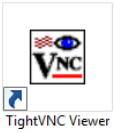
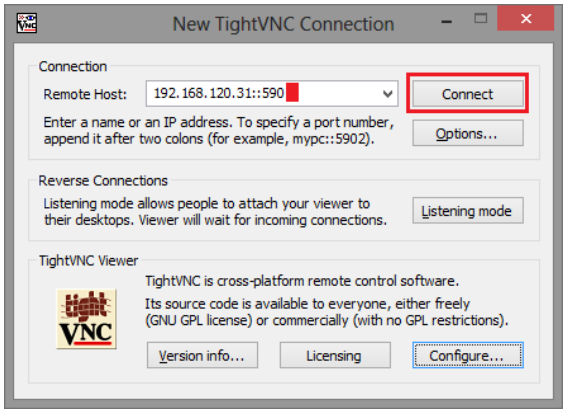
Подключение.

Воспользуйтесь ярлыком TightVNC Viewer на рабочем столе для подключения к удаленному рабочему столу

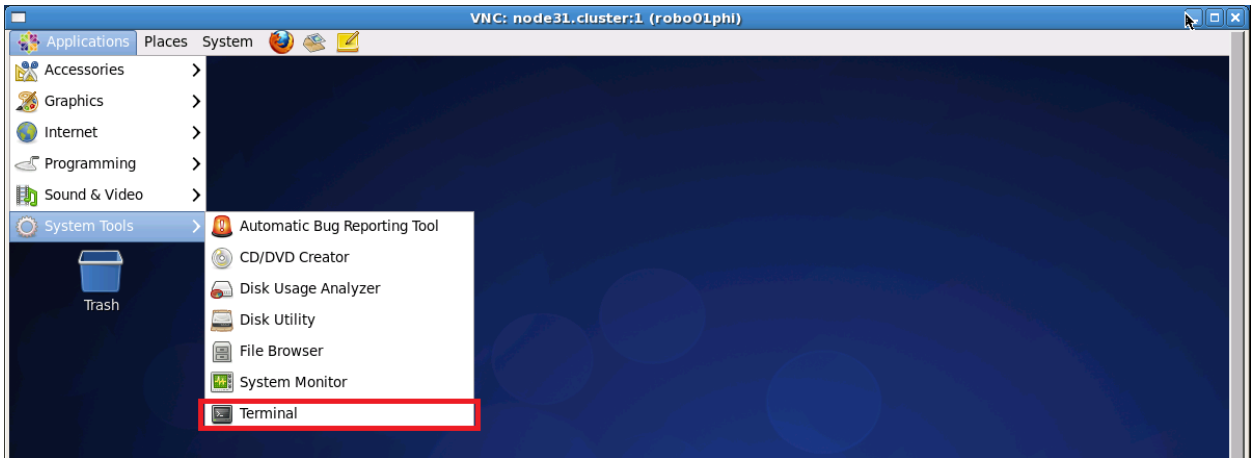


Введите IP-адрес удаленного компьютера и порт, согласно выданным индивидуальным учетным данным. Нажмите Connect для подключения.



После успешного подключения Вы увидете рабочий стол удаленной машины.

Для работы потребуется эмулятор терминала – Terminal.



Конфигурация.

Для проведения лабораторной работы доступен узел с 2-мя сопроцессорами Intel Xeon Phi:

node31
IP-адрес 192.168.120.31
2 CPU Intel Xeon E5-2660
128GB RAM
2 сопроцессора Intel Xeon Phi 5110P (mic0 и mic1)

Сборка и запуск MPI приложений с использованием Intel MPI на MIC (Many **I**ntegrated **C**ore Architecture**)**

Задание 0 - Предустановки

Установлен и настроен MPSS (созданы пользовательские аккаунты на картах, настроен беспарольный доступ ssh/scp).

- Компилятор установлен в директорию /opt/intel/compilers/composerxe.

Выполните в терминале команду для инициализации компилятора:

```
export ICCROOT=/opt/intel/compilers/composerxe
```

- Intel MPI установлен в директорию /opt/intel/compilers/impi/4.1.1.045.

