Lachlan Sinclair

sinclala@oregonstate.edu

4/4/2020

CS 475 Spring

Project #0: Simple OpenMP Experiment

Explanation: I used the code provided in the project description. Running the for loop using 4 threads increased the speed of execution as expected when compared to using a single thread since the iterations of the for loop were split amongst the 4 threads. I had originally tried to run the program locally using cygwin64, but apparently there is a known issue that slows down runtimes when using multiple threads through OpenMP in cygwin64. I also ran it locally in visual studios but didn't use those results in this report, rather I used the results from a slightly modified cpp file that was run via a simple bash script on the flip servers. The bash script will output 100 max MegaMults/Sec values obtained while using 1 thread and 4 threads. The first value in each row of the output is the value from using 1 thread and the second value is from using 4 threads. I will be including the VS version of the code and the code that runs with the bash script in the submission.

1. The results below were obtained from running the program using the bash script "proj0script" on the flip1 server, the results were piped to a txt file which I then loaded into Excel. I used Excel to find the average values over the 100 trials.
2. In this program I used an array size of 30,000. With four threads, the average max value of 6 tries over 100 separate runs was 2627.755 MegaMults/Sec. Using one thread, the average max value of 6 tries over 100 separate runs was 826.5232 MegaMults/Sec.
3. 4 to 1 thread speedup = 2627.755/826.5232 = 3.18
4. The speedup wasn't four or greater since running the threads likely encountered cache contention, bus contention and contention with other processes. It's been a while since I took the assembly course, but it safe to assume most of those contentions were likely slowing down the threads. Clearly much of the work was able to be done in parallel since there was a significant speed up, however I do not think the speedup should have a one to one ratio with the thread count increase, especially when you consider how costly generating and terminating the threads is (even when considering the size of the array).
5. And for the mysterious Parallel fraction I got Fp = (4/3)*(1-(1/3.18)) = .91