Practical sessions HPC

Hrachya Astsatryan , Hélène Hénon

June 19, 2024

Plan

Access HPC systems

1. Download the file Private.pem in your files
2. `chmod 600 Private.pem` in order to change the permissions
3. `ssh -i Private.pem ubuntu@185.127.66.38` to access the cluster
4. Create a directory for your work in the `students` directory

Job Submission and Control using SLURM

Task 1

- `sinfo -o "%P"`

Job Submission and Control using SLURM

Task 1

- `sinfo -o "%P"`
  - This command fetches information about the partitions (also known as queues) available on the cluster.
  - -o "%P" specifies the output format to only display the names of the partitions.

Job Submission and Control using SLURM

Task 1

- `sinfo -o "%P"`
  - This command fetches information about the partitions (also known as queues) available on the cluster.
  - -o "%P" specifies the output format to only display the names of the partitions.
- `sinfo -o "%N"`

Job Submission and Control using SLURM

Task 1

- `sinfo -o "%P"`
  - This command fetches information about the partitions (also known as queues) available on the cluster.
  - -o "%P" specifies the output format to only display the names of the partitions.
- `sinfo -o "%N"`
  - This command retrieves information about the number of nodes available in each partition.
  - -o "%N" specifies the output format to display only the number of nodes.

Job Submission and Control using SLURM

Task 1

- `sinfo -o "%P %T"`

Job Submission and Control using SLURM

Task 1

- `sinfo -o "%P %T"`
    - This command provides information about the state of each partition.
    - -o "%P %T" sets the output format to display both the partition name and its state.

Job Submission and Control using SLURM

Task 1

- `sinfo -o "%P %T"`
  - This command provides information about the state of each partition.
  - -o "%P %T" sets the output format to display both the partition name and its state.
- `sinfo -o "%c"`

Job Submission and Control using SLURM

Task 1

- `sinfo -o "%P %T"`
  - This command provides information about the state of each partition.
  - -o "%P %T" sets the output format to display both the partition name and its state.
- `sinfo -o "%c"`
  - This command retrieves information about the number of CPUs in each partition.
  - -o "%c" specifies the output format to display the total number of CPUs in each partition.

Job Submission and Control using SLURM

Task 1

- `sinfo -o "%m"`

Job Submission and Control using SLURM

Task 1

- `sinfo -o "%m"`
  - This command fetches information about the available memory resources in each partition.
  - -o "%m" sets the output format to display only memory-related information.
  - -MEMORY indicates that we're specifically querying memory resources.

Job Submission and Control using SLURM

Task 1

- `sinfo -o "%m"`
  - This command fetches information about the available memory resources in each partition.
  - -o "%m" sets the output format to display only memory-related information.
  - -MEMORY indicates that we're specifically querying memory resources.
- `sinfo --Node`

Job Submission and Control using SLURM

Task 1

- `sinfo -o "%m"`
  - This command fetches information about the available memory resources in each partition.
  - -o "%m" sets the output format to display only memory-related information.
  - -MEMORY indicates that we're specifically querying memory resources.
- `sinfo --Node`
  - This command fetches detailed information about the nodes in the cluster.
  - –Node is an option specifying that we're interested in node-related information.
  - -long requests a long format output, providing detailed information about the nodes.

Job Submission and Control using SLURM

Task 1

- `sinfo -N -h -O NODELIST`

Job Submission and Control using SLURM

Task 1

- `sinfo -N -h -O NODELIST`
  - This command retrieves a list of nodes in the cluster.
  - -N specifies that we're only interested in node-related information.
  - -h omits the header from the output.
  - -O NODELIST specifies the output format to display only the list of nodes.

Job Submission and Control using SLURM

Task 1

`scontrol show nodes` :

- used to display detailed information about the nodes in the Slurm cluster
- provides information such as the node name, state, CPUs, sockets, cores per socket, threads per core, memory, node features, and other relevant attributes.

.

Job Submission and Control using SLURM

Task 1

`scontrol show job` **followed by the job ID** :

- used to display detailed information about a specific job in the Slurm workload manager
- provides information about the job's state, resources requested, job ID, job name, submission time, queue name, user ID, etc...

.

Job Submission and Control using SLURM

Task 2

```
srun --pty --nodes=1 --cpus-per-task=4 --time=1:00:00 bash
-i
```

⇒ Slurm will schedule the job, find an available node that matches the resource request

- **Resource Allocation**: It requests one node with *4 CPU cores*.
- **Time Limit**: It sets a *maximum runtime of 1 hour* for the session.
- **Interactive Shell**: It opens an *interactive bash shell* on the allocated node.

