Course *Parallel and distributed programming*
Computer Lab no. 2: Running MPI on a cluster

IT, Uppsala University

Getting started

During this lab we use the UPPMAX system Snowy.

Briefly about Snowy

Snowy is a cluster, one of the Swedish National Infrastructure for Computing (SNIC) resources. It has 3648 cores, residing in 228 nodes with two CPUs (Intel Xeon E5-2660) HP ProLiant SL230s Gen8, each with 8 cores. The majority of the nodes has 128 GB memory. There are 13 nodes with 256 GB memory and 17 nodes - with 512 GB memory, There is only one 'very fat' node with 1TB of memory, eight CPUs and 80 cores.
The interconnection network is Gigabit Ethernet and 4xQDR band interconnect for MPI. The nodes are connected via 4x FDR10 compared to 4x FDR on Rackham (`https://en.wikipedia.org/wiki/InfiniBand#Performance`, `http://www.mellanox.com/page/performance_infiniband`).
The details about Snowy can be found at `http://uppmax.uu.se/resources/systems/the-snowy-cluster/`.

Accessing Snowy

Snowy's user guide is at
`https://www.uppmax.uu.se/support/user-guides/snowy-user-guide/`.
To access Snowy you need an UPPMAX account, which you should have already gotten by following the instructions from 'How to access UPPMAX resources' 'Course material' at Studium.

Since Snowy does not have login nodes, one has to first login to the Rackham system using `ssh`:

```
$ ssh -Y your-user-name@rackham.uppmax.uu.se
```

In the lab, a number of test programs are to be studied. These are provided on the course webpage at Studium, under 'Course Material/Labs'. The files are collected in an archive `lab2-code.zip`. Download this archive, and copy it to Rackham's file system using the `scp` (or `sftp`) command,

```
$ scp lab2-code.zip your-user-name@rackham.uppmax.uu.se:
```

Note that Rackham and Snowy have the same file system.
The files can then be unpacked by `unzip lab2-code.zip`. The commands `ssh` and `scp` are typically available by default on Linux and Mac systems. On Windows, the `PuTTY` program suite could be installed, which has both an SSH client and a SCP utility. (Alternatively, a full Cygwin Unix environment can be set up.)
To edit files located on your UPPMAX account, you can use a command line editor such as `nano`, `nedit`, `emacs` or `vim`. Alternatively, if your operating system supports X-forwarding (out of the box on Linux, can be set up on Macs), you can login with -Y and then start GUI programs. It is OK to do the editing and compiling on Rackham's login nodes:

```
$ ssh -Y your-user-name@rackham.uppmax.uu.se
$ gedit program.c
```

You can also edit copies of the files locally on your computer, and then use `scp` to transfer each time they are updated.

Using Snowy

On UPPMAX, a module system is used to enable various packages. Using the command `module avail` you can see all the available modules. For this course, we are going to need OpenMPI, which in turn requires a recent version of the GCC compiler. Run the following command to enable both of them:

```
$ module load gcc openmpi
```

You can now compile and run programs with `mpicc` and `mpirun` as before.

To use the Arm DDT (former Allinea DDT) debugger and performance analyser (MAP), load the module `ddt`.

Enabling the use of Infiniband

To use the full functionality of the current version of OpenMPI and the hardware (Infiniband), add the following to your `.bashrc` file the following command:

```
export OMPI_MCA_btl_openib_allow_ib=1
```

The batch system SLURM

When executing a program with mpirun on the command line, we are running it on the login node. This is a 'noisy' environment where the activity of other users affect the run time. Moreover, if your program needs a lot of resources, the node will appear slow for other users. To get accurate measurements, we run the code as a dedicated job on the compute nodes using the batch system SLURM. Specifically, a job is created using a batch script which is submitted using the command `sbatch`.
Consider as example the following script `alltoall.sh`, prepared for the code `alltoall`.

```
#!/bin/bash -l

#SBATCH -M snowy
#SBATCH -A uppmax2023-2-13
#SBATCH --reservation uppmax2023-2-13_1
#SBATCH -p core -n 2
#SBATCH -t 5:00

module load gcc openmpi
mpirun -np 2 ./alltoall
```

