Jordan Grant (grantjo)
CS 475- 400
Project 0 – Simple OpenMP Experiment

Platform

The experiment was conducted on a Macbook Pro (late 2011) running Ubuntu LTS 16.04 Xenial Xerus (kernel release 4.4.0-66-generic), with the following specs.

CPU:	Intel i5-2435M 2.4 GHz dual core with hyper-threading
Motherboard:	Apple Inc. Mac-94245B3640C91C81
Memory:	8GB DDR3-1333

Performance

Performance was calculated in Mega Multiples per Second (MM/s). This was performed with a float array size of 2^{23} and the calculations were performed 100 times to compute the average performance. First the maximum and average performance was computed with a single thread, followed by the maximum and average performance with 4 threads. The experiment was conducted using a bash script to automate each call to the program during a time of minimal system load (all editors and unrelated programs closed, no other users active).

Bash Script Usage: **runprog0**

The script will compile prog0.c, run it with 1 and 4 threads, and print out the computations to the console.

Here are the values for max and average performance for 1 and 4 threads:

Single Thread

Maximum (MM/s):	366.51
Average (MM/s):	349.97

Four Threads

Maximum (MM/s):	875.29
Average (MM/s):	862.06

Speed Up

Speed up was calculated as:

Avg. Performance with 4 Threads (MM/s) / Avg. Performance with 1 Thread (MM/s). The speed up calculation was performed using python from within the bash shell script.

Speed Up: 2.463

Theoretically, you would expect 4 threads to have four times the performance of 1 thread. Thus, you would expect a speed up of 4.0. However, you do not always have physical threads available to devote entirely to running the concurrent program. Often these resources will be shared with other programs. I have 2 cores with hyperthreading, meaning that I have 4 “processors” according to my linux OS. Each

thread in my 4 thread for loop will be able to run on separate hardware. However, there are many other programs running on my system that will be sharing these processors. This means that each thread will likely be interrupted for another program to get cpu time.

In addition to contention for processor time, there is overhead for creating threads and waiting for all the threads to join at the barrier of the for loop. This overhead is another significant contributor to the non-ideal speedup.

Parallel Fraction

As requested, below is my parallel fraction although we do not quite know what it is yet.

FP = 0.79