Job Submission and Control using SLURM

Task 2

```
srun -n1 sleep 1
```

⇒ Slurm will allocate resources for **one task**, which is to simply **pause for 1 second**

Job Submission and Control using SLURM

Task 3

- `squeue` : view information about jobs located in the Slurm scheduling queue
- `sacct` : displays accounting data for all jobs and job steps in the Slurm job accounting log or Slurm database
- `scontrol` : view or modify Slurm configuration and state.

OpenMP

- an application programming interface (API) that supports multi-platform shared-memory multiprocessing programming

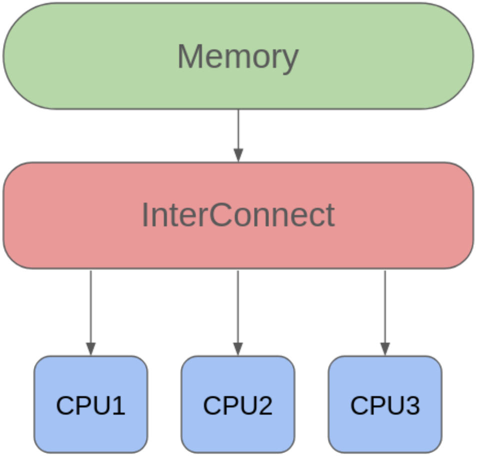


Figure 1: Shared memory programming

OpenMP

Task 1 : bubble sort

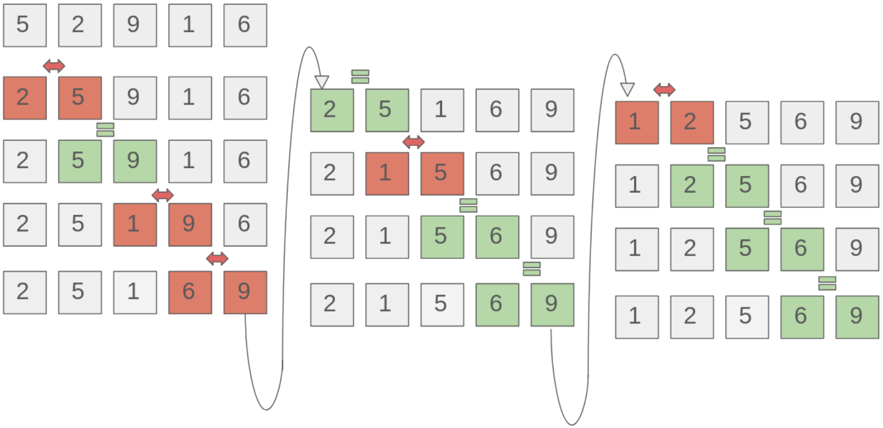


Figure 2: Bubble sort algorithm

OpenMP

Task 1 : bubble sort odd-even

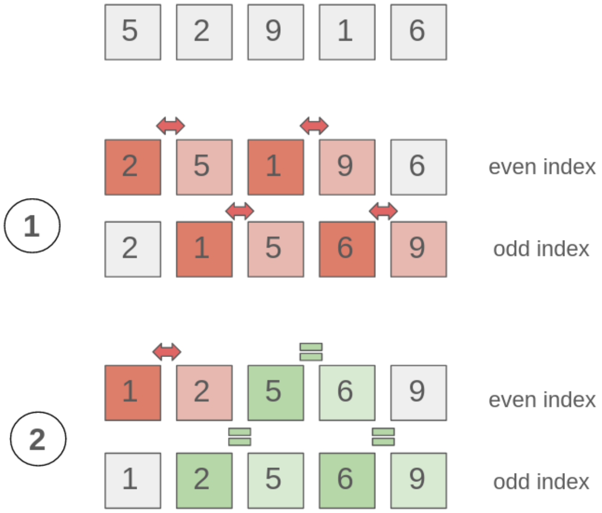
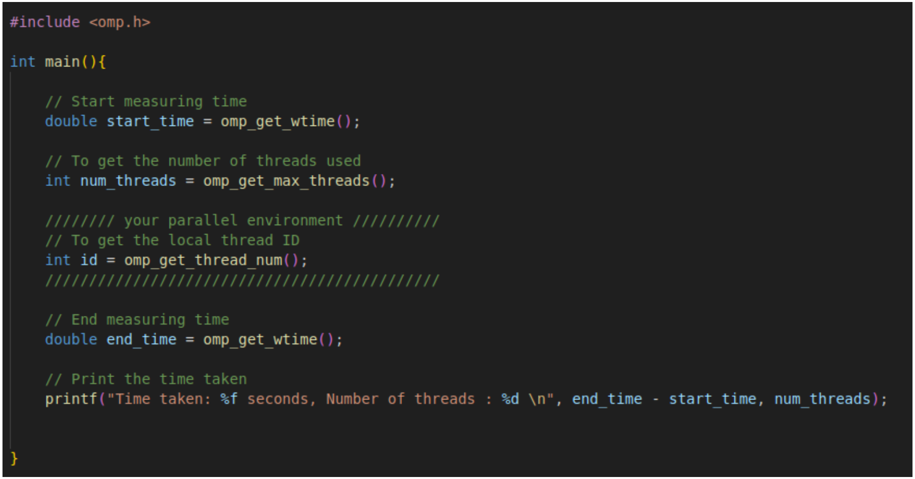


