HPC - HW 2

Kitty Li (wl2407)

Credit: Received help from Jimmy Zhu

Processor: Intel(R) Core(TM) i9-9980HK CPU @ 2.40GHz
Github repository: https://github.com/lwyhasacat/HPCHW3

Problem 1.

a) When n is an odd number, thread 0 and thread 1 would execute function f the same number of times. Thread 0 will execute f for $(n-1)/2 + ... + 1 = (n+1)(n-1)/8$ miliseconds, while thread 1 will execute f for $(n+1)/2 + ... + (n-1) = (3n-1)(n-1)/8$ miliseconds. The second for loop inverts the order so they will change the order and execute f again. The time they spent in waiting will be the two times the difference between the execution time of thread 0 and thread 1, which is $(n-1)^2/2$. Total time they spend is $(3n-1)(n-1)/4$ miliseconds.
When n is an even number, thread 0 will get one more function call than thread 1. Thread 0 will execute f for $n/2 + ... + 1 = n^2/8 + n/4$ miliseconds, while thread 1 will execute f for $(n/2+1) + ... + (n-1) = 3 * n^2/8 - 3 * n/4$ miliseconds. The second for loop inverts the order so they will change the order and execute f again. Thread 0 will execute f for $(n-1) + ... + n/2 = 3 * n^2/8 - n/4$ miliseconds, while thread 1 will execute f for $(n/2-1) + ... + 1 = n^2/8 - n/4$ miliseconds. The wait time is $n^2/2 - n$ miliseconds, in which thread 0 waits for $n^2/4 - n$ miliseconds and thread 1 waits for $n^2/4$ miliseconds. Total time will be $3 * n^2/4 - n$ miliseconds.

b) When we use schedule(static, 1), we are alternatively giving two threads one function call. When thread 0 executes f(1), f(3),..., thread 1 executes f(2), f(4), ... and therefore the waiting time will be shorter. Also, it doesn't matter which thread gets the function call first.
When n is odd, one of the thread will execute f for $(n-1) + ... + 2$ and the other one will execute f for $(n-2) + ... + 1$ miliseconds. For just the first for loop. the total time will be $(n-1) + ... + 2 = (n-1)(n+1)/4$ miliseconds, and the wait time will be $(n-1)/2$ miliseconds. The second for loop will be executed similarly as the first for loop, so the total time will be $(n-1)(n+1)/2$ miliseconds and the wait time will be $n-1$ miliseconds.
When n is even, the first for loop should be $(n-1) + .. + 1 = n^2/4$ miliseconds, and the wait time should be $n/2$ miliseconds. The second for loop should have the same run time as the first for loop. The total time will be $n^2/2$ miliseconds and the wait time will be n miliseconds.

c) For schedule(dynamic, 1), when n is odd, the two threads will behave the same as when we use schedule(static, 1) in the first for loop, meaning that the total time will be $(n-1)(n+1)/4$ miliseconds and the waiting time will be $(n-1)/2$ miliseconds. In the second for loop, we can consider four function calls as a set. In the first set, thread 0 gets f(n-1) and f(n-4), and thread 1 gets f(n-2) and f(n-3). We can see that either one uses $2n-5$ miliseconds. In the second set, thread 0 gets f(n-5) and f(n-8), and thread 1 gets f(n-6) and f(n-7). We can see that either one uses $2n-13$ miliseconds. Both threads end at the same time. If we can divide the tasks into groups of four, then the wait time will be 0 milisecond, and the total time will be $n(n-1)/4$ miliseconds.
If we have less than four function calls at the end, there will be a remainder. When $n\%4$ equals 3, meaning that the number of remaining tasks is 2, then we have total time $(n+2)(n-3)/4+2$ miliseconds, and the wait time will be 1. Therefore, when $n\%4$ equals 1, the total time would be $n(n-1)/4+(n-1)(n+1)/4 = (2n^2-n-1)/4$ miliseconds with wait time $(n-1)/2$ miliseconds, and when $n\%4$ equals 3, the total time would be $((n+2)(n-3)/4+2)+(n-1)(n+1)/4 = (2n^2-n+1)/4$ miliseconds with wait time $(n+1)/2$ miliseconds.
When n is even, the first loop will take $(n-1)+(n-3)+...+1 = n^2/4$ miliseconds, and the wait time will be $n/2$ miliseconds. For the second loop, when $n\%4$ equals 2, meaning that the number of remaining tasks is 1, then the total time will be $(n+1)(n-2)/4+1$ miliseconds with wait time 1 milisecond. When $n\%4$ equals 0, meaning that the number of remaining tasks is 3, then the total time for the second loop will be $(n-1)n/4$ miliseconds with wait time 0 milisecond. Therefore if we sum the two loops, when $n\%4$ equals 2, the total time will be $(2n^2-n+2)/4$ miliseconds with wait time $n/2+1$ miliseconds, and when $n\%4$ equals 0, the total time will be $(2n^2-n)/4$ miliseconds with wait time $n/2$ miliseconds.

