содержимого файла на экран с прокруткой, поиском и пр.

4 Быстрый старт

Доменное имя кластера: **aero.lan**

IP-адрес: **192.168.64.100**

Для компиляции программ с использованием MPI нужно подключить модуль openmpi-4.0.2:

```
module load openmpi-4.0.2
```

Для программ с использованием CUDA:

```
module load cuda
```

После этого станет доступна компиляция программ на языках C/C++, Fortran, или CUDA. Компиляция осуществляется командами:

C/C++:

```
mpicxx program.cc -o program
```

Fortran:

```
mpifort program.f -o program
```

CUDA:

```
nvcc program.cu -o program
```

Далее программа отправляется в систему очередей для запуска:

```
sbatch sbatch.cmd
```

где sbatch.cmd – файл задания. Примеры файлов заданий для различных типов задач приведены ниже.

Проверка статуса очереди выполняется командой squeue:

```
squeue
```

```
JOBID   PARTITION   NAME      USER     ST    TIME      NODES   NODELIST(REASON)
1075    gpu         ns_shock  vashen   PD    0:00      6       (Resources)
1074    cpu         Slider    vashen   R     2:25:53   6       node[01-06]
```

Видим, что в очереди стоят две задачи с идентификаторами 1074 и 1075. Одна из них запущена в очереди **cpu**, другая – в очереди **gpu**. Статус задач показывает состояние: «R» (Running) – расчет выполняется, «PD» (Pending) – задача ожидает

старта. Последнее поле показывает, на каких узлах задача считается или причину простоя. В данном случае задача **1074** считается на узлах **node[01-06]**, а задача **1075** стоит в очереди из-за нехватки ресурсов.

Снять задачу со счета или из очереди можно командой scancel:

```
$ scancel 1075
```

Статус системы очередей и доступных ресурсов показывается командой sinfo

```
sinfo
```

PARTITION	AVAIL	TIMELIMIT	NODES	STATE	NODELIST
cpu	up	14-00:00:0	6	mix	node[01-06]
gpu	up	14-00:00:0	6	mix	node[01-06]
hybrid	up	14-00:00:0	6	mix	node[01-06]

4.1 Примеры программ и скриптов запуска

В данном разделе приведены несколько программ на языке C++, демонстрирующих использование базовых средств параллелизации с примерами скриптов запуска. Они находятся у каждого пользователя в его домашней директории: $HOME/Examples/

В скриптах красным цветом выделены параметры, требующие особого внимания, так как от них зависит, запустится ли вообще ваша задача.

4.1.1 Программа с использованием MPI

Пример программы, использующей MPI, находится в файле $HOME/Examples/MPI/mpiProg.cc:

```
#include <mpi.h>
#include <iostream>
#include <unistd.h>

int main(int argc, char* argv[]){
    int size, rank;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    char* hostname = new char[128];
```

```
        gethostname(hostname, 128);

        std::cout << "I am " << rank << " of " << size <<
                    " processors on host " << hostname << std::endl;

        delete[] hostname;

        MPI_Finalize();

        return 0;
    }
```

Для компиляции программы необходимо подключить соответствующий модуль openmpi-4.0.2 :

```
module load openmpi-4.0.2
```

Компиляция программы:

```
mpicxx mpiProg.cc -o mpiProg
```

Пример скрипта запуска находится в файле $HOME/Examples/MPI/sbatch.cmd. Рассмотрим его построчно:

`#!/bin/bash`	Интерпретатор скрипта
`#SBATCH --partition=cpu`	Очередь для задач на ЦПУ
`#SBATCH --job-name=MPI_test`	Имя задачи
`#SBATCH --nodes=2`	Требуемое количество вычислительных узлов
`#SBATCH --ntasks-per-node=20`	Количество MPI-процессов на каждом узле
`#SBATCH --time=10:00`	Максимальное время счета. Если задача не закончится раньше, по истечении этого времени она будет снята системой.
`#SBATCH --output=stdout.txt`	Стандартный вывод перенаправить в файл stdout.txt
`#SBATCH --error=stderr.txt`	Вывод ошибок перенаправить в файл stderr.txt
`#SBATCH --input=/dev/null`	Программа не требует ввода данных от пользователя (либо указать файл, из которого будет происходить ввод данных)