Выполните в терминале команду для инициализации библиотеки Intel MPI:

```
export MPIROOT=/opt/intel/compilers/impi/4.1.1.045
```

*После перезагрузки карт сопроцессоров некоторые библиотеки компилятора должны быть заново загружены в файловую систему (RAM-диск) сопроцессоров. Для простоты используют директорию **/lib64/**, следовательно, нет необходимости устанавливать переменную **LD_LIBRARY_PATH**.*

Как правило, требуются библиотеки OpenMP и CilkPlus. Они могут быть скопированы командами:

```
sudo scp $ICCROOT/lib/mic/libiomp5.so mic0:/lib64/
sudo scp $ICCROOT/lib/mic/libcilkrts.so.5 mic0:/lib64/
```

(Для сервера с двумя сопроцессорами, используйте команды для mic0 и mic1).

Установите и проверьте пользовательское окружение компилятора, библиотеки Intel MPI, выполнив соответствующие скрипты:

```
source $ICCROOT/bin/compilervars.sh intel64
source $MPIROOT/intel64/bin/mpivars.sh
```

и команду

```
which icc mpiicc
```

Удостоверьтесь, что ssh доступ, требуемый MPI, работает без запроса пароля:

```
ssh mic0 hostname
```

Проверьте доступность компилятора, скриптов Intel MPI на сопроцессоре, выполнив команду:

```
ssh mic0 which mpiicc
```

Позвольте библиотеке Intel MPI использовать сопроцессоры, определив значение переменной

```
export I_MPI_MIC=1
```

Задание 1 – Основы.

Чистый MPI. Нативная модель.

Все проекты для лабораторной лежат по адресу ***Lab1.2_Intel_MPI_Xeon_PHI/ source_codes_from_Intel(R)/ part3/ intel_mpi_lab_C*** *в вашей домашней директории.*

Дистрибутив Intel MPI включает директорию test, содержащую пример программы на языках C, C++, и Fortran.

Зайдите в директорию **1_Basics** , где находится копия исходного тестового файла **test.c** из the Intel MPI дистрибутива. Вы должны использовать этот код и два варианта для первых запусков приложений с Intel MPI на MIC.

Задайте рабочую директорию на сопроцессоре, обычно это домашняя директория пользователя:

```
export MICHOME=/home/$USER/
```

Скомпилируйте и слинкуйте исходный файл с помощью компилятора Intel для хостового процессора XEON, используя обычный Intel MPI скрипт:

```
mpiicc -o test test.c
```

Задайте тип коммуникатора(ов) (разделяемая память (shm) для внутриузловых коммуникаций, tcp протокол для межузловых коммуникаций MPI процессов):

```
export I_MPI_FABRICS=shm:tcp
```

Запускаем приложение на хостовых CPU с двумя MPI процессами:

```
mpirun -n 2 ./test
```

и должны увидеть подобный вывод:

```
Hello world: rank 0 of 2 running on node31.cluster
Hello world: rank 1 of 2 running on node31.cluster
```

Затем скомпилируйте и слинкуйте исходный файл для сопроцессора Xeon Phi, используя ключ компилятора **"-mmic"**. Благодаря этому ключу, скрипт Intel MPI передаст версии библиотек Intel MPI для архитектуры MIC линковщику (можете добавьте ключ **"-v"** , чтобы увидеть этот процесс):

```
mpiicc -mmic -o test.MIC test.c
```

Суффикс **".MIC"** в имени файла добавляется пользователем, чтобы отличать бинарный файл от бинарного файла для хостового CPU. Это может быть любой суффикс!

Теперь бинарный MIC-файл должен быть загружен в файловую систему сопроцессора командой:

```
scp test.MIC mic0:$MICHOME
```

Запустите ваше первое Intel MPI приложение для MIC в нативном режиме запуска на сопроцессоре. **Запуск выполняется с хоста!**:

mpirun -wdir $MICHOME -host node31-mic0 -n 2 ./test.MIC

Должны увидеть подобный вывод:

Hello world: **rank 0** of **2** running on **node31-mic0.cluster**

Hello world: **rank 1** of **2** running on **node31-mic0.cluster**

Запуск MPI приложения с хоста с использованием NFS директории

Альтернативно, нужно скопировать файл в NFS-директорию, смонтированную на MIC.

