HPC23
Homework 3
Letao Chen
lc5187

git repo: `https://github.com/LetaoC/HPC23.git`

1. Assume n is even for simplification.

   (a) Take the first for loop as example, the first thread spent approximate $(1 + \frac{n}{2}) \times \frac{n}{4} = \frac{n}{4} + \frac{n^2}{8}$ milliseconds, while the second thread spent approximate $(\frac{n}{2} + 1 + n) \times \frac{n}{4} = \frac{n^2}{8} + \frac{n}{4} + \frac{n^2}{4}$ milliseconds. Hence, the first thread needs to wait approximate $\frac{n^2}{4}$ milliseconds since it is static.
   For the second loop, the time spent is opposite. The second thread spent less time, and it needs to wait approximate $\frac{n^2}{4}$ milliseconds.

   (b) The time spent reduced for both two threads. If we use `schedule(static,1)`, then the two threads execute loop alternatively.
   Take the first for loop as example, the first thread spent approximate $\frac{n^2}{4}$ milliseconds in total and the second thread spent approximate $\frac{n^2}{4} + \frac{n}{2}$ milliseconds in total. For the second loop, the time spent by two threads are the opposite.

   (c) If we use `schedule(dynamic,1)`, the total time spent by two threads don't change, but the total execute time reduced since the thread doesn't need to wait for each other.

   (d) The directive is `no wait`, it could improve performance by reducing the idle time of threads.

2. The running time with different thread numbers (4, 6, 8, 12, 16) when N = 100mm is listed below:

	sequential scan	parallel scan
4	0.450615s	0.281943s
6	0.459123s	0.258181s
8	0.361315s	0.210578s
12	0.307372s	0.227079s
16	0.295118s	0.223010s

   The architecture I use is x86_64, 4 cores, one thread per core.

3. The iteration steps needed to reach a decrease of the initial residual by a factor of $10e4$ for different N of two methods are shown below:

	Jacobi	Gauss-Seidel
N = 7	72	70
N = 15	273	267
N = 35	1272	1255
N = 49	2366	2338

   Which follows the fact that G-S method converges faster than Jacobi.
   The timing analysis for two methods are shown below:

(a) Jacobi

The execution times for Jacobi using OpenMP v.s. without OpenMP for different N are shown as follows:

	omp	non-omp
N = 7	0.000921s	0.000333s
N = 15	0.004309s	0.001424s
N = 35	0.023059s	0.025122s
N = 49	0.06391s	0.090576s

We can see that as N gets larger, the execution time difference becomes larger, the one uses OpenMP executes faster.

To view the timing for different threads, I compare the execution time of the first 1000 iterations when $N = 1000$. The timings are shown below:

	timing
Threads = 4	7.662612s
Threads = 6	10.722935s
Threads = 8	10.217466s

We can see that the running time for different number threads are still relatively close.

The architecture I use is x86_64, 4 cores, one thread per core.

(b) G-S

The execution times for Gauss-Seidel using OpenMP v.s. without OpenMP for different N are shown as follows:

	omp	non-omp
N = 7	0.001854s	0.001127s
N = 15	0.003748s	0.002816s
N = 35	0.030390s	0.042085s
N = 49	0.087682s	0.098790s

We can see that as N gets larger, the execution time difference becomes larger, the one uses OpenMP executes faster.

To view the timing for different threads, I compare the execution time of the first 1000 iterations when $N = 1000$. The timings are shown below:

	timing
Threads = 4	7.230244s
Threads = 6	9.226406s
Threads = 8	10.09028s

We can see that the running time for different number threads are still relatively close.

The architecture I use is x86_64, 4 cores, one thread per core.