Parallel Programming Homework 3

All-Pairs Shortest Path (CPU)

CS 107062546 楊仲愷

1. Implementation

(1) Which algorithm do you choose?

Although there are other algorithms to solve all-pairs shortest path, I choose Floyd-Warshall algorithm for my solution. I consider about the time complexity; for slow-all-shortest-paths, the time complexity is $O(V^4)$, and for faster-all-pairs-shortest-paths, it is $O(V^3\log V)$. Then, I also consider about the cases because if E is small enough, it can be faster than Floyd-Warshall; Johnson's algorithm with Fibonacci heap will be $O(V^2\log V + VE)$ and with binary heap will be $O(VE\log V)$, but number of edges won't be that small. Therefore, considering about the suitability for parallel programming, I prefer to choose Floyd-Warshall.

(2) What is the time complexity of your algorithm, in terms of number of vertices V, number of edges E, number of CPUs P?

The time complexity for my algorithm will be $O(V^3)$.

(3) How did you design & generate your test case?

Firstly, I take advantage of the same concept which is making a 2-dimension array (i.e., $|V| \times |V|$) for storing edge weight. Then, I have to generate weight for edge, and this is not that simple because I can't give all the edges weights. Therefore, I set a probability for producing a weight for an edge. Because of this, every edge has a weight means there are two vertices connected by the edge. Finally, I write them into a file.

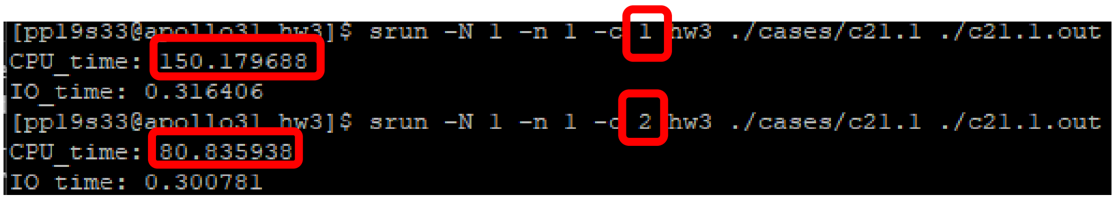
2. Experiment & Analysis

(1) System spec

Run on server.

(2) Strong scalability

According to figures mentioned below, as the number of CPUs increasing, the total amount of computation holds; therefore, the calculation quantity each CPU has to process will decrease, and the computing time will also lower with parallel computing. And the time for processing with only 1 CPU is approximately about 2 times as long as the processing time with 2 CPUs.

