9 April 2018
Project #0
OSU CS475 Sp2018
Joshua L. Rogers
rogerjos@oregonstate.edu

Simple OpenMP Experiment

I chose to run the experiment on flip, so I logged in and checked all of the flips for the lowest load via uptime. The best result was on flip2: "load average: 1.00, 1.01, 1.05," so this was the server I used.

To keep things simple (see: project title) I added two omp_get_wtime() calls to the code in order to track the overall runtime of the parallelized loop. I wrote the aptly-named simplescript that takes the array size and number of tries as arguments, then compiles the proj00.c code and runs it with 1 and 4 threads, sending output to simpletest1.out and simpletest4.out respectively. Then I chose 1000000 for both ARRAYSIZE and NUMTRIES, reasoning that the size was sufficient to ensure quality results despite threading overhead and the number of tries sufficient to avoid load surge. Then I executed the code twice contemporaneously via the script, running a version with four threads and a version with one thread in background with output redirected. The results were as follows:

	1 Thread	*4 Threads*
Time (s)	3137.39	830.91
Peak Perf. (Mm/s)	324.88	318.87
Avg. Perf. (Mm/s)	1246.39	1204.38

This indicates a speedup $\mathbf{S} = \frac{3137.39}{830.91} = \sim 3.78$ and a parallel fraction $\boldsymbol{Fp} = \frac{4.}{3.} * \left(1. - \frac{1.}{S}\right) = 0.98$

Based on the "How Reliable is the Timing" section of the project notes I believe that these results are reliable given that over a significant number of iterations the peak performance does not differ significantly form the average performance. I am not surprised that the speedup is close to but nor equal to or greater than 4. Obviously threading is not "free," and there must be some overhead involved or there would be no reason to use thread pools. In a hypothetical world in which thread creation and management had no cost the speedup would presumably equal 4.00. If the speedup were greater than 4 this would indicate "negative overhead" to threading. I'm not sure how that would be possible in the real world, although conceivably some very strange engineer might design a machine with one slow core and three fast cores, then a single threaded program run on the slow core versus a multithreaded program run on all cores would show a speedup greater than the number of threads.