`#SBATCH --export=ALL`	Передать переменные среды вычислительным узлам.
`module load openmpi-4.0.2`	Загрузить модули (если стоит опция #SBATCH --export=ALL, эту module можно не вызывать)
`# Run the parallel MPI executable`	
`srun ~/Examples/MPI/mpiProg`	Отправить программу mpiProg на выполнение

Запуск программы в очередь выполняется командой

```
sbatch sbatch.cmd
```

Результат работы программы появится в файле stdout.txt. Фрагмент из этого файла приведен ниже:

```
I am 11 of 40 processors on host node01.hpl.itpm
I am 22 of 40 processors on host node02.hpl.itpm
I am 14 of 40 processors on host node01.hpl.itpm
```

Порядок строк при повторном запуске может быть другим.

4.1.2 Гибридная MPI+OpenMP программа

Пример программы, использующей MPI находится в файле $HOME/Examples/MPI+OpenMP/mpi_ompi_Prog.cc:

```
#include <mpi.h>
#include <omp.h>
#include <iostream>
#include <unistd.h>

// ******************* main
int main( int argc, char *argv[]) {
    int size;
    int rank;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    char* hostname = new char[128];
```

```
        gethostname(hostname, 128);

    #pragma omp parallel
        {
    #pragma omp critical // section is included for seq. output
            {
                std::cout << "I am " << rank << " of " << size <<
                    " processors on host " << hostname << "\n" <<
                    " Threads number: " << omp_get_num_threads() <<
                    " Current thread: " <<  omp_get_thread_num() <<
                    std::endl;
            }
        }

        delete [] hostname;

        MPI_Finalize();
        return 0;

    }
```

Для компиляции программы необходимо подключить модуль openmpi-4.0.2:

```
module load openmpi-4.0.2
```

Компиляция программы:

```
mpicxx -fopenmp mpi_omp_Prog.cc -o mpi_omp_Prog
```

Пример скрипта запуска находится в файле $HOME/Examples/MPI+OpenMP/sbatch.cmd . Рассмотрим его построчно:

`#!/bin/bash`	Интерпретатор скрипта
`#SBATCH --partition=cpu`	Очередь для задач на ЦПУ
`#SBATCH --job-name=MPI_OMP_test`	Имя задачи
`#SBATCH --nodes=2`	Требуемое количество вычислительных узлов
`#SBATCH --ntasks-per-node=2`	Количество MPI-процессов на каждом узле
`#SBATCH --cpus-per-task=3`	Количество OpenMP-процессов на каждом MPI-потоке
`#SBATCH --time=10:00`	Максимальное время счета. Если

	задача не закончится раньше, по истечении этого времени она будет снята системой.
`#SBATCH --output=stdout.txt`	Стандартный вывод перенаправить в файл stdout.txt
`#SBATCH --error=stderr.txt`	Вывод ошибок перенаправить в файл stderr.txt
`#SBATCH --input=/dev/null`	Программа не требует ввода данных от пользователя (либо указать файл, из которого будет происходить ввод данных)
`#SBATCH --export=ALL`	Передать переменные среды вычислительным узлам.
`module load openmpi-4.0.2`	Загрузить модули (если стоит опция #SBATCH -- export=ALL, эту module можно не вызывать)
`# Run the parallel MPI executable`	
`srun $HOME/Examples/MPI+OpenMP/mpi_omp_Prog`	Отправить программу mpi_omp_Prog на выполнение

Запуск программы в очередь выполняется командой

```
sbatch sbatch.cmd
```

Результат работы программы появится в файле stdout.txt. Фрагмент из этого файла приведен ниже:

```
I am 1 of 4 processors on host node01.hpl.itpm
 Threads number: 6 Current thread: 4
I am 2 of 4 processors on host node02.hpl.itpm
 Threads number: 6 Current thread: 1
I am 2 of 4 processors on host node02.hpl.itpm
 Threads number: 6 Current thread: 5
```

Порядок строк при повторном запуске может быть другим.