Скопируйте MIC-бинарник **test.MIC** в NFS-директорию (/var/public/$USER) и используйте соответствующий путь, видимый в файловой системе сопроцессора. Задайте рабочую директорию на сопроцессоре:

export MICHOME=/var/public/$USER

cp test.MIC $MICHOME

(также копируем и хостовый вариант)

cp test $MICHOME

cd $MICHOME

ls

В выводе должны увидеть наш файл среди других:

> ***test.MIC***

Теперь запустите приложение для MIC в режиме запуска на сопроцессоре:

mpirun -wdir /var/public/$USER -host node31-mic0 -n 2 ./test.MIC

Должны увидеть подобный вывод:

Hello world: rank **0** of **2** running on **node31-mic0.cluster**

Hello world: rank **1** of **2** running on **node31-mic0.cluster**

Запуск MPI приложения непосредственно из окружения файловой системы Xeon Phi

Альтернативно, нужно перейти в файловую систему сопроцессора и запустить приложение оттуда обычным способом запуска MPI приложений.

Копируем исполняемый файл на сопроцессор или удостоверяемся, что исполняемый файл расположен в NFS директории.

вариант с копированием в файловую систему сопроцессора:

scp test.MIC mic0:~

ssh mic0

ls

> ***test.MIC …***

Запускаем:

mpirun -host mic0 -perhost 60 -np 120 ./test.MIC

для выхода из окружения ФС сопроцессора введите

exit

вариант с NFS директорией:

ssh mic0

cd /var/public/$USER/

ls

> ***test.MIC …***

mpirun -host mic0 -perhost 60 -np 120 ./test.MIC

для выхода из окружения ФС сопроцессора введите

exit

Запуск тестового MPI приложения с хоста в симметричном режиме на хостовых CPU и сопроцессоре Xeon Phi

Разместив два нативных приложения (**test** - для CPU Xeon и **test.MIC** - для сопроцессора Xeon Phi) рядом, можно запустить тестовый код в симметричном режиме на хостовых CPU и MIC.

export MICHOME=/var/public/$USER

cd $MICHOME

ls

> ***test test.MIC***

Запускаем гибридное тестовое приложение с хоста:

mpirun -host node31 -n 2 ./test : -wdir $MICHOME -host mic0 -n 4 ./test.MIC

Должны увидеть подобный вывод:

Hello world: rank **0** of **6** running on **node31.cluster**

Hello world: rank **1** of **6** running on **node31.cluster**

Hello world: rank **2** of **6** running on **node31-mic0.cluster**

Hello world: rank **3** of **6** running on **node31-mic0.cluster**

Hello world: rank **4** of **6** running on **node31-mic0.cluster**

Hello world: rank **5** of **6** running on **node31-mic0.cluster**

Обратите внимание: в симметричном способе запуска вы должны указать флаг «-host» как для хостовых, так и для MIC процессов MPI.

Альтернативно предыдущей команде вы можете задать machinefile (список хостов и количество процессов на хост). Используйте этот файл вместе с переменной окружения **I_MPI_MIC_POSTFIX**. Значение этой переменной автоматически добавится к именам исполняемых на сопроцессоре файлов.

Создаем файл

echo node31:2 > machinefile

и дописываем в него информацию о хостах-сопроцессорах

echo mic0:4 >> machinefile

Определяем переменную

export I_MPI_MIC_POSTFIX=.MIC

и запускаем приложение в 6 процессов (2 на хосте и 4 на сопроцессорах)

mpirun -machinefile machinefile -n 6 ./test

Деинициализируем переменную

export -n I_MPI_MIC_POSTFIX=.MIC

Перед выполнением следующих заданий с гибридными программами, исследуем планировку (mapping)/ привязку (pinning) Intel MPI процессоров.

