HPC Assignment 3 (due Apr. 4, 2022)

Yuan Chen (yc5588)

2. Approximating Special Functions Using Taylor Series & Vectorization.

1. Improve the accuracy to 12-digits for any one vectorized version by adding more terms to the Taylor series expansion.

   The processor is : Intel(R) Xeon(R) CPU E5-2680 v2 @ 2.80GHz, 20 nodes.

   So I improved the accuracy on **AVX part of the function `sin4 intrin()`** and on `sin4 vec()`. After adding totally **6 terms** (to x^{11}), we get 12-digits accuracy. Results are in the following Figure.

```
[yc5588@snappy4 homework3]$ ./fast-sin
Reference time: 17.2294
Taylor time:    3.2719      Error: 6.928125e-12
Intrin time:    0.8937      Error: 6.928125e-12
Vector time:    0.8933      Error: 6.928125e-12
```

2. develop an efficient way to evaluate the function outside of the interval $x \in [-\pi/4, \pi/4]$ using symmetries.

   Taylor series approximation of $\cos(x)$ at $x = 0$:

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \frac{x^{10}}{10!} + \cdots$$

   We can do approximation at for $\cos(x)$ at $x \in [-\pi/4, \pi/4]$ using the same technique as before, then for $\tilde{x} = x + k \cdot \frac{\pi}{2}$, where $x \in [-\pi/4, \pi/4]$, we have

$$\sin(\tilde{x}) = \begin{cases} \sin(x), & k\%4 = 0 \\ \cos(x), & k\%4 = 1 \\ -\sin(x), & k\%4 = 2 \\ -\cos(x), & k\%4 = 3 \end{cases}$$

   I improved the accuracy on `sin4_taylor_all()` and `sin4_vector_all` in the code.

   After testing on domain $[-100\pi, 100\pi]$, I get the following result:

```
Reference time: 38.1208
Taylor time:    24.6457      Error: 6.944556e-12
Intrin time:    0.8937      Error: 7.362222e+19
Vector time:    13.1466      Error: 6.944556e-12
```

(I didn't improve for Intrin, so the result is not accurate for it.)

3. Parallel Scan in OpenMP

- the architecture I run it on:

```
Model:                   62
Model name:              Intel(R) Xeon(R) CPU E5-2680 v2 @ 2.80GHz
Stepping:                4
CPU MHz:                 1200.048
CPU max MHz:             3600.0000
CPU min MHz:             1200.0000
```

- the number of cores of the processor: **20**

- Result with different thread numbers:

Run command: `./omp-scan n` where `n` is the number of threads.

	time (s)	
number of threads	sequential	parallel
1	0.259619	0.343844
2	0.256372	0.234137
3	0.262151	0.179302
4	0.260642	0.143779
5	0.256775	0.128303
6	0.258259	0.097869
7	0.260097	0.078647
8	0.259986	0.06862
9	0.261015	0.065818
10	0.259837	0.061889
11	0.259046	0.061412
12	0.259137	0.058899
13	0.259362	0.063069
14	0.259275	0.059169
15	0.259449	0.065005

It is shown that as number of threads increases, the parallel timing decreases; and it runs to the bottleneck at around **12** threads.