Barış(Jeffrey ) He/ James Lin

COMP429 Assignment 2 Writeup

For this life game, two parallel programming implementations were created. One utilizes open MP and the other utilizes pthreads. Below, we discuss our implementation approach for both, and then together compare the results of each implementation with each other as well as the serial version.

OpenMP: For this approach, we started by paralleling the mesh update algorithm with a simple ‘omp parallel for’ clause. Since the calculation of the new mesh values are written to a different world from the world we are reading from, there are no data dependencies and data parallelism is easily implemented. We assign a single thread to swap world pointers and to plot the mesh grid, and allow the rest of the threads to start calculating the new mesh for the next function iteration. It is important to note the ‘nowait’ flag that follows the ‘omp single’ clause which allows the other threads to continue their execution without waiting for this single thread to swap pointers and plot the mesh. (Thus implementing task parallelism). It is also important to note the ‘omp barrier’ right before this ‘omp single’ section since we need to ensure the previous plot has been plotted completely, and that the new plot has been fully calculated before we can swap pointers and repeat the same process over again.

Pthreads: This approach contains the same concept of implementing task and data parallelism. We assign one thread to update the points and the other threads to calculate the new mesh. Here, I will only discuss the differences from the OpenMP version, since the logic for the barriers, assignment of roles, and data/task parallelism is the same. The first difference is the specifying of the master thread, (threadID == 0) to do the pointer updating and plotting. It does not matter from a functional standpoint which thread completes these tasks, but as a pthreads programming convention, thread 0 is chosen since it will always exist no matter how many how many threads are created. The OpenMP version only calls for a single thread, as opposed to the master thread. A second difference is the explicit division of the tasks among threads. While OpenMP divides the work implicitly among created threads, we much manually calculate and assign tasks for each of the threads created through pthreads. You will notice in the code that multiple lines are dedicated to the purpose of task division and assignment, including clauses that ensure code correctness even in the case that there is only one thread. Another difference with pthreads is the need to manually pass parameters due to the method that threads are created via pthreads. One will notice a large number of lines are used to pack and unpack parameters to be passed to each of the threads. Other than these differences, the code implementation approach remains the same.

Results: Here below are timing results from the provided timer function, as well as the environment they were run in. Discussion of results can be found below.

* Condition n=250, iterations=100, p=0.2, seed=1234, tested on cinekop cluster via ssh and x11 forwarding for the results with display
  + OpenMP with display 17.45s
  + OpenMPwithout display 0.23s
  + Ptherad with display 18.71s
  + Pthreadwithout display 0.23s
  + Serial with display 17.72s
  + Serial without display 0.45s
* Condition n=500, iterations=300, p=0.2, p=0.2, seed=1234, tested on cinekop cluster via ssh (note the increased mesh size and iterations. This was done to better gauge time differences between different versions without the display.
  + OpenMPwithout display 2.42s
  + Pthreadwithout display 2.97s
  + Serial without display 5.11s

Discussion: The first important point to note is the significant timing difference between the results with display and without display. In each of the versions, the openMP, pthread, and serial, the I/O time completely masks the time necessary for the completion of the algorithm. As a result, the timing difference each of the versions is less than 5% deviation from any of the other results. Due to I/O and the saturation of this bridge, parallelism has negligible speedup. For the results without the displays, for both the version with small amount of computation (n=250,i=50) and for the version with a larger amount of computation (n=500,i=300, and so about 24 times more data crunching necessary), one can note that the parallel versions have about 2 times speedup. While the data provided above is for the calling of 4 threads for both openMP and pthreads, we also tested for a higher number of threads with similar results (though speedups actually decreased a bit, due to context switching and additional overhead). The 2 times speedup is expected, however, due to the 2 core architecture of cinekop which is the hardware system we ran the program on.