Чтобы увидеть информацию о планировке (mapping) процессов, установим значение отладочной переменной **I_MPI_DEBUG** равным или большим 4.

export I_MPI_DEBUG=4

Для чистых MPI программ (негибридных) переменная окружения I_MPI_PIN_PROCESSOR_LIST контролирует планировку/привязку (mapping/ pinning). Для гибридных приложений приоритет отдается значению переменной **I_MPI_PIN_DOMAIN**. Ее значение разбивает (логические) процессоры на непересекающиеся домены, для которых применяется правило «1 MPI процесс на 1 домен».

Повторите предыдущий Intel MPI тест с заданным значением отладочной переменной **I_MPI_DEBUG**.

export I_MPI_DEBUG=4

mpirun -n 2 ./test

mpirun -host mic0 -n 2 ./test.MIC

mpirun -host node31 -n 2 ./test : -host mic0 -n 4 ./test.MIC

Теперь установим значение переменной I_MPI_PIN_DOMAIN через флаг командной строки "-env". Возможные значения переменной: "auto", "omp" (связано с OMP_NUM_THREADS), или фиксированное число логических ядер. Если задать значение I_MPI_PIN_DOMAIN через глобальный "-genv" флаг, то оно идентично установится для хоста и сопроцессоров Xeon Phi.

Обычно рекомендуется задавать автоматическую адаптацию под архитектурные особенности:

mpirun -env I_MPI_PIN_DOMAIN auto -n 2 ./test

mpirun -env I_MPI_PIN_DOMAIN auto -host mic0 -n 2 ./test.MIC

mpirun -env I_MPI_PIN_DOMAIN 4 -host node31 -n 2 ./test : -env\
I_MPI_PIN_DOMAIN 12 -host mic0 -n 4 ./test.MIC

Знакомство с гибридными MPI/OpenMPI приложениями для сопроцессора

Теперь рассмотрим процедуру сборки и запуска гибридного MPI/OpenMP приложения на сопроцессоре Xeon Phi. Вспоминая работу с OpenMP приложениями, сравним два тестовых кода Intel MPI test:

```
diff test.c test-openmp.c
```

Скомпилируем и слинкуем приложение с использованием флага "-openmp":

```
mpiicc -openmp -o test-openmp test-openmp.c
mpiicc -openmp -mmic -o test-openmp.MIC test-openmp.c
```

Благодаря флагу "-openmp" скрипт Intel® MPI автоматически слинкует приложение с соответствующей поточной версией библиотеки Intel MPI (libmpi_mt.so).

Как и ранее, запускаем приложения:

```
unset I_MPI_DEBUG  # (чтобы отключить отладочные сообщения)
mpirun -n 2 ./test-openmp
mpirun -host mic0 -n 2 ./test-openmp.MIC
mpirun -host node31 -n 2 ./test-openmp : -host mic0 -n 4 ./test-openmp.MIC
```

Вы должны увидеть довольно большой вывод информации! Это произошло из-за того, что по умолчанию библиотека OpenMP порождает столько потоков, сколько доступно логических процессоров. Для следующих тестов будем явно задавать значения **OMP_NUM_THREADS**: различные для хоста и сопроцессора Xeon Phi.

В следующем тесте проверим OpenMP сродство. Важно: диапазон используемых логических процессоров всегда определяется расщеплением потоков на основе переменной **I_MPI_PIN_DOMAIN**.

Кроме того, будем использовать значение **I_MPI_PIN_DOMAIN=omp**, чтобы увидеть как выполнение приложения зависит от значения **OMP_NUM_THREADS**:

```
mpirun -prepend-rank -env KMP_AFFINITY verbose -env \
OMP_NUM_THREADS 4 -env I_MPI_PIN_DOMAIN auto -n 2 \
./test-openmp 2>&1 | sort > test_run1.txt

mpirun -prepend-rank -env KMP_AFFINITY verbose -env \
OMP_NUM_THREADS 4 -env I_MPI_PIN_DOMAIN omp \
-host mic0 -n 2 ./test-openmp.MIC 2>&1 | sort > test_run2.txt
```

```
mpirun -prepend-rank -env KMP_AFFINITY verbose -env \
OMP_NUM_THREADS 4 -env I_MPI_PIN_DOMAIN 4 \
-host node31 -n 2 ./test-openmp : -env KMP_AFFINITY verbose -env \
OMP_NUM_THREADS 6 -env I_MPI_PIN_DOMAIN 12 \
-host mic0 -n 4 ./test-openmp.MIC 2>&1 | sort > test_run3.txt
```

