Project #0
Simple OpenMP Experiment

By Michael Rose

Due: April 4th 2016

Written for: CS475 Spring 2016

Oregon State University

Table of Contents

What I did

I ran this project on 3 machines to get a feel for how my older linux boxes stacked up to eachother and to flip. One of my machines is running an Intel Core 2. The other machine is running a first or second generation i5. I ran the program with an array size of 1,000,000 and used 1000 cycles to calculate the average. I ran the programs with less cycles as well to see if my local machines were just overheating, but the results were similar to the results listed below.

Intel Core 2

Running a single thread, I got 100/78. Running with 4 threads, I got 158.85. Running with 8 threads I got 156.94.

I think that there is only about a 60% increase because the processor is limiting the improvement of the multithreading.

Local i5

Running with a single thread, I got an average of 270.02 MegaMults/Sec. Running with 4 threads, I got an average of 587.69 MegaMults/Sec. Running with 8 threads, I got a decrease in performance with only 467.07 MegaMults/Sec and a peak of 584.98 MegaMults/Sec.

Having results that cap off at around 3 times the value of a single thread makes me think that my local machine is also hitting a processes limit.

Flip

On flip, a single thread got an average of 196.24 MegaMults/Sec. 4 threads got an average of 656.17 MegaMults/Sec and 8 threads got 1539.18 MegaMults/Sec.

These results make the most sense as the threading is basically increasing the efficiency by the number of threads. For the 4 thread count, the peak performance is 811.68 which is more like 4 times the value for a single thread.

When I ran this code, I got the following values for uptime.

Before running: .09, .04, .05

After running: .27, .11, .07

Summary

Core	1 Thread (MegaMults/Sec)	4 Threads (MegaMults/Sec)	8 Threads (MegaMults/Sec)
Intel Core 2	103.65	158.85	156.94
I5	270.02	587.69	467.07
Flip	196.24	656.17	1539.18