HPC - Homework 3

1. OpenMP warm-up

a)

- First loop: T2's execution time, (3n - 1)(n - 1)/8, dominates over T1's execution time of (n + 1)(n - 1)/8. So each thread would spend (3n - 1)(n - 1)/8 in execution, and T1 would spend (n^2 - 2n + 1)/4 in waiting for T2.
- Second loop: T1's excution time of (3n - 1)(n - 1)/8 dominates over T2's execution time of (n + 1)(n - 1)/8. Both threads would spend (3n - 1)(n - 1)/8 in execution, and T2 would spend (n^2 - 2n + 1)/4 in waiting for T1.

To sum up, both threads spent **(n^2 - n)/2** in executing the parallel region and spent **(n^2 - 2n + 1)/4** in waiting the other thread.

b)

- First loop: T1's execution time is equal to the sum of 1, 3, 5, 7, ... till n - 1 or n - 2 depends on n is odd or even. The execution time of T2 equals to the sum of all the even numbers from 2 to n - 1 or n - 2 depending on whether n is odd or even. The total time is about n^2/4.
- Second loop: The execution time of T1 is equal to the equation time of T2 in the first loop, and T2's execution time equals to the one of T1 in the first loop. So the overall execution time is still bounded by n^2/4.

To sum up, both threads spent **n^2/2** in executing the parallel region and spent almost no time in waiting the other thread.

Compared to the static schedule in part (a). The execution time increased slightly by n/2.

c)

Using schedule(dynamic, 1) would not improve the execution time since each thread still get a block of size 1 and the overall execution time is bound by the thread that takes longer to execute due to the existence of the barrier at the end of the loop.

d)

The OpenMP directive that eliminates the waiting time is **nowait**. If this clause is used, the execution time of each thread becomes the real execution time, not including the wait time.

2. Finding OpenMP bugs

omp_bug2

1. tid is shared across all threads. It should be private
2. Total is a shared variable. There is read and write conflicts. When updating it, we need to use reduction directive. Otherwise, we would get unpredictable result.

omp_bug3

The function print_results is called inside a section, where only one thread is running. The omp barrier, however, is expecting more threads to arrive. This would cause the program hang at the barrier directive. Removing the barrier directive at the end of the implementation of the print_results function solves this issue.

omp_bug4

The segmentation fault is caused by stack overflow. There is not enough memory space to store the 2-D array a of size 1048x1048.

omp_bug5

Deadlock occurs becaue the first thread holding lock a and requesting for lock b, while the second thread holding the lock b and requesting lock a. I solved this by implementing the 2PL where each thread acquires two locks at the same time and only release them once that thread is done.

omp_bug6

If we have #pragma omp parallel directive outside the dotprod function, the variable sum declared in dotprod is private to the thread, while the reduction directive expects it to beshared among threads. Thus, we need to move the parallel directive inside the dotprod function.

3. Parallel Scan in OpenMP

# of threads	time
1	2.698287s
2	1.694251s
4	1.462898s
8	1.365168s

Architecture

```
vendor_id   : AuthenticAMD
cpu family  : 21
model       : 1
model name  : AMD Opteron(TM) Processor 6272
stepping    : 2
microcode   : 0x600063e
cpu MHz     : 2100.063
cache size  : 2048 KB
physical id : 0
siblings    : 16
core id     : 0
cpu cores   : 8
```

4. OpenMP version of 2D Jacobi/Gauss-Seidel smoothing

N	1 thread	2 threads	4 threads	8 threads
10	0.002	0.002	0.003	0.003
20	0.025	0.020	0.018	0.017
50	0.749	0.549	0.433	0.376
100	8.373	6.105	4.601	3.936

See implementation in jacobi2D-omp.cpp and gs2D-omp.cpp

How to compile

```
g++ -Wall -pedantic -std=c++11 -fopenmp gs2D-omp.cpp && ./a.out N
(N is the size of the matrix)
```

Architecture

```
vendor_id   : AuthenticAMD
cpu family  : 21
model       : 1
model name  : AMD Opteron(TM) Processor 6272
stepping    : 2
microcode   : 0x600063e
cpu MHz     : 2100.063
cache size  : 2048 KB
physical id : 0
siblings    : 16
core id     : 0
cpu cores   : 8
```