Важно: обычно избегают раздела физических ядер Xeon Phi между процессами MPI. Для этого либо количество процессов MPI выбирается так, чтобы **I_MPI_PIN_DOMAIN=auto** создало автоматически домены, охватывающие полные ядра или значение этой переменной окружения должно быть кратно 4.

Используйте "scatter", "compact", или "balanced" (только для Xeon) для модификации сродства OpenMP.

```
mpirun -prepend-rank -env KMP_AFFINITY \
verbose,granularity=thread,scatter -env OMP_NUM_THREADS 4 -env \
I_MPI_PIN_DOMAIN auto -n 2 ./test-openmp > test_run4.txt
```

```
mpirun -prepend-rank -env KMP_AFFINITY \
verbose,granularity=thread,compact -env OMP_NUM_THREADS 4 -env \
I_MPI_PIN_DOMAIN omp -host mic0 -n 2 ./test-openmp.MIC \
2>&1 | sort > test_run5.txt
```

```
mpirun -prepend-rank -env KMP_AFFINITY \
verbose,granularity=thread,compact -env OMP_NUM_THREADS 4 -env \
I_MPI_PIN_DOMAIN 4 -host node31 -n 2 ./test-openmp : -env \
KMP_AFFINITY verbose,granularity=thread,balanced -env \
OMP_NUM_THREADS 6 -env I_MPI_PIN_DOMAIN 12 \
-host mic0 -n 4 ./test-openmp.MIC 2>&1 | sort > test_run6.txt
```

Заметьте, что как и другие опции, опция сродства OpenMP может быть задана по-разному в каждом из наборов аргументов Intel MPI, то есть, различна для хоста и сопроцессоров.

MPI+OFFLOAD

Рассмотрим запуск тестового приложения Intel MPI test с частью кода, выгружаемого для исполнения на сопроцессоре

Вспоминая методы разгрузки кода на сопроцессор, сравните два исходных файла:

```
diff test.c test-offload.c
```

Скомпилируйте и слинкуйте приложение для хостовых CPU Xeon с флагом компилятора "-openmp", как и ранее. Начиная с версии 13 компиляторы Intel автоматически распознают директивы offload и создают соответствующий бинарный код. Если требуется игнорировать директивы offload, компилируйте с флагом "-no-offload":

```
mpiicc -openmp -o test-offload test-offload.c
```

Запустите приложение на хосте:

```
OFFLOAD_REPORT=2 mpirun -prepend-rank -env KMP_AFFINITY \
granularity=thread,scatter -env OMP_NUM_THREADS 4 -n 2 ./test-offload
```

Запустите приложение еще раз, но с использованием фильтра и сортировки для выборки только важной выходной информации:

```
mpirun -prepend-rank -env KMP_AFFINITY \
verbose,granularity=thread,scatter -env OMP_NUM_THREADS 4 \
-n 2 ./test-offload 2>&1 | grep bound | sort
```

Должны увидеть, что все OpenMP потоки привязываются к идентичным потокам сопроцессора! При этом переменная I_MPI_PIN_DOMAIN не может быть использована, так как разделение на домены будет рассчитано в соответсвии с числом логических процессоров на хосте с Xeon CPU!