Figure 3: Bubble sort odd-even algorithm

OpenMP
Useful commands



```c
#include <omp.h>

int main(){

    // Start measuring time
    double start_time = omp_get_wtime();

    // To get the number of threads used
    int num_threads = omp_get_max_threads();

    //////// your parallel environment //////////
    // To get the local thread ID
    int id = omp_get_thread_num();
    /////////////////////////////////////////////

    // End measuring time
    double end_time = omp_get_wtime();

    // Print the time taken
    printf("Time taken: %f seconds, Number of threads : %d \n", end_time - start_time, num_threads);


}
```

Figure 4: Some useful commands

OpenMP

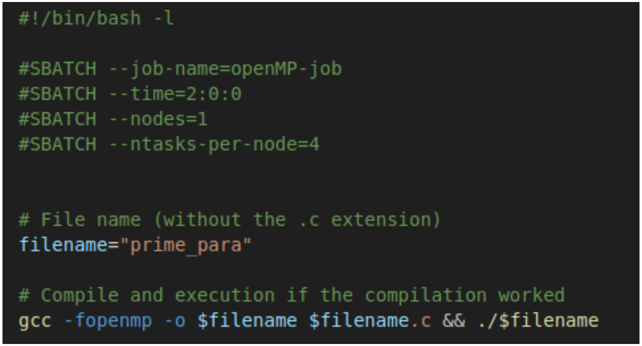
How to compile

1. Choose how many threads you want with the command in the shell : `export OMP_NUM_THREADS=n`
2. Compile your file with `gcc -fopenmp -o exec_name file_name.c`
3. Execute it with `./exec_name`

OpenMP

How to compile

OR with a bash script (better because you won't have to use slurm command to allocate the resources you need)



```bash
#!/bin/bash -l

#SBATCH --job-name=openMP-job
#SBATCH --time=2:0:0
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=4


# File name (without the .c extension)
filename="prime_para"

# Compile and execution if the compilation worked
gcc -fopenmp -o $filename $filename.c && ./$filename
```

Figure 5: Bash script for OpenMP

MPI

Message Passing Interface (MPI) is a standardized and portable message-passing standard

⇒ Contrary to OpenMP, the memory is not shared between the threads

⇒ the threads communicate with each others by sending **messages**

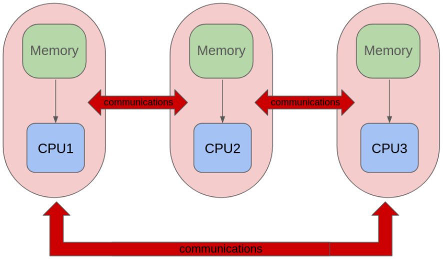


Figure 6: Distributed memory programming

MPI
Load

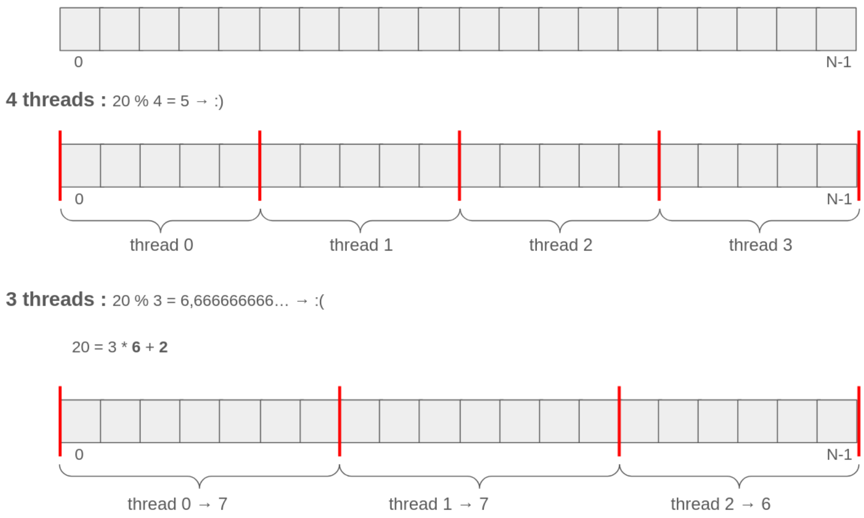
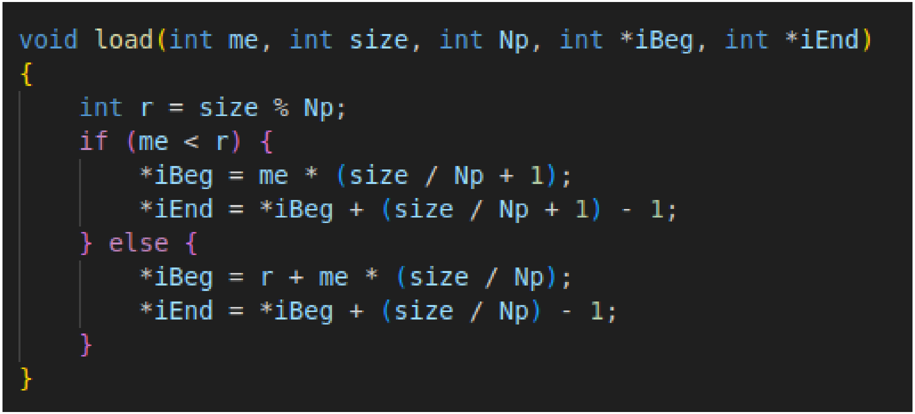


