CSC5001

Fiche 4 : Introduction to MPI

1 Bootstrapping MPI

Launching MPI requires that the nodes participating in the calculation be directly accessible (i.e., without entering a password) via SSH. If you do not have any operational SSH keys, here are the key steps to follow before doing the labs. You need two ssh keys, one on your laptop and another one on your TSP account.

1.1 Key generation

To generate your ssh key, use the following command protected by a passphrase :

```
$ ssh-keygen
# creates ~/.ssh/id_rsa.pub (public key), and ~/.ssh/id_rsa (private key, DO NOT DISCLOSE !)
```

1.2 Installing your ssh keys

Copy the public part of your both ssh key on TSP server using the following command on your laptop and from a TSP machine :

```
$ ssh-copy-id my_login@ssh1.it-sudparis.eu
```

From a labs machine, you should be able to connect to the labs machine by running ssh 3a401-13 using your TSP ssh key (without typing your password). From your laptop, since you are still outside the TSP network, you should be able to connect to the TSP gateway only by running ssh ssh1.it-sudparis.eu using your laptop ssh key (without typing your password).

1.3 Forwarding ssh to reach labs machines through TSP gateway from your laptop

In order to be able to connect to the labs machine from your laptop by running ssh 3a401-13, edit /.ssh/config and add the following lines :

```
ForwardAgent yes

Host tsp
user my_login
hostname ssh1.it-sudparis.eu

Host 3a401*
user my_login
ProxyJump tsp
```

For now, each time you connect to a new machine, ssh asks you to check its fingerprint. You can automatically accept/update these fingerprints by running :

```
~trahay_f/opt/renew_ssh_fingerprints.sh
```

You should now be able to connect to the labs machine from your laptop by running ssh 3a401-13 using your ssh key (without typing your password).

2 Hello world with MPI

 The aim of this exercise is to discover the compilation and deployment chain of MPI.

2.1 Basic parallel hello

Implement a MPI program <code>hello.c</code> where each MPI process you launch prints a message like :

```
Hello from task <rank> / <nb tasks> on  <machine>.
```

For this purpose : <ul> Initialize MPI, Retrieve the rank of the current MPI process, Retrieve the number of running MPI proccesses, Retrieve the hostname on which the current MPI process is running on (cf <code>man gethostname</code>), Close MPI.

On labs machine, OpenMPI is installed by default but in the labs, we use MPICH implementation you can load thanks to the following script (you have to execute it each time you open a new terminal) :

```
source /netfs/inf/trahay_f/opt/asr5_env.sh
```

Compile your code thanks to MPI compiler mpicc and execute your program on one and several machines. The `hosts` file indicates the machines on which computation will be deployed by `mpirun`, transparently through ssh (hence the importance of configuring your SSH keys).

```
$ mpicc hello.c -o hello
$ mpirun -f hosts  -np 4 ./hello1
Hello World from task 2 / 4 on b313-03
Hello World from task 3 / 4 on b313-04
Hello World from task 1 / 4 on b313-02
Hello World from task 0 / 4 on b313-01
```

2.2 Ordered parallel hello

Modify your program in order to have orderded printings (ie. rank 0 first, followed by rank 1, rank2, etc.). For this purpose, each process <code>N</code> waits that the process of rank

<code>N-1</code> printed its message by waiting a message from it, print its own message and send a message to the process <code>N+1</code> in order to notify it is its turn. Process 0 initiates the chain by not doing any reception but printing its message directly and sending the token to rank 1.

3 Mandelbrot with MPI

EASYPAP makes it easy to use MPI by relieving you of the need to call `MPI_Init()` and `MPI_Finalize()`, `Makefile`, and `mpirun`. To pass parameters to the `mpirun` command, use the `-mpi` EASYPAP option followed by a string of characters. You can use the `-d M` debugging option to display a window for each process : `./run -k mandel -v variante_mpi -mpi "-np 4" -d M`

3.1 Baseline MPI version

In this first version, all processes participate in the computation. At the end, the master process receives the data calculated by the other processes to perform the display. The calculation is performed in the procedure `mandel_compute_mpi()`. Communications are executed by the function `mandel_refresh_img_mpi()` which is automatically called by EASYPAP before each display.

Integrate the file `mandel-mpi-skeleton.c` into the file `mandel.c` and complete the latter. The program can be initially tested without completing the function `mandel_refresh_img_mpi()`. The processes then calculate their image area independently, without synchronization. It can be visually verified that each process correctly calculates its pixel area thanks to the option `-d M`.

```
1  static int rank, size;
2
3  void mandel_init_mpi ()
4  {
5    easypap_check_mpi (); // check if MPI was correctly configured
6
7   // TODO obtenir rank et size ;
8
9    mandel_init ();
10 }
11
12 static int rankTop (int rank)
13 {
14   return 0; // indice de la 1ere ligne du processus rank
15 }
16
17 static int rankSize (int rank)
18 {
19   return 0; // nombre de lignes à traiter par le processus rank
20 }
21
22 void mandel_refresh_img_mpi ()
23 {
24   // le maitre réceptionne les données
25   // les autres processus envoient les données
26   // &cur_img(ligne,0) correspond à l'adresse d'une ligne du tableau.
27 }
```

```
28
29 ////////// MPI basic variant
30 // Suggested cmdline:
31 // ./run -k mandel -v mpi -mpi "-np 4"  -d M
32
33 unsigned mandel_compute_mpi (unsigned nb_iter)
34 {
35   for (unsigned it = 1; it <= nb_iter; it++) {
36     do_tile (0, rankTop (rank), DIM, rankSize (rank));
37     zoom ();
38   }
39   return 0;
40 }
41 #endif
```

3.2 Point-to-point communication

To implement `mandel_refresh_img_mpi ()`, use point-to-point communication. The master receives data contribution from all other ranks while all the others send their data to the master.

3.3 Communications collectives

Adapt the function `mandel_refresh_img_mpi ()`. by replacing point-to-point communications with collective communications (here gather). Compare execution times obtained with those obtained by the point-to-point version. Refine the measurements by timing the transmission times (distribution and collection phases) with `gettimeofday` function.

3.4 Hybrid MPI + OpenMP version

Using a tiled version parallelized with OpenMP, write a function `mandel_compute_mpi_omp()`. You will also need to write the functions `mandel_init_mpi_omp()` and `mandel_refresh_img_mpi_omp()`.