Решением такой проблемы является явное указание списка процессоров на MPI процесс:

```
OFFLOAD_REPORT=2 mpirun -prepend-rank -env KMP_AFFINITY \
granularity=thread,proclist=[1-16:4],explicit -env OMP_NUM_THREADS 4 \
-n 1 ./test-offload : -env KMP_AFFINITY \
granularity=thread,proclist=[17-32:4],explicit -env OMP_NUM_THREADS 4 \
-n 1 ./test-offload
```

Повторите запуск с использованием фильтра и сортировки для выборки только важной выходной информации:

```
mpirun -prepend-rank -env KMP_AFFINITY \
verbose,granularity=thread,proclist=[1-16:4],explicit -env \
OMP_NUM_THREADS 4 -n 1 ./test-offload : -env KMP_AFFINITY \
verbose,granularity=thread,proclist=[17-32:4],explicit -env \
OMP_NUM_THREADS 4 -n 1 ./test-offload 2>&1 | grep bound | sort
```

Задание 2 - Гибридные приложения MPI/OpenMP

Найдите директорию 2_MPI_OpenMP.

В этой директории находятся исходные коды приложения Poisson.

Выполните следующие команды для компиляции бинарных файлов для Intel Xeon хоста and Intel Xeon Phi сопроцессоров. При компиляции будет использован

флаг "-openmp" для создания гибридной версии кода. Потоки OpenMP реализованы в файле compute.c:

```
make clean; make
make clean; make MIC
```

Запустите приложение Poisson только хосте, только на Intel Xeon Phi, и в симметричном режиме на обеих платформах.

Программа принимает следующие входные параметры:

"-n x" изменение размера сетки. По умолчанию, x=1000.

"-iter x" задает максимальное число итераций. По умолчанию, x= 4000.

"-prows x" изменяет ряды процессора. По умолчанию вычисляется.

```
mpirun -env OMP_NUM_THREADS 12 -n 1 ./poisson -n 3500 -iter 10

mpirun -env OMP_NUM_THREADS 12 -host mic0 \
-n 1 ./poisson.MIC -n 3500 -iter 10

mpirun -env OMP_NUM_THREADS 12 -host node31 \
-n 1 ./poisson -n 3500 -iter 10 : -env OMP_NUM_THREADS 12 \
-host mic0 -n 1 ./poisson.MIC -n 3500 -iter 10
```

Изменяйте число MPI процессов и потоков OpenMP (скорее всего, они будут различными для хоста и сопроцессора) для оптимизации производительности.

Используйте знания о планировке MPI процессов и сродстве потоков OpenMP, полученные при выполнении предыдущих простых знаданий.

Задание 3 – Использование Intel Trace Analyzer and Collector

Зайдите в директорию 4_ITAC.

Выполните следующие команды для компиляции бинарных файлов для Intel Xeon хоста and Intel Xeon Phi сопроцессоров. При компиляции будет использован флаг "-tcollect", требуемый инструментами ITAC:

```
source /opt/intel/compilers/itac/8.1.3.037/intel64/bin/itacvars.sh
export I_MPI_FABRICS=shm:tcp
export I_MPI_MIC=1
make clean; make
make clean; make MIC
```

Выполните приложения только на хосте, только на сопроцессоре, и в симметричном режиме на обеих платформах (смотри команды ниже). Каждый запуск приложения создает файл трассировки ITAC в формате stf, который будет проанализирован с помощью анализатора трассировки ITAC.

```
export VT_LOGFILE_FORMAT=stfsingle
```

```
mpirun -env OMP_NUM_THREADS 1 -n 12 ./poisson -n 3500 -iter 10

traceanalyzer poisson.single.stf

mpirun -env OMP_NUM_THREADS 1 -host mic0 \
-n 12 ./poisson.MIC -n 3500 -iter 10

traceanalyzer poisson.MIC.single.stf

mpirun -env OMP_NUM_THREADS 1 -host node31 \
-n 12 ./poisson -n 3500 -iter 10 : -env OMP_NUM_THREADS 1 \
-host mic0 -n 12 ./poisson.MIC -n 3500 -iter 10

traceanalyzer poisson.single.stf
```

Для симметричного запуска с одинаковым числом MPI процессов на хосте и сопроцессоре, вы увидите дисбаланс нагрузки из-за различия характеристик архитектуры (тактовая частота, и т.п.).