EE 451: Parallel and Distributed Computation

PA4 — Spring 2021

Due date: Sunday 14th March 2021 11:59 PM

1. **Examples**

   Copy example files to your home directory.

   1. Login to HPC
   2. Copy

      ```
      cp -r /project/xuehaiqi_652/examples .
      ```

   3. Goto `examples`

      ```
      cd examples
      ```

   The `openmp_example.c` contains the OpenMP implementation of matrix vector multiplication.

   1. Login to HPC
   2. Compile

      ```
      gcc -O3 -fopenmp openmp_example.c
      ```

   3. Run

      ```
      srun -c8 ./a.out
      ```

      The option `-c` specifies the number of CPUs allocated for a task. By default, the value is 1. For OpenMP program, the number of threads should equal the number of CPUs.

   The `mpi_examples` folder includes the source codes used in discussions. To run an mpi program, for example, the 'scatter.c', follow the steps:

   1. Login to HPC
   2. Setup MPI toolchain:

      ```
      module purge
      module load intel/19.0.4 intel-mpi
      ```

   3. Compile

      ```
      mpicc -O3 scatter.c
      ```

   4. Run

      ```
      srun --exclusive --mpi=pmi2 -n4 ./a.out
      ```

The option `-n` specifies the number of tasks (processes). By default, the value is 1. There is 1 task per node (machine), but note that the `-c` option will change this default.

2. **Parallel Matrix Multiplication**[**50 points**]

   Parallelize the **naive** matrix multiplication which you implemented in PA 1 using OpenMP. Name the program as `openmp.c`. Take a screenshot.

   - The matrix size is $4K \times 4K$. Print out the execution time and the value of $C[100][100]$ in your program.
   - Pass the number of threads p as a command line parameter [1].
   - Report the execution time for $p = 1$, 2, 4.

3. **Pass Message in a Ring** [**50 points**]

   Write an MPI program that passes a value around 4 processes using the following steps. Name this program as `mpi.c`. Take a screenshot.

   1. Process 0 initializes $Msg = 451$ and prints value of Msg
   2. Process 0 sends the value of Msg to Process 1
   3. Process 1 receives the value of Msg, increases it by 1, prints the value and sends the current value of Msg to Process 2
   4. Process 2 receives the value of Msg, increases it by 1, prints the value and sends the current value of Msg to Process 3
   5. Process 3 receives the value of Msg, increases it by 1, prints the value and sends the current value of Msg to Process 0
   6. Process 0 receives the value of Msg from Process 3 and prints the value

   The output messages look like (Note that your code is still correct if the order of messages is different):

   - Process 0: Initially $Msg = 451$
   - Process 1: $Msg = 452$
   - Process 3: $Msg = 454$
   - Process 2: $Msg = 453$
   - Process 0: Received $Msg = 454$. Done!