CS 4230/6230
Programming Assignment 1
Due Friday, 10/25/2024, 11:59pm

For parallel code development and testing for this assignment, use any multicore platform with OpenMP, e.g., CADE Lab or your laptop. After code development, performance should be reported on a 20-core node on the Lonepeak CHPC cluster (use batch scripts buggy_lonepeak_c20.bat and trmm_lonepeak_c20.bat). Note that you should not perform direct execution of programs on CHPC login nodes (account will get suspended if some threshold of total interactive CPU time is exceeded. **Submission**: 1) A PDF report providing explanations as specified for each question, 2) Submission of the fixed buggy.c file and only your final _par.c files for questions 2e and 2f (optional, extra credit question).

1. (25 points) Compile and execute the program buggy.c. It has sequential reference code and a parallel code for incrementing the values of elements in an array. The parallel code produces incorrect results. What is the reason? Fix the bug in the parallel program.

2. (75 points) The following code performs matrix multiplication of two upper-triangular matrices (stored as standard dense matrices with the lower triangular parts storing zero values) to produce the values for the result upper-triangular matrix.

```
for (i=0; i<600; i++)
 for (j=i; j<600; j++)        // trmm
  for (k=i; k<=j; k++)
   C[i][j] += A[i][k]*B[k][j];
```

   For each of the exercises below, use the files trmm_main.c, trmm_ref.c, and trmm_par.c, using "gcc -O3 trmm_main.c trmm_ref.c trmm_par.c" to compile the code. The file trmm_ref.c contains reference sequential code. The driver code in trmm_main.c invokes the sequential code in trmm_ref.c and performs multiple executions of the parallel version in trmm_par.c, using different numbers of threads, based on the number of cores on the multicore system. Performance results are reported for different thread counts, showing the best and worst performance (different executions of the same code result in run-to-run variation due to interference from other processes on the system that interleave execution with the tested code). A correctness check is also performed on the results by comparing the output of the parallel code version with the sequential code version.
   a) (15 points) Create and execute three parallel versions, one each by placing a work-sharing "#pragma omp for" above each of the three loops. Provide a clear explanation for the differences you observe regarding the code's execution in the 3 cases.
   b) (15 points) For the parallel version corresponding to the "#pragma omp for" on the outermost "**i**" loop, compare the maximum achieved GFLOPs performance for the following three variants, using **static** scheduling: i) default chunk size; ii) chunk size = 10; iii) chunk size = 1. What could explain the performance differences?
   c) (10 points) For the parallel version corresponding to the "#pragma omp for" on the outermost "**i**" loop, measure the maximum achieved GFLOPs performance for the following two variants, using **dynamic** scheduling: i) chunk size = 10; ii) chunk size = 1. Compare and contrast performance trends with those in part (b), attempting to explain the reason for the observed trend.
   d) (15 points) Repeat part (c) for a parallel version with "#pragma omp for" on the middle "**j**" loop. Compare and contrast the trends in part (c) with that observed here. What could explain the different trends with chunk size?
   e) (20 points) Perform loop permutation guided by stride analysis to enhance data locality. Attempt to maximize parallel performance by suitable choice of scheduling directives for the transformed code. Target performance: 100 GFLOPs on the target system (20-core node on Lonepeak cluster).
   f) (25 points) Extra Credit: Use loop unrolling on the version produced in part (e) to achieve performance of 150 GFLOPs on target Lonepeak 20-core node. Not all loops produce a benefit from unrolling. Why?