Chido Nguyen
nguychid@oregonstate.edu
931506965

Program 0: Simple OpenMP Experiment

Machine Tested On: Oregon State University Servers (flip1.engr.oregonstate.edu)

Results: (Array Size = 65536, Number of Tries = 50)

For a single thread running the array size listed above the **peak** performance over 50 runs was **329.24 MegaMults / Sec**, whereas the **average** performance was **319.09 MegaMults/ Sec**.

```
Using 1 threads
Peak Performance = 329.24 MegaMults/Sec
Average Performance = 319.09 MegaMults/Sec
```

When using multiple threads (4) to implement the same code with the same array size and number of tries **peak** performance was **1203.46 MegaMults/ Sec**, and **average** performance was **1110.57 MegaMults/ Sec**.

```
Using 4 threads
Peak Performance = 1203.46 MegaMults/Sec
Average Performance = 1110.57 MegaMults/Sec
```

By increasing the number of threads to do concurrent calculations we were able to achieve a speed up of ~3.4804. Speed up calculation is figured by taking execution time of 1 thread divided by execution time of 4 threads. I used basic algebra to manipulate the MegaMults / Sec equation to figure our execution time. Instead of using just 1 MegaMult I'm using the average performance value. Normally Average Performance = ArraySize / execution time / 1,000,000; we can solve for Execution Time = ArraySize / (Performance * 1,000,000). I added this into my code and had it calculate the Speedup for me after both 1-thread and 4-threads has had a chance to run.

Intuitively I believe the speed up is happening because we're adding more "workers" to the mix i.e. the threads. How I understood it is that if we had 1 thread doing 1 calculation a second it would take 4 seconds to do 4 calculations, now if we had 4 threads doing 4 calculations it would take us 1 second. Instead of seeing a speedup of 4.0 , my data shows 3.48 is probably due to not actually stressing the system. The array size or the computation might still be do-able for our system without requiring it to push 100% of its power.

```
Speedup from 1 to 4 is about 3.4804
```

Afterwards I took the speedup value (3.4804) and inserted it into the parallel fraction equation and got .9502.

```
Parallel fraction is: 0.9502
```

Reliability:

Server loads are within acceptable range in my opinion sitting around 2.8.

```
12:27:12 up 97 days, 16:24, 88 users,  load average: 2.79, 2.80, 2.85
```

Chido Nguyen
nguychid@oregonstate.edu
931506965

Data for first run @ 65536 and 50 tries

```
flip1 ~/cs475/p0 1024$ Project0Run; uptime
Using 1 threads
Peak Performance = 329.24 MegaMults/Sec
Average Performance = 319.09 MegaMults/Sec
Using 4 threads
Peak Performance = 1203.46 MegaMults/Sec
Average Performance = 1110.57 MegaMults/Sec
Speedup from 1 to 4 is about 3.4804
Parallel fraction is : 0.9502
 12:27:12 up 97 days, 16:24, 88 users,  load average: 2.79, 2.80, 2.85
flip1 ~/cs475/p0 1025$
```

Comments about my code:

I added a duplicate section of code to run both 1 thread and 4 threads at the same time. I also added in the equations/formulas to solve for the speed up and parallel fraction after both threads experiments are done running.

It ran how I expected it to ran. 4 threads doing work should be faster than 1 thread doing the work. By faster I mean putting out more results. But I also know that up to a certain amount of threads the “returns” become diminishing. With 4 threads we can expect 4x the speed up , but as we add more cores/threads it won’t be as linear.