It first contains a number of comment lines with configurations to `sbatch`, followed by the actual commands to run.
The script is configured for allocation of two cores for a maximum runtime of 5 minutes, and also specifies the course project ID and the reservation for the lab.
Provided that we have included `#SBATCH -M snowy` in the batch script, the job is submitted to Snowy by issuing

```
$ sbatch alltoall.sh
```

Don't forget to compile `alltoall` before submitting the job. The output of the job is written to the file `slurm-<jobid>.out`.
To see a list of your jobs in the queue, use the command
`squeue -M snowy -u <your-user-name>`. To cancel a running or queued job, use the command `scancel <job-id> -M snowy`. Note that the '-reservation' line is only valid during the scheduled lab session, and will give an error otherwise. Remove it if you want to run outside of that time.
More at `http://uppmax.uu.se/support/user-guides/slurm-user-guide/`.

Storing data on UPPMAX file systems

Note, that the disk space, related to your home directory is rather limited. Larger data files shall be stored under `/crex/proj/uppmax2023-2-13/nobackup/`. As the name suggests, no backup is done for this directory. More information about the rules related to disk storage you find at `https://www.uppmax.uu.se/support/user-guides/disk-storage-guide/`.

Exercises

Exercise 1 (Two-dimensional integral) Consider the code `integral2d.c`, which computes an integral equal to ≈ 2.558041407. The file contains a code for computing this integral, including time measurement.

- Set up a batch script by copying the one from the alltoall example. For the given example `integral2d.c`, setting the project id, the reservation, and the time limit is sufficient;

other options are set on the command line. Also, remove any `-np` option to `mpirun` as we want to run on all available cores.

- Now submit some jobs for different numbers of processors, check the output to see the runtime and plot the speedup. Do more cores always give a speedup over fewer ones? For example,

```
$ sbatch  -n 1 integral2d.sh          # single process
$ sbatch  -n 4 integral2d.sh          # four cores on single node
$ sbatch  -n 16 integral2d.sh         #  16 cores (entire node)
$ sbatch  -p node -n 32 integral2d.sh  #  32 cores (two full nodes)
```

You can do similar experiments but by computing the scaled speedup for the problem.

Exercise 2 Read through the following programs, which illustrate different types of communications in MPI. Compile and run the codes for varying number of PEs.

1. The test code `alltoall` is an example of using nonblocking communications.
2. The test code `IO_gather` illustrates gathering messages from all processes.
3. The test code `ring`, as the name suggests, simulates a ring architecture and sends a short message from one to the next PE, until the message comes back to the process which initiated the send-loop.

Exercise 3 (Testing the debugger - debugging) Follow the instructions at `https://www.uppmax.uu.se/support/user-guides/allinea-ddt-user-guide/` to get started with Arm DDT. It is also important to set the following environment variables:

```
OMPI_MCA_btl_openib_allow_ib=1
OMPI_MPIR_DO_NOT_WARN=1
```

Screen shots, illustrating where to do this, are included at the end of these instructions.
Consider the program provided in `codetodebug.c`, which asks you for a number. Each process is supposed to print the sum of its rank and this number, but a bug has sneaked into this code. Compile the code and run it on more than 1 core and study the output. Thereafter, use DDT to step through the code and see if you can find the bug. More information on the debugger can be found at `https://developer.nvidia.com/allinea-ddt`.

