TP1: introduction to Grid'5000 and MPI

September 16, 2024

We are going to use:

- **Grid'5000**, our computing infrastructure.
- **OpenMPI**[1], an open source Message Passing Interface implementation. We can use it e.g. on several cores of a laptop, or on several machines of a lab ("TP") room, or on several nodes of a cluster of Grid'5000, etc.

We will stick with the *SPMD* programming model: a single executable for all processes, with branching depending on the "number" of the process (called its *rank*) allowing this process to execute its own tasks that might differ from those of other processes.

1 Grid'5000

At any hesitation, refer to the documentation on the website `https://www.grid5000.fr`, and in particular to the page "*Getting Started*" which gives very useful information.

1.1 SSH

To use Grid'5000, you need to have an account on the platform and to have uploaded your `ssh` public key through the web portal. You also need an `ssh` client on your own machine. `ssh` is widely used; documentation and help is easily found online.

If you don't already have one, you have to create a pair of `ssh` keys (public key, private key). On linux, this can be done via the command[2]:

```
$ ssh-keygen
```

It is good practice to protect these keys by a password (the key generation tool will ask for one; it is not advised to leave it empty). By default, the keys are stored in the folder `~/.ssh/`. It is of course the public part which must be uploaded to Grid'5000; the corresponding file has extension `.pub`. The secret part, which allows you to prove your identity, does not have an extension.

In case of problems, here is a troubleshooting checklist:

1. You have a folder[3] `~/.ssh/`
2. It contains two files `id_rsa` and `id_rsa.pub` (following a wrong manipulation, it is possible that they have been stored elsewhere).
3. You have uploaded the content of `id_rsa.pub` on the web interface of Grid'5000.

Make sure you fill the checklist on the course Moodle page at the same time as you are getting familiar with Grid'5000.

[1] `http://www.open-mpi.org/`

[2] The symbol `$` indicates the command prompt.

[3] Recall that in the shell, the `~` symbol denotes your home directory, just like the environment variable `$HOME`.

Exercice 1 – First steps with Grid'5000

1. Connect to the access point of Grid'5000:

   ```
   $ ssh login@access.grid5000.fr
   ```

   (of course replace `login` by your actual username!). If this does not work, either you have a problem with `ssh`, or your account has not yet been created (you should have received an email confirming the creation of your account).

2. Write a C program which prints "Hello World" and upload it to your home directory of the site of Nancy, via `ssh`, thanks to the `scp` utility:

   ```
   $ scp hello.c login@access.grid5000.fr:nancy/
   ```

   Then connect to the frontend node of the Nancy site, compile your program with `gcc`, and execute it (still on the frontend note: this should generally not be used for computing, but small tasks like compilation are accepted).

3. Tweak your `ssh` configuration as suggested by Grid'5000 documentation in order to facilitate access the different sites.

1.2 OAR

Grid'5000 resources are shared by all users. There is a *scheduler* that allows you to reserve access to the resources in a way that is respectful of other users.

Exercice 2 – Getting Started with OAR

Read the documentation of the scheduler `OAR` used on Grid'5000.

1. What is the command for using a node of any cluster for 2 hours, in interactive mode?
2. What is the command for: executing the program "Hello World" as soon as possible on a node of the `gros` cluster from Nancy, retrieving the output in a file named `toto.txt`, and sending an email to yourself when the *job* starts? (`man oarsub` is your friend...)
3. It can quickly become a headache to specify all these options each time. The "submission script" technique described at the bottom of the `oarsub` man page is very useful to automate this process. Write a submission script that does the same thing as before.
4. Obtain interactive access to 3 nodes of some cluster. Which nodes are these? If you connect to one of them, an environment variable named `$OAR_NODE_FILE` contains the name of a file which itself contains the list of nodes. We can thus print this file as follows:

   ```
   $ cat $OAR_NODE_FILE
   ```

   Alternately, one can retrieve the job number and ask the program `oarstat`. Use `oarsh` to connect to the nodes, and verify that this works.

5. If you start an interactive session and are somehow disconnected, your reservation stops and the resources are released. This can sometimes be a problem. Submit a job that runs the command `sleep infinity` on 4 nodes, reserved for 20 minutes, non-interactively. Once the job is running, connect to it using the command: The following shorter form works if you have a single running job:

   ```
   $ oarsub --connect 4860278   # <---- replace this by YOUR job number
   ```

   Alternatively, the following shorter form works if you have a single running job:

```
$ oarsub --connect
```

You can connect to your job several times and do whatever you want with it. Then, kill the job with `oardel`. Using `screen` or `tmux` is very practical in this context (but is outside the scope of this tutorial).

6. Reserve ONE node of any cluster, for 10 minutes from now. What is the command? Go and have a coffee and a short walk, and then verify that you can connect to your reserved node (with the command `oarsub --connect JOB_ID`).
7. Use the `funk` program to identify the next time at which you will be able to reserve 64 nodes of the cluster `gros` in Nancy for 4 hours, outside hours where the user policy applies.

   `funk` gives you the commande to do the reservation... DO NOT ACTUALLY DO THE RESERVATION! What is the command?