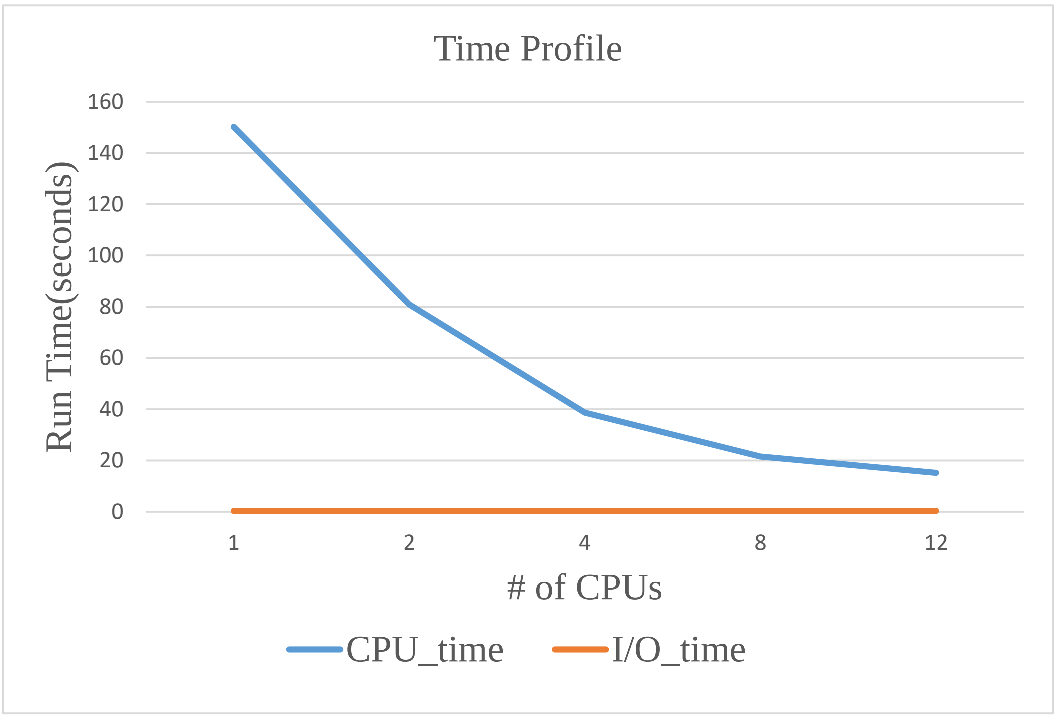


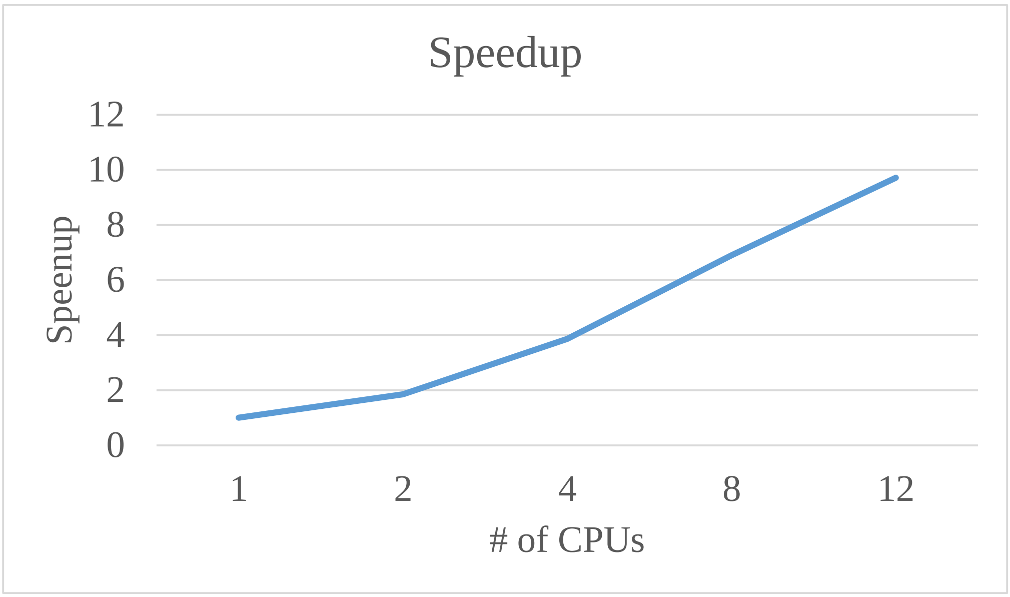
```
[pp19s33@apollo31 hw3]$ srun -N 1 -n 1 -c 1 hw3 ./cases/c21.1 ./c21.1.out
CPU_time: 150.179688
IO_time: 0.316406
[pp19s33@apollo31 hw3]$ srun -N 1 -n 1 -c 2 hw3 ./cases/c21.1 ./c21.1.out
CPU_time: 80.835938
IO_time: 0.300781
```