Exercise 4 (Testing the debugger - communication patterns) You have two parallel implementations of one and the same algorithm that solves numerically a one-dimensional wave equation. The corresponding codes are `wave-persistent.c`, using persistent communications, and `wave-parallel.c`, which uses other send-receive constructions. Compile the codes and run them one after another via the debugger. Concentrate on the communication part (lines 130-182, 304-305 in `wave-persistent` and lines 225-254, 320-321 in `wave-parallel`. From the

`Tools` menu choose `Message queue`. By stepping through the code and updating the system topology window, you are able to monitor the communications performed during the execution. You should be able to see the difference between persistent and `Isend-Irecv` communications.
Note: the codes use the so-called *derived data types*, however, these are not relevant for the exercise.

<u>Exercise</u> 5 (Testing the debugger - monitoring distributed data arrays) Consider the code `arrays.c`. It creates one ordinary (statically allocated) array and two three-dimensional arrays (one that is statically allocated and one that is dynamically allocated). Compile the program without optimizations. Set a breakpoint eg. on line 39 and run the code to the breakpoint. Now you will be able to monitor the distributed arrays and observe the values of the entries in the address space, local to each process: Use the `Multidimensional array viewer` from the `Tools` menu. Study the possibilities it offers - to evaluate array expressions, to slice and visualize the array, to change local values etc.
You are encouraged to extend the code by performing some arithmetic operations with the arrays and monitor the result. You may also change the size of the arrays by changing `N` in the c file.

<u>Exercise</u> 6 (Testing the performance analyser Arm MAP) Test Arm MAP, for instance using the code `wave-parallel.c` or some other example code, having in mind that the experiment should be sufficiently large.

