IN4026 Assignment A - Prefix/Suffix Minima

He Cheng 4420837

August 30, 2015

1 Introduction

The requirement of this assignment is to design and implement an efficient parallel algorithm with time complexity O(log(N)) in OpenMP and Pthreads that computes the prefix and suffix minima of any given array. If the prefix and suffix minima is calcuted sequentially, the algorithm can be implemented in one simple loop. However, to parallelize the algorithm within the required time complexity, balanced tree method is used.

2 Algorithm Proof

The pseudocode of the algorithm is shown as below.

Algorithm 1 Prefix/Suffix Minima

```
delta ← 1;
for i ← 0 to log(n) do
    delta ← 1;
    for j ← 0 to n − 1 pardo
        prefixMinima(j, j + r − 1);
        suffixMinima(j, j + r − 1);
    end for
end for
```

The balanced tree method is used to implement the parallel algorithm. At the bottom, each thread compute prefix/suffix minima from two values. Next, each thread calculates as twice as many values than the former level. Since they are already placed in order, we only need check the first value from left leaf and the last value from the right leaf. After iterating log(N) times, prefix/suffix minima is obtained.

3 Analysis of the Time-complexity

Since prefixMinima and suffixMinima both have a time-complexity of O(1), the inner loop also O(1) and the outer loop O(log(N)), it is obvious to conclude that the total time-complexity of this algorithm is O(log(N)).

4 Implementation

In sequential implementation, prefix minima is set as the first data of the array, then P is completely iterated to check if P[i] is lower than the current minium. If it is, the minimum is set to P[i]. In the end, we get prefix minima. Suffix minima is calculated similarly. The OpenMP implementation is very close to the sequential implementation but utilizes two pragmas to dynamically schedule the two for loops. The Pthreads implementation is more complicated, needing more control over the program including threads joining. Threads join when all prior threads finish their tasks.

5 Program Test

In this section, the execution time of the algorithm is measured at different problem sizes and with different number of threads, shown below. Since Pthreads implementation does not achieve a good performance, much slower than the sequential implementation, the execution times with sizes larger than 256 are not measured.

Table 1: OpenMP Execution Time (1000 Times)

NSize	Seq	Th01	Th02	Th04	Th08	Th16
4096	0.075300	0.073479	0.050513	0.042773	0.038524	0.042546
8192	0.151440	0.146394	0.101616	0.080394	0.070718	0.074522
16384	0.303173	0.291787	0.150231	0.100376	0.079207	0.111778
32768	0.573729	0.279689	0.268657	0.196986	0.156628	0.174247
65536	1.061863	0.558833	0.526260	0.371395	0.252704	0.287592
131072	1.450889	1.117485	1.032413	0.577824	0.502034	0.519787
262144	2.749761	2.232428	1.740884	1.152492	1.004371	1.042568

Table 2: Pthreads Execution Time (1000 Times)

NSize	Seq	Th01	Th02	Th04	Th08	Th16
32	0.001021	0.881694	0.537683	0.538776	0.845875	0.872255
64	0.001402	1.767919	1.072738	1.052702	1.712651	1.692273
128	0.002418	3.316401	1.919311	1.582565	3.385182	2.822418
256	0.004782	7.131851	4.342998	4.059479	5.521487	4.763613

According to the two table above, the speed-ups of OpenMP and Pthreads are calculated as below.

Table 3: OpenMP Speed-ups

NSize	Th01	Th02	Th04	Th08	Th16
4096	1.025	1.491	1.760	1.955	1.770
8192	1.034	1.490	1.884	2.141	2.032
16384	1.039	2.018	3.020	3.828	2.712
32768	2.051	2.136	2.913	3.663	3.293
65536	1.900	2.018	2.859	4.202	3.692
131072	1.298	1.405	2.511	2.890	2.791
262144	1.232	1.560	2.386	2.738	2.637

Table 4: Pthreads Speep-ups

NSize	Th01	Th02	Th04	Th08	Th16
32	0.001157	0.001899	0.001895	0.001207	0.001171
64	0.000793	0.001306	0.001332	0.000819	0.000828
128	0.000729	0.001260	0.001528	0.000714	0.000857
256	0.000671	0.001101	0.001178	0.000866	0.0010004

According to the above tables, it is obvious to find Pthreads implementation is not only slower than OpenMP implementation, but also much slower than the sequential implementation, which is totally out of expectation. Maybe this is because the overheads in Pthreads implementation including synchronization, context switch, etc. are extremely large. However, the OpemMP implementation achieves relatively good performance, more than 2x speep-up with 8 threads or 16 threads when the input size is large enough.

6 Conclusion

In this assignment, I learn basic knowledge on how to use OpenMP and Pthreads to implement the prefix/suffix minima. The OpenMP implementation achieves good speed-ups while the execution times of Pthreads implementation are unacceptably long, which means Pthreads implementation is not well constructed. As expected, Pthreads would obtain better performance than OpenMP if Pthreads algorithm is well constructed.