CS 4175 Fall 2023
Shirley Moore, Instructor
Programming Assignment 3
50 points

Data Decomposition and Task Decomposition Using OpenMP

Please use Bridges-2 for this assignment. If you have trouble getting interactive access to a GPU-shared node using interact, please submit batch jobs instead.

1. For our data decomposition example, we will parallelize Jacobi iteration using the OpenMP **parallel** and **for** directives.

   a. (10 pts) You are provided with a sequential Jacobi iteration code in **jacobi.c**. Compile the code by typing

   ```
   gcc -fopenmp -o jacobi jacobi.c -lm
   ```

   Then get on a compute node by typing

   ```
   interact -p GPU-shared --gres=gpu:v100-32:1 -t 10:00
   ```

   Run the code by typing **./jacobi**

   The reason we had to compile with -fopenmp, even though jacobi.c is a sequential code, is that we are using the OpenMP timing routines. Run three times and report the average sequential execution time. Be sure to type **exit** or **Ctrl-D** to exit the compute node when you are done.

   b. (10 pts) You are provided with an OpenMP version of the Jacobi iteration code in **omp_jacobi.c**. The code has a single parallel region inside the iteration loop, with several parallel for loops inside that region. The code is correct in that all data sharing is done correctly and there are not race conditions or bad interleaving. Compile the code by typing

   ```
   gcc -fopenmp -o omp_jacobi omp_jacobi.c -lm
   ```

   If you are not still on a compute node, get on a compute node again by typing

   ```
   interact -p GPU-shared --gres=gpu:v100-32:1 -t 10:00
   ```

   Set number of threads to 1 by typing

   ```
   export OMP_NUM_THREADS=1
   ```

   Run the code by typing **./omp_jacobi**

   Set number of threads ot 2 by typing

   ```
   export OMP_NUM_THREADS=2
   ```

   Run the code again by typing **./omp_jacobi**

   Is the parallel code with one thread about as fast as the sequential code, or is it slower? Does the code speedup with 2 threads or does it slow down? Examine the code and explain the reason for the poor performance.

   c. (10 pts) Fix the performance problem with omp_jacobi.c that you discovered in part b and recompile the code. Then get on a compute node with 4 GPUs/16 CPU cores by typing

   ```
   interact -p GPU-shared --gres=gpu:v100-32:4 -t 30:00
   ```

   Set OMP_NUM_THREADS appropriately to run with 2, 4, 8, and 16 threads. Report your times in a table. Graph the results showing the parallel speedup and comparing with linear speedup. Turn in your modified **omp_jacobi.c** along with your writeup.

2. For our task decomposition example, we will use the OpenMP quicksort example from Matloff Chapter 4. The code is provided for you in the file **omp_quicksort.c**.
    a. (10 pts) Add code to call built-in qsort() and compare the execution time of omp_quicksort using one thread with that of built-in qsort() using an input of sufficient size to get several seconds of execution time.
    b. (10 pts) Run omp_quicksort with the same input size from part a using 2, 4, 8, and 16 threads/cores. Remember that you need to request 4 GPUs to get 16 CPU cores. Discuss your results. If you don’t get significant speedup, give possible reasons.
    c. (25 pts Extra Credit) Make changes to omp_quicksort to try to get acceptable speedup. Explain why you made each change and evaluate the effect of each change and of all your changes combined. Turn in your modified **omp_quicksort.c** along with your writeup.