

Introduction to HPC

Preparing environment

Radim Janalík
Juraj Kardoš

Università della Svizzera italiana

September 17, 2019

ICS cluster - icsmaster

- 2 login nodes
 - icsmaster[01,02]
- 40 compute nodes
 - icsnode[01-40]
 - 24 nodes only with CPU
 - 2 x Intel E5-2650 v3, 20 (2 x 10) cores
 - 64GB RAM
 - 8 nodes with GPU
 - 2 x Intel E5-2650 v3, 20 (2 x 10) cores
 - 1 x NVIDIA GeForce GTX 1080, 2560 CUDA cores
 - 128GB RAM
 - 8 nodes with Intel Xeon Phi coprocessors
 - 2 x Intel E5-2650 v3, 20 (2 x 10) cores
 - 3 x Intel Xeon Phi SC7120P coprocessors, 61 cores
 - 128GB RAM

Accessing icsmaster

- You will receive your login credentials by email
- Connect to the cluster using ssh

```
$ ssh studXX@hpc.ics.usi.ch
```

- To avoid typing the password you can generate ssh-key (on your laptop) and copy it to the cluster

```
$ ssh-keygen -t rsa
```

```
$ ssh-copy-id -i ~/.ssh/id_rsa.pub studXX@hpc.ics.usi.ch
```

Accessing icsmaster

- Add host configuration to `~/.ssh/config`

```
Host icsmaster
  Hostname hpc.ics.usi.ch
  Port 22
  User studXX
  IdentityFile ~/.ssh/id_rsa
```

- Now you can connect to icsmaster without password

```
$ ssh icsmaster
```

Moving data

■ laptop → icsmaster

```
$ scp file.c icsmaster:~/remote_dir/  
$ scp -r local_dir icsmaster:~/remote_dir/
```

■ icsmaster → laptop

```
$ scp icsmaster:~/remote_dir/file.c local_dir/  
$ scp -r icsmaster:~/remote_dir/ local_dir/
```

Modules

- Software on the cluster is organized into modules
- Before using some program, you have to load a module

```
$ module load gcc/6.1.0
```

- You can unload modules you don't want anymore

```
$ module unload gcc/6.1.0
```

- Or you can swap modules to get different version

```
$ module swap gcc/6.1.0 gcc/5.3.0
```

- List currently loaded modules

```
$ module list
```

- List all available modules

```
$ module avail
```

Editing code

- To edit files on the cluster you can use vim

```
$ vim main.c
```

- For easier moving / editing, you can mount home directory from cluster to your local machine, then you can work with files on the cluster as if it would be on your laptop

- First install FUSE and SSHFS from <https://osxfuse.github.io>
- Then you can mount remote directory

```
$ sshfs icsmaster: <mountpoint>
```

- And unmount

```
$ umount <mountpoint>
```

Compiling code

- Load modules before compiling the code

```
$ module load gcc/6.1.0
```

- Then you can compile the code

```
$ gcc main.c -o main
```

- Pro tip: add `module load ...` to your `~/.bashrc`
 - It will load modules automatically when you log in

Running code

- When you log in to the cluster, you are on login node
 - There are two login nodes
 - icsmaster01
 - icsmaster02
 - You can use login node to edit and compile your code
 - **But you should never run the code on login node**
- For running your code, there are 40 compute nodes
 - icsnode[01-40]
- There are two ways how you can work on compute node
 - Interactive session
 - Batch job

Interactive session

- In interactive session you have direct access to compute node from your terminal
- Interactive session is useful especially for debugging
- When you allocate a node, nobody else can use it at the same time
- But it can take a long time to get access to the node

- First you have to allocate the node
- Let's say you want 1 node for 1 hour

```
$ salloc --nodes=1 --time=01:00:00
```

- Then you can run your app on the compute node

```
$ ./your_app
```

- Or you can use srun that does the allocation automatically

```
$ srun --nodes=1 ./your_app
```

Batch job

- When running batch job, you don't have direct access to the compute node
- You write a script with commands you want execute on the node
- The script is added to a queue and executed later
- Output of the script is written to a file. You can look at it when the job is finished
- Batch job is useful when you have working code and you want to run your app on large data

Batch job

■ Job script template

```
#!/bin/bash -l

#SBATCH --job-name=my_job
#SBATCH --time=01:00:00
#SBATCH --nodes=1
#SBATCH --output=%j.out
#SBATCH --error=%j.err

# load modules

# your commands
```

