CSE 411 - Fall 2021: Project 1

TBB for Parallelism

Due date: Wednesday September 22 at 11:59pm (Groups of max 2)

- Part 1:

  The purpose of this part is to assess the efficiency of TBB. The attached file (prime.cc) implements a naive O(n) prime checker for integers 1--N. We are not going to try to change the algorithm to improve performance. Instead, we will try to parallelize it using three different approaches:

  1. TBB parallel_for.
  2. Manually using fork-join (std::thread) with static load balancing.
  3. Manually using fork-join (std::thread) with dynamic load balancing.

  Keep in mind that for small problems, it's likely that different approaches perform similarly. But for N = 10ˆ6 and 10ˆ9, it is possible to get a HUGE speedup and the differences between approaches should arise.

  The goal is to analyze how different approaches could be "efficient" and "hard to implement". That is why it is important to report "all" your trials (or at least those that make sense). For example, report the performance of the straightforward static load balancing approach that we discussed in lecture.

  You must produce a 2-3 page write-up of your experience. Describe the techniques you used to parallelize the code. Include graphs showing the performance for N = 10ˆ3, 10ˆ6 and 10ˆ9, with threads on the X axis and time on the Y axis. Results should be the average of 5 trials and should discuss variance.

- Part 2:

  The purpose of this assignment is to make correct code run faster. The attached file (gauss.cc) file implements the O(n^3) Gaussian Elimination algorithm. We are not going to try to change the algorithm to improve performance. Instead, we will try to use a variety of techniques, to include improving locality, using SIMD operations, and exploiting multicore, to accelerate the program.

  Keep in mind that for small problems, it's likely not possible to get a parallel speedup. But for 2048 and 4096, it is possible to get a HUGE speedup.

  You must produce a 2-3 page write-up of your experience. Describe the techniques you used to parallelize the code. Include graphs showing the performance for 2048 and 4096 matrices, with threads on the X axis and time on the Y axis. Results should be the average of 5 trials, and should discuss variance.

General Instructions for both parts:

- For Part 2 and the parallel_for implementation of Part 1, we will be using Intel's oneAPI Threading Building Blocks (oneTBB) to take advantage of multiple cores. You will need to install libtbb-dev in order to use TBB.

- It is wise to vary the grainsize in TBB. You may add a command-line parameter for that purpose.

- TBB allows specifying the number of threads. In order to generate charts that show speedup at different thread counts, you will want to use the task_scheduler_init object.

- You will need to understand C++ lambdas in order to complete this assignment.

- You should also think about numerical stability. "Pivoting" is essential!

- If you are using Docker, chances are good that Docker on your laptop won't let you access all the cores of your laptop. At some point, you'll need to test your code on the sunlab. When you do that, you'll need to update your Makefile accordingly. You will probably also need to manually place a copy of TBB in your home folder on sunlab.

- When testing on Sunlab, keep in mind that the machines are shared. If you wait until the last minute, you may not have exclusive access to the machine, and your results will be invalid.