HPC Hw3 (Zhe Chen)

Machine configuration

All results are got by running on my office's machine 'blob'. It has a 4 core CPU, Intel Xeon E5-1603, 2.80GHz.

Max memory size is 375GB, max memory bandwidth is 31.4GB/s. It uses Sandy Bridge EP, so operations per cycle is 8. Thus, theoretic FLOPS/s is 4 Cores * 2.80GHz * 8 operations/cycle = 89.6 GFLOPs/s

Makefile

All codes are compiled with g++, "O3" flags, openmp. Command is like:

```
g++  -std=c++11 -O3  -march=native -fopenmp  target.cpp -o target-omp
```

P1: Approximating Sine Function with Taylor Series & Vectorization

1.First, I improve sin4_vec to be 12 digit accuracy just by adding higher Taylor terms up to order 11. The performance is shown as below.

Extra credit

2.To evaluate sine function outside of $[-\pi/4, \pi/4]$ efficiently, I develop scheme as following.

2.a). Move any 'x' to $[-\pi/4, 7\pi/4]$ since it's 2π circulant by $x = x - 2\pi floor(x + \pi/4)$.

2.b). By $sin(x) = -sin(x - \pi)$, move $x \in [3\pi/4, 7\pi/4]$ to $[-\pi/4, 3\pi/4]$.

2.c). We can already compute $sin(x)$ in $[-\pi/4, \pi/4]$. To evaluate it in $[\pi/4, 3\pi/4]$, we use formula $sin(x) = cos(x - \pi/2)$. Thus, the problem is converted to compute $cos(x),\ x \in [-\pi/4, \pi/4]$. We could just use cosine function's taylor expansion up to x^{12} to get 12 digits accuracy.

The reslt is shown in below as "(extended range)".

```
Reference time: 22.9906
Reference time (extended range): 32.7627
Taylor time:    4.4044      Error: 6.928125e-12
Taylor time (extended range):    19.5440      Error: 6.928014e-12
Intrin time:    1.3712      Error: 2.454130e-03
Vector time:    1.5247      Error: 6.928125e-12
```

P2: Parrallel Scan in OpenMP

To parallelize scan operation, I divide n into nthreads batches with each batch's length=n/nthreads. Generally, it's better to have nthreads less or equal to max core number, which is 4 in my machine.

In each batch, we calculate prefix-sum locally and record the last term $batch[i]$. Then we scan $batch[i]$ to get $prefix\_sum\_batch[i]$, which is the value that should be added into batch i+1.

Importantly, we should add $prefix\_sum\_batch[i]$ to each batch parallelingly too. Or we lose the point of paralleling scan operator. It's also a compete scan to do this operation!

The result is shown below and we can see good scaling.

Number of Threads	1	2	3	4
Parallel Scan / Sequential Scan	1.4336	0.8748	0.6125	0.5047