CS 475 Parallel Computing
Professor Mike Bailey
Student: Khuong Luu
Project 0 – Simple OpenMP Experiment

1. **Tell what machine you ran this on**

The computer I ran this experiment on has:
- Operating System: Ubuntu 16.04
- Kernel: 4.13.0-37-generic
- Processor: Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz, 4 cores
- Memory: 16GB, 2400Mhz

2. **What performance results did you get?**

I used arrays of 1000 elements and tried 10 times

```
#define ARRAYSIZE      100000000
#define NUMTRIES       10
```

Performance for 1 thread:

```
$ proj0 1
Using 1 threads
   Peak Performance =   502.10 MegaMults/Sec
Average Performance =   466.71 MegaMults/Sec
Average Execution Time = 226804373.00 nano seconds
```

Performance for 4 threads:

```
$ proj0 4
Using 4 threads
   Peak Performance =  1822.87 MegaMults/Sec
Average Performance =  1704.57 MegaMults/Sec
Average Execution Time = 61793663.40 nano seconds
```

3. **What was your 4-thread-to-one-thread speedup?**

S = (Execution time with one thread) / (Execution time with four threads)
  = 226804373.00 / 61793663.40
  = 3.67

So my 4-thread-to-one-thread speedup is 3.67

4. **Why do you think it is behaving this way?**

By splitting up the work (multiplication of 2 corresponding elements in array A and array B) upon 4 threads, the parallelization of this program have increased the performance and decreased the running time of the program. So the execution time with one thread is longer than the execution time with 4 threads.

However, as I also observed in on of my further experiments, if we run the program on only around 10^2 or 10^3 elements, then the execution time with 1 thread will in fact faster than running with 4 thread. This is because the problem size (number of elements in array) is so small that the cost of overhead from parallelization has outweighed the speed up benefit.

5. **What was your Parallel Fraction, Fp?**

Fp = (4./3.)*(1. - (1./S))
 = (4./3.)*(1. - (1./3.67))
 = 0.970027248