2 OpenMPI

OpenMPI is a set of programs which allow one to use **MPI** on a heterogeneous network of machines. It can be used just as well on a simple multicore machine as on very large clusters. There are packages to install OpenMPI on most linux distributions, (`apt install openmpi-bin` will work on Debian and its derivatives).

OpenMPI is already installed on all Grid'5000 machines.

2.1 Executing MPI programs at home

The MPI environment contains a program called `mpiexec` (or `mpirun`) which is used to run programs in parallel, i.e. run n copies of the same executable possibly on several machines.

By default, the command

```
$ mpiexec ./prog
```

detects the number of physical cores of your computer and runs one copy of `./prog` for each core. To verify that this works, we can for example run `mpiexec date` (`date` is a standard system utility that prints the date and time).

One can control the number of executed copies with the option `-n`. For example, to run two copies of `date` on your own computer:

```
$ mpiexec --n 2 date
```

If one indicates a number which is greater than the number of physical cores, recent versions of OpenMPI print a long message basically asking whether you are really sure of what you're doing.

2.2 Executing MPI programs on Grid'5000

The real strength of `mpiexec` is its ability to run the same program on several nodes of a cluster simultaneously. For this, one has to provide a file which contains a list of these nodes (this is the *host file* or *node file*).

Configuration on Grid'5000. OpenMPI uses `ssh` to run the executable on several machines. But on Grid'5000, by default `ssh` cannot be used as such. One has to first create a configuration file which tells OpenMPI to use `oarsh` instead of `ssh`. It is therefore **required** to create a folder `~/.openmpi` and then create a file named `~/.openmpi/mca-params.conf` whose content is:

```
plm_rsh_agent=oarsh
filem_rsh_agent=oarcp
```

Be careful, the file systems are individual to each site, so the above configuration has to be done on every used site. In principle, once this is done, when connecting to a site's frontend node the command

```
$ cat ~/.openmpi/mca-params.conf
```

should print the content of the file. If one forget to do this step, the most common symptom is an error message like `ORTE was unable to reliably start one or more daemons`. However, of course, do not do this on your own machine, otherwise OpenMPI will not work properly there anymore...

Use with OAR. On Grid'5000, after reserving two nodes and connecting to one of them, one can run:

```
$ mpiexec --hostfile $OAR_NODEFILE ./prog
```

This will execute `./prog` on each core of all reserved nodes. you can try to run the system utility `hostname` this way.

`mpiexec` has many, many options, and `man mpiexec` is your friend. Examples :

- Run *one* copy of `./prog` on each reserved node:

  ```
  $ mpiexec --map-by ppr:1:node --hostfile $OAR_NODEFILE ./prog
  ```

- Run *three* copies of `./prog`, but only one on each *processor* of each reserved node:

  ```
  $ mpiexec --n 3 --map-by ppr:1:socket --hostfile $OAR_NODEFILE ./prog
  ```

- The `--display-map` option allows one to show the distribution of the different MPI processes on the computing nodes. This distribution is made cyclic with respect to the node numbers thanks to the option `--map-by node`. Without this option `--map-by node`, OpenMPI detects the number of processes that each multicore node may execute in parallel, and first "fills" a node completely before considering the next node.

For the first practical sessions on MPI, you can first try your code at home on your own machine, and then see how to make this work on Grid'5000.

2.3 Compiling an MPI program

To compile the C program `toto.c`, one just needs to type the following command:

```
$ mpicc -o toto toto.c
```

Remark: `mpicc` is a *wrapper* that invokes `gcc` with the right options. In fact one can find these options out via the command `mpicc -showme`.

The obtained executable is then `toto`. It is ran in parallel thanks to the command `mpiexec`, as seen above.

2.4 Debugging MPI programs

To debug our MPI programs, we will mostly insert well-designed `printf` statements at well-chosen places (be careful with the order of printing between the different processes, see Section 3).

One may also use `gdb`: then the easiest way is to run all MPI processes locally on your machine, but this modifies the parallel execution and therefore will not always allow you to detect all bugs that appear in parallel runs.

For more information on debugging with **OpenMPI**, see:
`http://www.open-mpi.org/faq/?category=debugging#serial-debuggers`

2.5 Using high-performance networks on Grid'5000

OpenMPI has a system of software components (say "plugins") that auto-detects "special" network interfaces, and use them with higher priority. On Grid'5000, OpenMPI is compiled with native support for the high-performance network interfaces InfiniBand and OmniPath. Therefore, on clusters which only have Ethernet (TCP/IP) network, a warning message is printed:

```
A high-performance Open MPI point-to-point messaging module
was unable to find any relevant network interfaces:

Module: OpenFabrics (openib)
  Host: .....

Another transport will be used instead, although this may result in
lower performance.
```

This does not prevent the program from running. Note that `OpenFabrics (openib)` is a synonym of InfiniBand: the above message means that there is no InfiniBand network interface on the considered machine... and that it is a pity!