d) We can use nowait. When n odd, the two threads are given the same amount of function calls and should end together, so the total time should be $(n-1)+...+1 = (n-1)n/2$ miliseconds, and the waiting time should be 0. When n even, thread 0 will get one more function call than thread 1, so the total time will be $((n-1)+...+n/2)+(n/2+...+1) = n^2/2$ miliseconds, and the waiting time will be $n^2/2$ miliseconds.

Problem 2.

Processor: Intel(R) Core(TM) i9-9980HK CPU @ 2.40GHz

p = 2: sequential-scan = 0.262186s parallel-scan = 0.256076s
p = 3: sequential-scan = 0.254410s parallel-scan = 0.207615s
p = 4: sequential-scan = 0.250087s parallel-scan = 0.154154s
p = 5: sequential-scan = 0.253337s parallel-scan = 0.137764s
p = 6: sequential-scan = 0.255503s parallel-scan = 0.129882s
p = 7: sequential-scan = 0.251848s parallel-scan = 0.126926s
p= 8: sequential-scan = 0.253870s parallel-scan = 0.125671s
p= 9: sequential-scan = 0.257199s parallel-scan = 0.128664s
p= 10: sequential-scan = 0.258090s parallel-scan = 0.129190s
p= 15: sequential-scan = 0.266516s parallel-scan = 0.131781s
p = 20: sequential-scan = 0.273825s parallel-scan = 0.131979s
p = 30: sequential-scan = 0.249938s parallel-scan = 0.133453s
p = 50: sequential-scan = 0.258560s parallel-scan = 0.136586s

Problem 3.

Using thread = 2:

——————— N = 10 ———————
Jacobi Method: time = 0.008069
Gauss-Seidel Method: time = 0.003256
——————— N = 100 ———————
Jacobi Method: time = 0.151984
Gauss-Seidel Method: time = 0.153242
——————— N = 200 ———————
Jacobi Method: time = 0.159061
Gauss-Seidel Method: time = 0.142596
——————— N = 400 ———————
Jacobi Method: time = 0.163341
Gauss-Seidel Method: time = 0.162014
——————— N = 1000 ———————
Jacobi Method: time = 0.211874
Gauss-Seidel Method: time = 0.225391
——————— N = 10000 ———————
Jacobi Method: time = 0.684946
Gauss-Seidel Method: time = 0.725793
——————— N = 50000 ———————
Jacobi Method: time = 2.8902
Gauss-Seidel Method: time = 2.61201

Using thread = 5:

——————— N = 10 ———————
Jacobi Method: time = 0.010741
Gauss-Seidel Method: time = 0.004553
——————— N = 100 ———————
Jacobi Method: time = 0.195726
Gauss-Seidel Method: time = 0.195393
——————— N = 200 ———————
Jacobi Method: time = 0.215786
Gauss-Seidel Method: time = 0.195518
——————— N = 400 ———————
Jacobi Method: time = 0.206891
Gauss-Seidel Method: time = 0.202107
——————— N = 1000 ———————
Jacobi Method: time = 0.251678
Gauss-Seidel Method: time = 0.238038

—————— N = 10000 ——————
Jacobi Method: time = 0.580741
Gauss-Seidel Method: time = 0.527823
—————— N = 50000 ——————
Jacobi Method: time = 1.86369
Gauss-Seidel Method: time = 1.37416

Using thread = 10:

—————— N = 10 ——————
Jacobi Method: time = 0.015999
Gauss-Seidel Method: time = 0.007501
—————— N = 100 ——————
Jacobi Method: time = 0.340212
Gauss-Seidel Method: time = 0.339645
—————— N = 200 ——————
Jacobi Method: time = 0.347779
Gauss-Seidel Method: time = 0.335187
—————— N = 400 ——————
Jacobi Method: time = 0.348796
Gauss-Seidel Method: time = 0.331007
—————— N = 1000 ——————
Jacobi Method: time = 0.367323
Gauss-Seidel Method: time = 0.335686
—————— N = 10000 ——————
Jacobi Method: time = 0.651094
Gauss-Seidel Method: time = 0.585486
—————— N = 50000 ——————
Jacobi Method: time = 1.77732
Gauss-Seidel Method: time = 1.20809