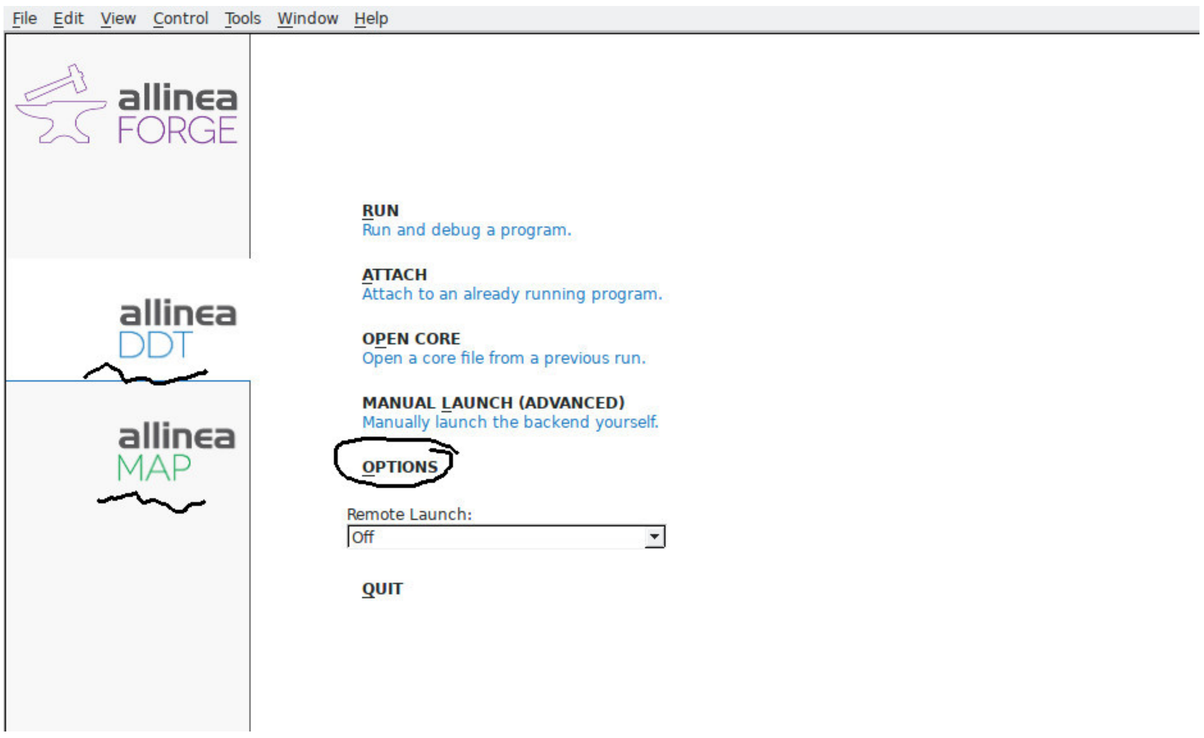
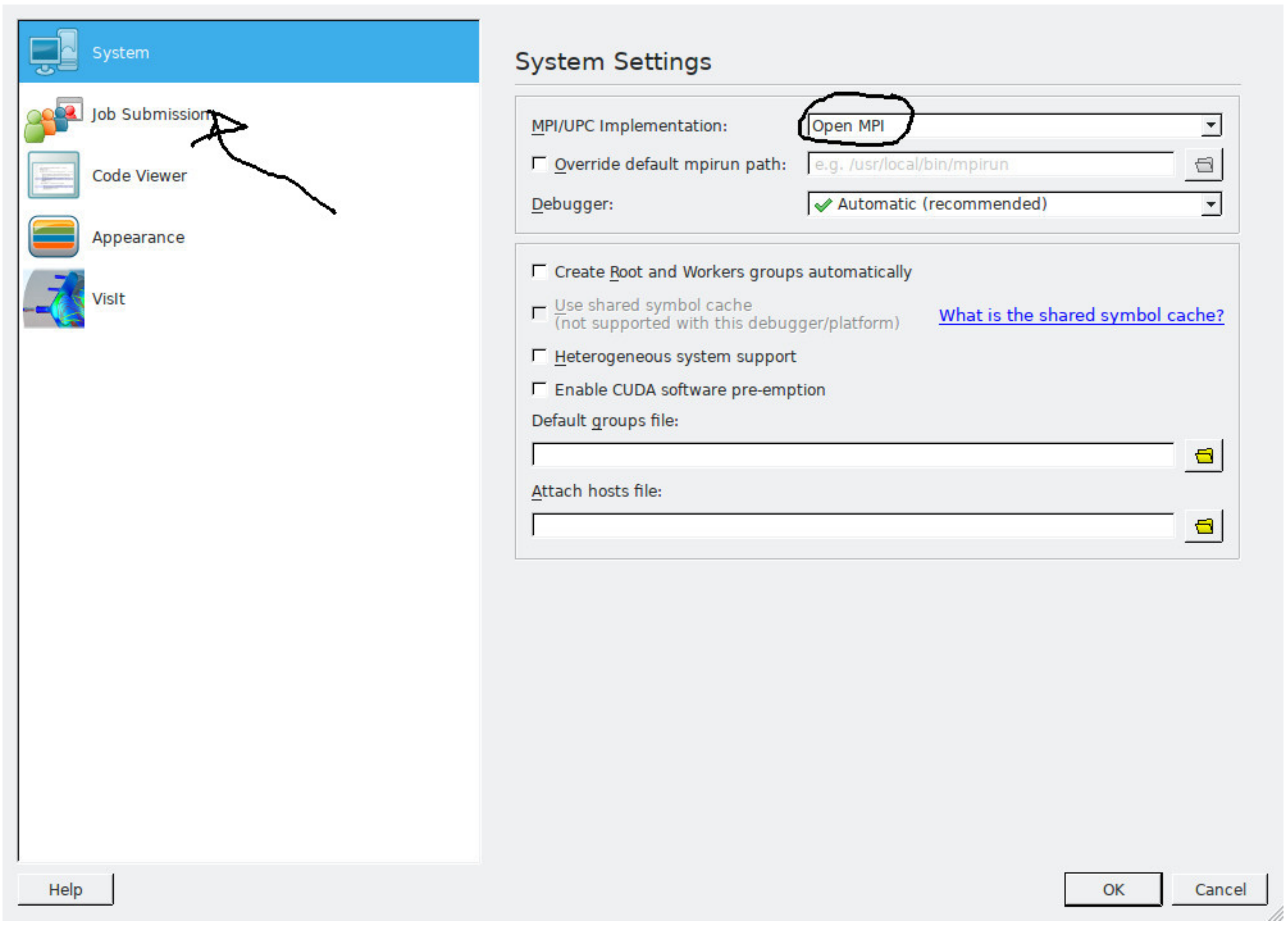

