HPC101 – Hands-on Session

I. Navigating, Writing Jobs, Submission, and Monitoring

1. Download a repository as `.zip` file from `https://github.com/IITDhTraining/hpc101` and copy it to your home drive on the master node.
2. Log-in to your cluster.
3. Move the file copied from home drive to scratch space on your cluster.
4. Go to the directory where you have moved the file and extract the files from the `.zip` file that you have copied.
5. Go to the directory `hpc101` (case-sensitive).
6. Write and submit a job script, `procinfo.job`, to find processor info (using the command `cat /proc/cpuinfo`) of the compute node on which you run the job. Your job should request 1 node and 1 core for a maximum time of 15 seconds. Rename the output file produced as `cpuinfo_compute.out`
    a. How many physical cores are there in your compute node?
    b. How many cores are there including hyperthreading?
7. Now find the processor info on the master node. Capture the output in a file called `cpuinfo_master.out`. Compare `cpuinfo_compute.out` and `cpuinfo_master.out` and note down your observations.
    a. How many physical cores are there in your master node?
    b. How many cores are there including hyperthreading?
8. Write and submit job script, `wordcount.job`, to sum the count of the number of words in each of the `.txt` files extracted (hint: use the commands `wc, cut, and bc. Use pipes, backtick, and variables in shell`). Your job should request 1 node and 1 core for a maximum time of 5 minutes.
    a. Report the job status of the job submitted. When the job completes, what is the status that you see? rename the output file produced as `wordcount.out`
9. Write and submit a job script, `hostname.job`, to find the name of the compute nodes (using `hostname` command) on which you run the job. Your job should request 4 nodes and 1 core on each node. The job must run on all 4 nodes within a maximum time of 15 seconds. Rename the output file produced as `hostname.out`

II. Programming

1. Parallelize computing the value of *pi* by numerical integration using the *Reimann sums* approach:

Let $f(x) = \sqrt{1 - x^2}$ describe the quarter circle for x = 0 … 1

$pi / 4 = \sum_{i=0}^{N-1} \Delta x \, f(x_i)$ where $x_i = i \, \Delta x$ and $\Delta x = 1/N$

Sequential version of the program (`pi_seq.cpp`) is given to you. Write an OMP parallel program to do the same. Name your file as `pi_omp.cpp`

Background: You can compile your program as:

```
g++ -fopenmp pi_omp.cpp -o pi_omp
```

If the compilation is successful, you will see an executable file called `pi_omp`. Create jobs `pi_omp_1.job, pi_omp_2.job`. that run the program with 1 and 2 threads respectively on 1 node. Similarly, create jobs to run your program with 4, 8, 16, 32, and 64 threads on 1 node. Does the speedup (runtime compared to 1 thread run) you see match your expectations considering the processor configuration of your machine?

To run your program, you can type: `./pi_omp` on your terminal. **However, do not run your program as mentioned previously on your terminal.** This is because you would be launching the executable on master node!. You should launch your executable on compute nodes. Instead, you should be using the above command in your .job files.

2. Write an MPI parallel program to parallelize computing the value of pi using Reimann sums approach. You have been given `pi_mpi.cpp.` You need to fill in a line of code.

a. Use Reductions in MPI to implement your program.

Background: MPI parallel programs can be written in C/C++/Fortran (and in Python as well). You compile a C/C++ program having MPI constructs as:

```
mpicc pi_mpi.c -o pi_mpi
mpic++ pi_mpi.cpp -o pi_mpi
```

If the compilation is successful, you will see an executable file called `pi_mpi`. Create jobs `pi_mpi_1.job, pi_mpi_2.job`. that run the program with 1 and 2 processes respectively on 1 node. Similarly, create jobs to run your program with 4, 8, 16, 32, and 64 processes on 1 node. Does the speedup (runtime compared to 1 process run) you see match your expectations considering the processor configuration of your machine? How do the runtimes compare to omp runs?

To create 10 copies of your executable and run them, you would use the command: `mpirun -np 10 pi_mpi` On your terminal.

However, **do not run your program as mentioned above on your terminal.** This is because you would be launching the executable on master node!. You should launch your executable on compute nodes. Instead, you should be using the above command in your .job files.