Figure 7: Load principle

MPI

Load



```
void load(int me, int size, int Np, int *iBeg, int *iEnd)
{
    int r = size % Np;
    if (me < r) {
        *iBeg = me * (size / Np + 1);
        *iEnd = *iBeg + (size / Np + 1) - 1;
    } else {
        *iBeg = r + me * (size / Np);
        *iEnd = *iBeg + (size / Np) - 1;
    }
}
```

Figure 8: Load code

MPI

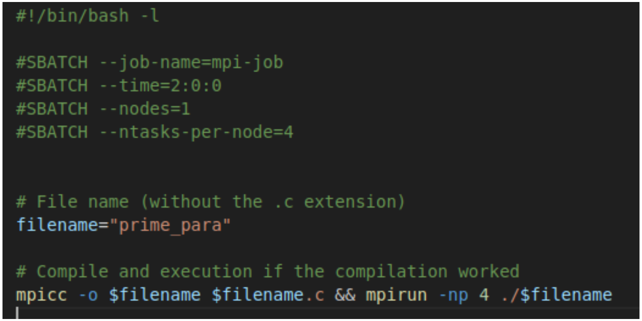
How to compile

1. Compile your file with `mpicc -o filename filename.c`
2. Execute it with `mpirun -np <number of process> ./filename`

MPI

How to compile

OR with a bash script (better because you won't have to use slurm command to allocate the resources you need)

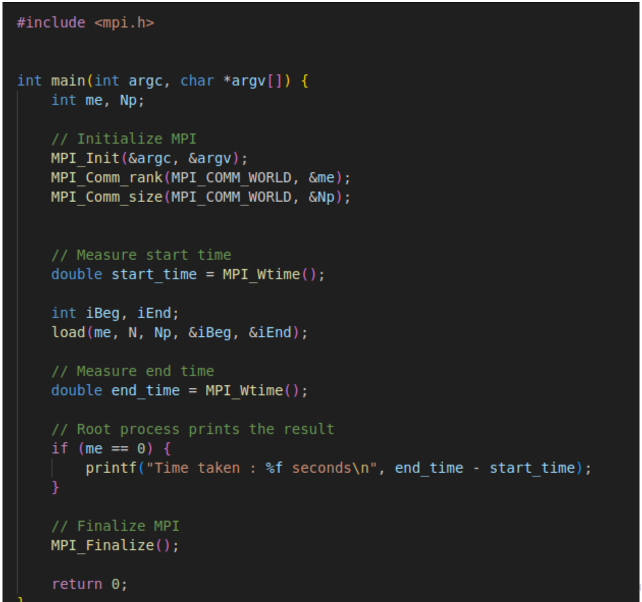


```
#!/bin/bash -l

#SBATCH --job-name=mpi-job
#SBATCH --time=2:0:0
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=4


# File name (without the .c extension)
filename="prime_para"

# Compile and execution if the compilation worked
mpicc -o $filename $filename.c && mpirun -np 4 ./$filename
```

Figure 9: Bash script for MPI

MPI

Useful commands



```c
#include <mpi.h>


int main(int argc, char *argv[]) {
    int me, Np;

    // Initialize MPI
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &me);
    MPI_Comm_size(MPI_COMM_WORLD, &Np);


    // Measure start time
    double start_time = MPI_Wtime();

    int iBeg, iEnd;
    load(me, N, Np, &iBeg, &iEnd);

    // Measure end time
    double end_time = MPI_Wtime();

    // Root process prints the result
    if (me == 0) {
        printf("Time taken : %f seconds\n", end_time - start_time);
    }

    // Finalize MPI
    MPI_Finalize();

    return 0;
```