4.1.3 Программа для расчетов на ГПУ

Пример программы для использования ГПУ приведен в файле

$HOME/Examples/CUDA/cuda_query.cu:

```
#include <stdio.h>
#include <unistd.h>

int main ( int argc, char *  argv [] )
{
    int            deviceCount;
    cudaDeviceProp devProp;

    char* hostname = new char[128];
    gethostname(hostname, 128);
    cudaGetDeviceCount ( &deviceCount );

      printf ( "Found %d devices on host %s\n", deviceCount,
hostname );

    for ( int device = 0; device < deviceCount; device++ )
    {
        cudaGetDeviceProperties ( &devProp, device );

        printf ( "Device %d\n", device );
        printf ( "Compute capability     : %d.%d\n",
                 devProp.major, devProp.minor );
        printf ( "Name                   : %s\n",
                 devProp.name );
        printf ( "Total Global Memory    : %ld\n",
                 devProp.totalGlobalMem );
        printf ( "Shared memory per block: %ld\n",
                 devProp.sharedMemPerBlock );
        printf ( "Registers per block    : %d\n",
                 devProp.regsPerBlock );
        printf ( "Warp size              : %d\n",
                 devProp.warpSize );
        printf ( "Max grid size (blocks) : %d %d %d\n",
                 devProp.maxGridSize[0],
                 devProp.maxGridSize[1],
                 devProp.maxGridSize[2] );

        printf ( "Max block size(threads): %d %d %d\n",
                 devProp.maxThreadsDim[0],
                 devProp.maxThreadsDim[1],
                 devProp.maxThreadsDim[2] );

        printf ( "Max threads per block  : %d\n",
                 devProp.maxThreadsPerBlock );
        printf ( "Total constant memory  : %ld\n",
```

```
                devProp.totalConstMem );
        }

        delete[] hostname;
        return 0;
    }
```

Для компиляции программы необходимо подключить модуль cuda/10.1:

```
module load cuda/10.1
```

Компиляция программы:

```
nvcc cuda_query.cu -o cuda_query
```

Пример скрипта запуска находится в файле $HOME/Examples/CUDA/sbatch.cmd. Рассмотрим его построчно:

`#!/bin/bash`	Интерпретатор скрипта
`#SBATCH --partition=gpu`	Очередь для задач на ГПУ
`#SBATCH --job-name=CUDA_test`	Имя задачи
`#SBATCH --nodes=1`	Требуемое количество вычислительных узлов
`#SBATCH --ntasks-per-node=1`	Количество MPI-процессов на каждом узле
`#SBATCH --cpus-per-task=1`	Количество OpenMP-процессов на каждом MPI-потоке
`#SBATCH --gres=gpu:1`	Количество ГПУ на каждом узле
`#SBATCH --time=10:00`	Максимальное время счета. Если задача не закончится раньше, по истечении этого времени она будет снята системой.
`#SBATCH --output=stdout.txt`	Стандартный вывод перенаправить в файл stdout.txt
`#SBATCH --error=stderr.txt`	Вывод ошибок перенаправить в файл stderr.txt
`#SBATCH --input=/dev/null`	Программа не требует ввода данных от пользователя (либо указать файл, из которого будет происходить ввод данных)

`#SBATCH --export=ALL`	Передать переменные среды вычислительным узлам.
`module load cuda/10.1`	Загрузить модули (если стоит опция #SBATCH -- export=ALL, эту module можно не вызывать)
`# Run the parallel MPI executable`	
`srun $HOME/Examples/CUDA/cuda_query`	Отправить программу cuda_query на выполнение

Запуск программы в очередь выполняется командой

```
sbatch sbatch.cmd
```

Результат работы программы появится в файле *stdout.txt.* Пример этого вывода показан ниже:

```
        Found 2 devices on host node01.hpl.itpm
        Device 0
        Compute capability     : 7.0
        Name                   : Tesla V100-PCIE-32GB
        Total Global Memory    : 34089730048
        Shared memory per block: 49152
        Registers per block    : 65536
        Warp size              : 32
        Max grid size (blocks) : 2147483647 65535 65535
        Max block size(threads): 1024 1024 64
        Max threads per block  : 1024
        Total constant memory  : 65536
        Device 1
        Compute capability     : 7.0
        Name                   : Tesla V100-PCIE-32GB
        Total Global Memory    : 34089730048
        Shared memory per block: 49152
        Registers per block    : 65536
        Warp size              : 32
        Max grid size (blocks) : 2147483647 65535 65535
        Max block size(threads): 1024 1024 64
        Max threads per block  : 1024
Total constant memory  : 65536
```