IN4026 Assignment C - List Ranking

He Cheng 4420837

August 30, 2015

1 Introduction

The requirement of this assignment is to design and implement an efficient parallel program with time complexity O(log(N)) in OpenMP and PThreads that calculates an array R such that each value R(i) is the distance from node i to the end of the list.

2 Algorithm Proof

The pseudocode for the algorithm is shown below.

Algorithm 1 List Ranking (Pointer Jumping)

```
for i ← 0 to n pardo
  S[i] = P[i]
  if( i != S[i])
     R[i] = 1;
  else
     R[i] = 0;
  end if
  while( S[i] != S[S[i]])
     R[i] = R[i] + R[S[i]];
     S[i] = S[S[i]];
  end while
end for
```

This algorithm first sets R[i] to 1 if S[i] equals to i, otherwise sets R[i] to 0. Then this algorithm checks if S[i] equals to S[S[i]]. If so, it means that node i has already pointed to the root; otherwise, S[i] is set to S[S[i]], the value of the current node's successor; R[i] is updated to R[i] + R[S[i]], the current node's distance value plus the successor's distance value. In this algorithm, each step node i points further away from its original node and always doubles the distance until it reach the node, recorded by R[i]. Therefore, it is obvious that node i will point to the root and R[i] is the distance from its original node to the root. The

correctness of this algorithm is verified through testing the example provided in this assignment and printing the result.

3 Analysis of the Time-complexity

The time-complexity of this algorithm is determined by the number of iteration steps. After each iteration step, the distance of node i to node S[i] doubles. Therefore, we need O(log(N)) iterations before S[i] reaches the root. So the time-complexity is O(log(N)).

4 Implementation

The sequential implementation is translated directly from the pseudocode.

In the OpenMP implementation, the chunk, the ratio of the input size and the number of available threads, is first decided and then XXX pragmas are used to dynamically schedule the for loops. Pthreads implementation is similar to but more complicated than OpenMP implementation.

In the Pthreads version, threads are joined only after prior threads terminate and barriers are used to synchronize data by forcing threads to wait until all threads finish.

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.

Table 1: OpenMP Execution Time (1000 Times)

NSize	Seq	Th01	Th02	Th04	Th08	Th16
4096	0.044329	0.042866	0.025712	0.017212	0.012758	0.015985
8192	0.088380	0.084536	0.048791	0.027428	0.018142	0.020875
16384	0.177233	0.167982	0.089168	0.049056	0.029706	0.031674
32768	0.353974	0.335436	0.164341	0.090003	0.050845	0.079538
65536	0.707395	0.373473	0.340273	0.182713	0.095909	0.086419
131072	1.036582	0.641621	0.322198	0.171851	0.192399	0.170091
262144	1.635429	1.283570	0.988130	0.352067	0.197133	0.171396
524288	3.119224	2.573374	1.843678	0.693382	0.361395	0.341054

Table 2: Pthreads Execution Time (1000 Times)

NSize	Seq	Th01	Th02	Th04	Th08	Th16
4096	0.046375	0.078035	0.068664	0.093170	0.317502	0.608931
8192	0.088664	0.144592	0.090847	0.089400	0.308947	0.647058
16384	0.177061	0.230416	0.169640	0.113345	0.262122	0.581802
32768	0.353967	0.401968	0.252629	0.198537	0.306155	0.593037
65536	0.707819	0.735055	0.427144	0.257234	0.404782	0.599855
131072	1.160769	1.394216	0.777823	0.417100	0.538422	0.617913
262144	1.693534	1.868379	1.368964	0.639471	0.789996	0.995420
524288	3.153417	3.517906	2.673562	1.300786	1.148059	1.508406

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	1.034	1.724	2.575	3.475	2.773
8192	1.045	1.811	3.222	4.872	4.234
16384	1.055	1.988	3.613	5.966	5.596
32768	1.055	2.154	3.933	6.962	4.450
65536	1.894	2.079	3.872	7.376	8.186
131072	1.616	3.217	6.032	5.388	6.094
262144	1.274	1.655	4.645	9.296	9.542
524288	1.212	1.692	4.499	8.631	9.145

Table 4: Pthreads Speed-ups

NSize	Th01	Th02	Th04	Th08	Th16
4096	0.594	0.675	0.498	0.146	0.076
8192	0.613	0.976	0.992	0.287	0.137
16384	0.768	1.044	1.562	0.675	0.304
32768	0.881	1.401	1.783	1.156	0.597
65536	0.963	1.657	2.752	1.749	1.180
131072	0.832	1.492	2.783	2.156	1.879
262144	0.906	1.237	2.648	2.144	1.701
524288	0.896	1.179	2.424	2.747	2.091

Since usually the 8-core computer can handle 8 threads concurrently at most, it is expected that running the algorithm with 8 threads would achieve the largest speed-ups if the input size is large enough. The result of Pthreads implementation is as what I expect. However, in OpenMP implementation, running the algorithm with 16 threads obtains better performance than 8 threads. Maybe it is because this 8-core computer can run 16 threads concurrently. With

the growth of input size, the performance of 16 threads may exceed the performance of 8 threads in Pthreads implementation.

6 Conclusion

In this assignment, OpenMP and Pthreads are used to implement the list ranking. The OpenMP implementation achieves good speed-ups while the speed-ups achieved by Pthreads is relatively limited. However, as expected, Pthreads would obtain better performance than OpenMP if Pthreads algorithm is well constructed.