■ Add job to a queue

```
$ sbatch job.sh
```

■ Show your running and queued jobs

```
$ squeue -u studXX
```

■ Cancel job

```
$ scancel <job_id>
```

Reservation

- For the lectures there are some nodes reserved only for the students of this class

- Use argument `--reservation=HPC_tuesday` or `--reservation=HPC_wednesday`

```
$ salloc --reservation=HPC_tuesday  
$ srun --reservation=HPC_tuesday ./your_app
```

- Use the reservation only in the class
- When the reservation is not active, you would wait forever

- In class

```
$ srun --reservation=HPC_tuesday ./your_app
```

- At home

```
$ salloc --nodes=1 --time=01:00:00  
$ ./your_app
```

Specifying resources

- You can call `salloc`, `sbatch` or `srun` with the following parameters

- | | |
|--|---|
| <code>-N</code> or <code>--nodes</code> | set number of nodes |
| <code>-n</code> or <code>--ntasks</code> | set number of tasks |
| <code>--mem=MB</code> | Specify the real memory required per node |
| <code>--exclusive</code> | Only your job is allowed on the nodes allocated to this job |

- For debugging use `--mem=10GB`

- You can share node with other users
- It's easier to get node for you and for others

- For running benchmarks use `--exclusive`

- To make sure only your job is running on the node
- Other jobs can't influence your measurements

Git repository with source codes

- We prepared for you git repository with source codes for this lecture
 - https://github.com/rjanalik/HPC_2019

- Clone the repository on both icsmaster and your laptop

```
$ git clone https://github.com/rjanalik/HPC_2019.git
```

- Later we will update the repository with source codes for other lectures and assignments
- To download the latest version use

```
$ git pull
```

BLAS and LAPACK on OS X

- BLAS and LAPACK libraries are already installed on your Mac
- To link BLAS library, add `-lblas` switch to `gcc`

```
$ gcc blas.c -o blas -lblas  
$ ./blas
```

- For LAPACK add `-lblas` and `-llapack`

```
$ g++ -c -Wall main.cpp -o main.o  
$ g++ main.o -o testprog -lblas -llapack  
$ ./testprog
```


OpenMPI on OS X

- To compile and run MPI applications on your Mac you have to build OpenMPI first
- To not mess up system directories I compile everything in ~/Apps
 - It makes your life easier when you decide to remove it
 - Feel free to use your favourite directory instead of ~/Apps

- Download the latest version from

<https://www.open-mpi.org/software/ompi/v3.1/>

```
$ cd ~/Apps
```

```
$ wget https://www.open-mpi.org/software/ompi/v3.1/downloads/↵
    openmpi-3.1.2.tar.gz
```

```
$ tar -zxf openmpi-3.1.2.tar.gz
```

- Configure, build and install the library
- Don't forget --prefix

```
$ cd openmpi-3.1.2/
```

```
$ ./configure --prefix=$HOME/Apps/openmpi-3.1.2
```

```
$ make all
```

```
$ make install
```

OpenMPI on OS X

- Now you have OpenMPI library installed in `~/Apps/openmpi-3.1.2`
- To use it you have to add `bin` directory to variable `$PATH`

```
$ export PATH=$PATH:$HOME/Apps/openmpi-3.1.2/bin
```
- You can add this export to `~/.bashrc` so you don't have to do it every time you open terminal

- Now you can try if it works

```
$ mpicc mpi_hello.c -o mpi_hello  
$ mpirun -np 2 ./mpi_hello
```

Intel Math Kernel Library

- Intel Math Kernel Library is part of Parallel Studio XE
- Download and register at <https://software.intel.com/en-us/intel-parallel-studio-xe>
- Choose Intel Parallel Studio XE Student License
- Download Intel Parallel Studio XE Composer Edition for C++
- Install Intel Parallel Studio XE

Intel MKL installation & usage

■ Before using MKL set the environment variables

```
$ . /opt/intel/mkl/bin/mklvars.sh intel64  
$ . /opt/intel/bin/compilervars.sh intel64
```

■ Compile with

```
$ gcc dgemm_example.c -lmkl_intel_lp64 -lmkl_core -lmkl_sequential -lm -o dgemm_example
```

■ Run

```
$ ./dgemm_example
```

■ After successful execution

```
Deallocating memory; Example completed.
```