Bhaven Patel

5. Distributed-Memory Parallel Processing

5.2. Hybrid Parallel Processing

I am using two t2.2xlarge instances on AWS for this question. These instances are using an Intel(R) Xeon(R) CPU E5-2686 v4 @ 2.30GHz with 4 CPUs. These were the same instances that we created during the lab session for MPI. I ran the code using multiple processes as specified.

I compiled the *jacobi1d_mpi_hybrid.c* (P52.c) code using "mpicc -fopenmp -DUSE_CLOCK jacobi1d_mpi_hybrid.c -o jacobi1d_mpi_hybrid".

1. Single MPI task launched per node
   I used the following run command to run 1 task/process per node with 4 threads each:
   ```
   mpirun -np 2 -genv OMP_NUM_THREADS=4 -hosts master,node1
   ./jacobi1d_mpi_hybrid 100000000 100
   ```

Elapsed time: 6.852s

2. Single MPI task launched on each socket
   I used the following run command to run 2 tasks/processes per node with 2 threads each:
   ```
   mpirun -np 4 -genv OMP_NUM_THREADS=2 -hosts master,node1
   ./jacobi1d_mpi_hybrid 100000000 100
   ```

Elapsed time: 6.916s

3. Each core on a node is assigned an MPI task (all MPI)
   I used the following run command to run 4 tasks/processes per node with 1 thread each: `mpirun -np 8 -genv OMP_NUM_THREADS=1 -hosts master,node1 ./jacobi1d_mpi_hybrid 100000000 100`

Elapsed time: 6.812s

We find that all of the methods are about equivalent in terms of performance because each method is essentially utilizing 8 threads/cores for execution; the overhead costs from MPI and OpenMP are small compared to the decrease in execution time for this problem-size of 10^8. In SMP nodes run configuration, we lean heavier on OpenMP to run our parallel code because we have one process per node and 4 threads per process, which gives us 8-fold parallelization. In the SMP Sockets run configuration, we lean equally on MPI and OpenMP to run our parallel code because we have 2 processes per node and 2 threads per process, which again is 8-fold parallelization. Lastly with the all MPI run configuration, we lean heavily on MPI because we have 4 processes per node and 1 thread per process, resulting in 8-fold parallelization. Thus, we see equivalent results for all three methods.