File Edit View Control Tools Window Help
allinea FORGE
allinea DDT
allinea MAP
RUN
Run and debug a program.
ATTACH
Attach to an already running program.
OPEN CORE
Open a core file from a previous run.
MANUAL LAUNCH (ADVANCED)
Manually launch the backend yourself.
OPTIONS
Remote Launch:
Off
QUIT

Figure 1: Arm (Allinea) first window, choose ’Options’

Screenshots illustrating settings for Arm DDT/MAP



System Settings

MPI/UPC Implementation: Open MPI

☐ Override default mpirun path: e.g. /usr/local/bin/mpirun

Debugger: ✓ Automatic (recommended)

☐ Create Root and Workers groups automatically

☐ Use shared symbol cache
(not supported with this debugger/platform)

What is the shared symbol cache?

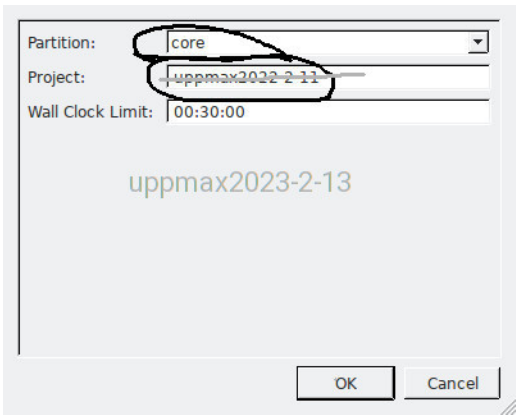
☐ Heterogeneous system support

☐ Enable CUDA software pre-emption

Default groups file:

Attach hosts file:

Figure 2: Arm (Allinea) settings, choose 'Job submission' − > 'Edit queue parameters'



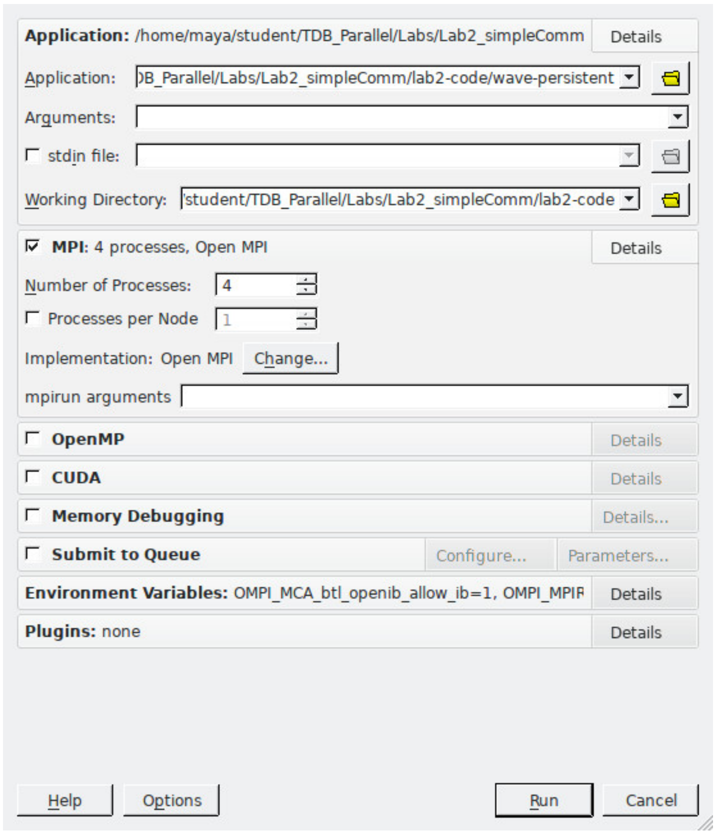
Partition: core

Project: ~~uppmax2022-2-11~~

Wall Clock Limit: 00:30:00

uppmax2023-2-13

Figure 3: Arm (Allinea) settings, choose 'core' and input project number!