To avoid this warning message, it is possible to deactivate the software component "InfiniBand" with the option `--mca btl ^openib` (MCA = Modular Component Architecture, BTL = Byte Transport Layer, this deactivates the component `openib`). Similarly, one can deactivate the "normal" TCP/IP network with the option `--mca btl ^tcp` (but be careful, then there MUST be some network of another type).

On the other hand, one can reserve nodes that have high-performance networks. Several clusters have InfiniBand. In OAR, this can be done through the option `--property "ib_rate=56"` to reserve nodes that have an InfiniBand network at 56Gbit/s. Similarly, `--property "opa_rate=100"` reserves nodes that have OmniPath at 100Gbit/s (only `dahu` and `yeti` at Grenoble will be available to you).

3 TP

To start with, you can get the MPI documentation in `pdf` format at the following address: `http://www.mpi-forum.org/`. For this first TP, we will manipulate the functions `Send` and `Recv`.

Exercice 3

What will printed by the following program? Verify this by compiling it, and executing it with **OpenMPI**.

```c
#include <stdio.h>
#include <mpi.h>

int main(int argc, char* argv[])
{
    int rank, p, value, tag = 10;
    MPI_Status status;

    /*  Initialisation */
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &p);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    if (rank == 1) {
        value = 18;
        MPI_Send(&value, 1, MPI_INT, 0, tag, MPI_COMM_WORLD);
    } else if (rank == 0) {
        MPI_Recv(&value, 1, MPI_INT, 1, tag, MPI_COMM_WORLD, &status);
        printf("I have received the value %d from the process of rank 1.\n", value);
```

```
    }

    MPI_Finalize();
}
```

Exercice 4

Here is now a slightly more complex program:

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

#define SIZE_H_N 50

int main(int argc, char* argv[])
{
    int         my_rank;      /* rank of the process  */
    int         p;            /* number of processes  */
    int         source;       /* rank of the source   */
    int         dest;         /* rank of the receiver */
    int         tag = 0;      /* tag of the message   */
    char        message[100];
    MPI_Status  status;
    char hostname[SIZE_H_N];

    gethostname(hostname, SIZE_H_N);

    /* Initialisation  */
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
    MPI_Comm_size(MPI_COMM_WORLD, &p);

    if (my_rank != 0) {
        /* Creation du message */
        sprintf(message, "Coucou du processus #%d depuis %s!",
                my_rank, hostname);
        dest = 0;

        MPI_Send(message, strlen(message)+1, MPI_CHAR, dest, tag, MPI_COMM_WORLD);
    } else {
        for (source = 1; source < p; source++) {
            MPI_Recv(message, 100, MPI_CHAR, source, tag, MPI_COMM_WORLD, &status);
            printf("Sur %s, le processus #%d a recu le message : %s\n",
                   hostname, my_rank, message);
        }
    }

    MPI_Finalize();
}
```

1. Run this program several times with the same number of processes; then run it several times with different numbers of processes. What do you observe?
2. Now, insert some `printf` at many places in the program. What do you observe?
3. Replace the variable `source` in `MPI_Recv` by the identifier `MPI_ANY_SOURCE`. Run the program several times. What do you observe? Explain.

4. Write a program in which each process sends a string to its successor (the process $rank+1$ if $rank < p-1$, the process 0 otherwise), and receives a message from its predecessor. Once your program works, replace `MPI_Send` by `MPI_Ssend`. What do you observe? In what follows we will call this program `ex_ssend.c` .
5. Copy `ex_ssend.c` into `ex_ssend_corrected.c` . While keeping `MPI_Ssend`, change the algorithm so that the process 0 sends first its message to process 1, which will only send its message to process 2 after having received the one from 0; then the process 2 will only send its message to process 3 after having received the one from 1; etc.
6. Copy `ex_ssend.c` into `ex_send.c` . Replace `MPI_Ssend` by `MPI_Send`, and make the size of the data sent through `MPI_Send` vary up to 100 kB. What do you observe?

Exercice 5 – Token passing (Bonus)

Considering the previous exercise, modify the program so that a token (say, an integer) is transmitted on a ring of processes ($0 \to 1 \to 2 \to \cdots \to p-1 \to 0 \to 1 \dots$). The number of rounds must be a modifiable parameter, and your program must print the time that was used for this operation.

Execute this, during the night, on different Grid'5000 cluster which have different networks (OmniPath, InfiniBand, Ethernet, etc.) and measure the network *latency*. For example, `sagitaire` in Lyon possesses a cheap network (1Gbit/s Ethernet), whereas `dahu` in Grenoble has 100Gbit/s Omnipath; `grimoire` and `gros`, in Nancy, are in between.

Exercice 6 – Network Speed Benchmark (Bonus)

Write an MPI program which measures the *bandwidth* of the network, and test it on a few clusters of Grid'5000 (two nodes are enough).

For this, you can simply do a ping-pong between two nodes with big data volume. Be careful to send a single big message, rather than many small messages.

Exercice 7 – Network Torture Test (Bonus)

Write an MPI program which performs an `MPI_Alltoall` as big as possible (in terms of volume and number of nodes), and test it on a few clusters of Grid'5000.