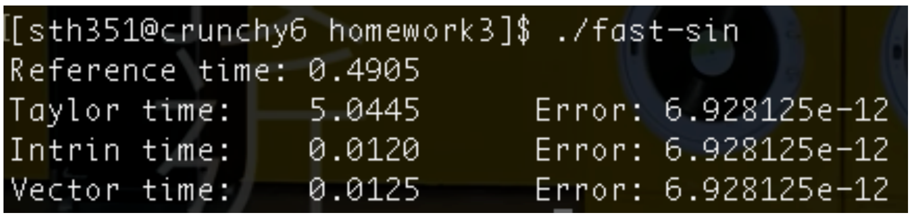
Spring 2019: Advanced Topics in Numerical Analysis: High Performance Computing Assignment 3

Name: Shih-Ting(Emily) Huang
NetID: sth351

1. **Approximating Special Functions Using Taylor Series & Vectorization :**

   I just add the more terms to sin4_intrin() using both SSE and AVX and sin4_vec(), and this is the result that is run on CIMS crunchy6 server:



```
[sth351@crunchy6 homework3]$ ./fast-sin
Reference time: 0.4905
Taylor time:    5.0445      Error: 6.928125e-12
Intrin time:    0.0120      Error: 6.928125e-12
Vector time:    0.0125      Error: 6.928125e-12
```

   According to the above result, we can know that the accuracy of the version that I revised is about 12-digits because the error is the same as sin4_taylor(). But the speedup of sin4_intrin() and sin4_vec() can be up to 420 times.

2. **Parallel Scan in OpenMP:**

   In my implementation, I divide it into p parts, and each thread is responsible for n/p number. If there is a remainder, I distribute the remaining to the last thread. However, we know that the for loop in serial code is not independent. Thus, if we do it in different threads, we have add the correction at the end in serial. The following are the result that is executed on CIMS crunchy6 server with different thread number and n = 100000000:

Thread number	Time (sec)
Serial	1.156331
2	0.345736
4	0.206021
8	0.119277
16	0.136855
32	0.153455
64	0.150528

   According the the above table we can find that when we increase the number of threads, we can higher performance, for example: for thread number = 2, 4, 8. But when the thread number is too large, for example: for thread number = 8, 16, 32 ,64, we can not get a speedup or even has worse performance due to the overhead of creating and managing threads. The highest performance is happened when thread number = 8, the speedup can up to about 10 times. On CIMS crunchy6 server, there are 64 cores, and I also check it with omp_get_max_threads() and getting the output = 64.