Application: /home/maya/student/TDB_Parallel/Labs/Lab2_simpleComm

Application:)B_Parallel/Labs/Lab2_simpleComm/lab2-code/wave-persistent

Arguments:

stdin file:

Working Directory: student/TDB_Parallel/Labs/Lab2_simpleComm/lab2-code

☑ **MPI:** 4 processes, Open MPI

Number of Processes: 4

☐ Processes per Node 1

Implementation: Open MPI

mpirun arguments

☐ **OpenMP**

☐ **CUDA**

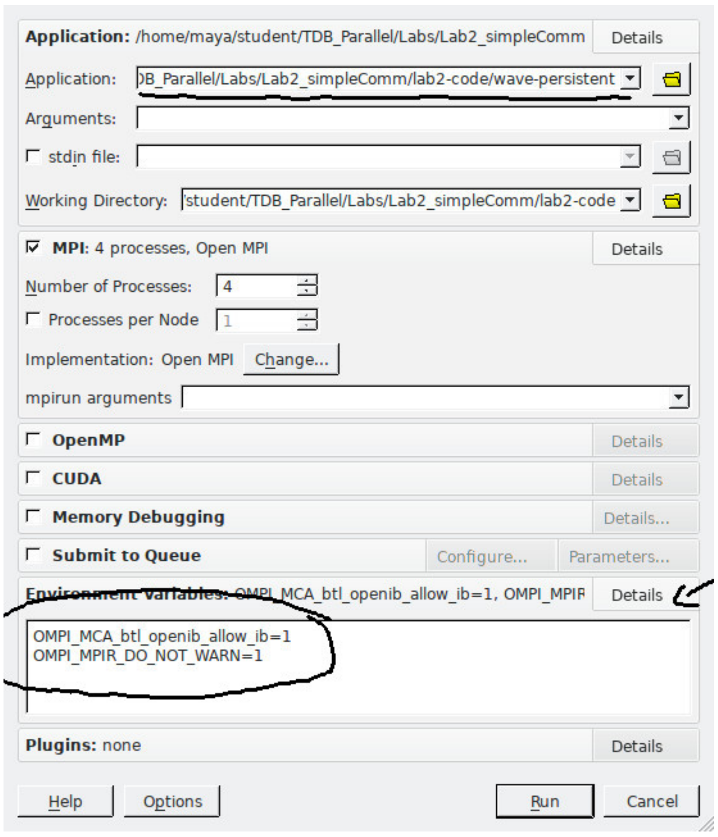
☐ **Memory Debugging**

☐ **Submit to Queue**

Environment Variables: OMPI_MCA_btl_openib_allow_ib=1, OMPI_MPIR

Plugins: none

Figure 4: Arm (Allinea) settings, click on 'Details'



Application: /home/maya/student/TDB_Parallel/Labs/Lab2_simpleComm

Application:)B_Parallel/Labs/Lab2_simpleComm/lab2-code/wave-persistent

Arguments:

☐ stdin file:

Working Directory: student/TDB_Parallel/Labs/Lab2_simpleComm/lab2-code

☑ **MPI:** 4 processes, Open MPI

Number of Processes: 4

☐ Processes per Node 1

Implementation: Open MPI

mpirun arguments

☐ **OpenMP**

☐ **CUDA**

☐ **Memory Debugging**

☐ **Submit to Queue**

Environment Variables: OMPI_MCA_btl_openib_allow_ib=1, OMPI_MPIR

```
OMPI_MCA_btl_openib_allow_ib=1
OMPI_MPIR_DO_NOT_WARN=1
```

Plugins: none

Figure 5: Arm (Allinea) settings