Jessica Spokoyny
April 9, 2017
Project #0
Simple OpenMP Experiment

1. Using the given code, I ran this program on the flip server in linux from a windows 10 laptop. My main function is contained in a file called project0.cpp which I compiled by typing:

   % g++ -I/usr/local/common/gcc-5.2.0/ project0.cpp –o proj0 –O3 –lm –fopenmp

   Then I executed the program with:

   % ./proj0

   I set the array size to be 1024 x 1024 = 1,048,576 and I set the number of tries to be 100

2. My performance results:

   <u>When number of threads is 1:</u>

   NUMT 1
   ARRAYSIZE 1048576
   NUMTRIES 1000

   Peak Performance = 1852.64 MegaMults/Sec
   Average Performance = 1814.78 MegaMults/Sec

   <u>When number of threads is 4:</u>

   NUMT 4
   ARRAYSIZE 1048576
   NUMTRIES 1000

   Peak Performance = 5440.44 MegaMults/Sec
   Average Performance = 5297.29 MegaMults/Sec

3. My 4-threads-to-1-thread speedup (S) is calculated by dividing (MegaMults for 4 threads) / (MegaMults for 1 thread). Therefore, S = 3570.88 / 1814.78 = **2.919**

4. The speedup is faster because we are running 4 operations at a time instead of just 1. This make sense because if we had 100 operations to run and only 1 thread available, this would take 100t (where t is the time it takes to perform the operations). If we had these same 100 operations but were able to parallelize them all to run 4 simultaneously, this would theoretically take .25(100t) = 25t. Cleary, using parallel threads speeds up the time needed to execute the program.

5. My parallel fraction (Fp) is calculated by multiplying (4/3)*(1 - (1/S)). Therefore, Fp = (4/3)*(1-(1/2.919)) = **.877**