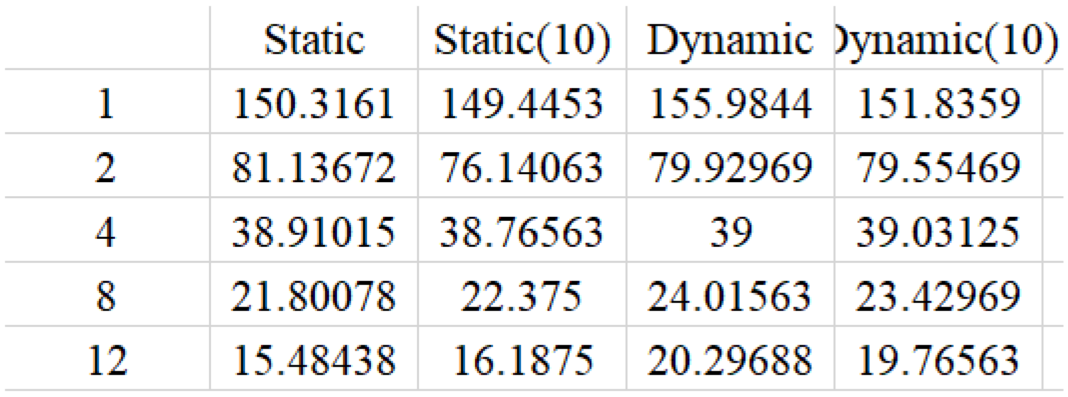
(3) Time profile

(a) Static

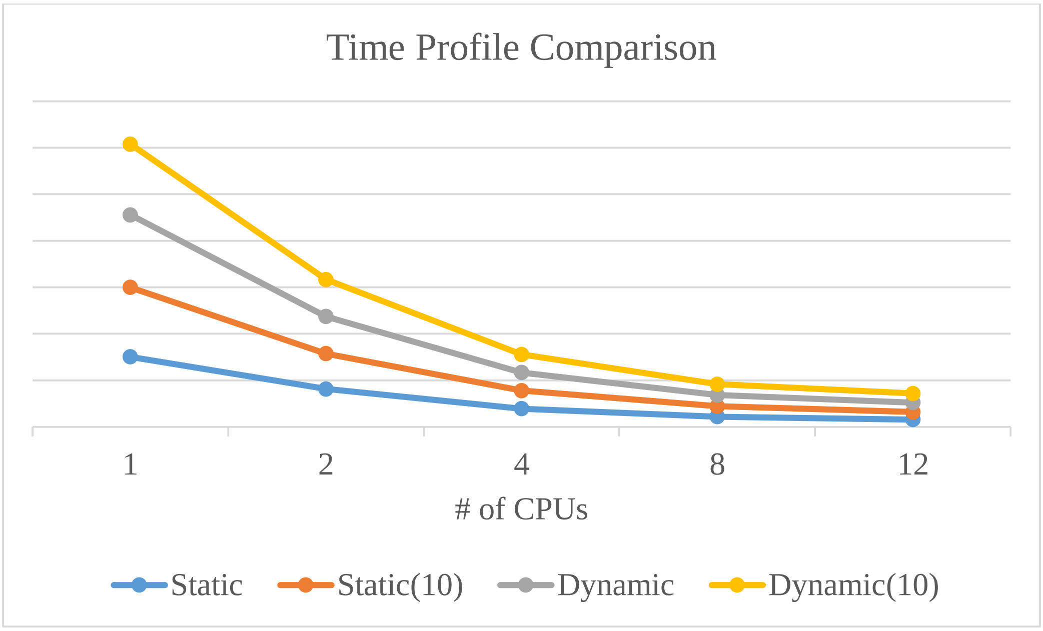
	CPU Time	I/O Time	Total Time	Speedup
1	150.1797	0.136406	**150.3161**	**1**
2	80.83594	0.300781	**81.13672**	**1.852627**
4	38.63281	0.277344	**38.91015**	**3.863159**
8	21.54297	0.257812	**21.80078**	**6.894987**
12	15.21875	0.265625	**15.48438**	**9.707599**







	Static	Static(10)	Dynamic	)ynamic(10)
1	150.3161	149.4453	155.9844	151.8359
2	81.13672	76.14063	79.92969	79.55469
4	38.91015	38.76563	39	39.03125
8	21.80078	22.375	24.01563	23.42969
12	15.48438	16.1875	20.29688	19.76563



3. Experience & Conclusion

(1) What have you learned from this homework?

At first, I found I have to rewrite the make file because it should be executable with OpenMP. Then, I had to think and test some algorithms for time complexity and suitability for parallel programming. Moreover, I also had to write a file to generate test cases.

(2) Feedback

From the figure (i.e., time profile comparison), we can discovery if we choose the dissimilar modes (i.e., static or dynamic) and chunk sizes, we will also get different performance. Therefore, according to this figure, we will get the best one with static scheduling without giving chunk size because OpenMP will support an average distribution and execute them with round-robin. Hence, we can test this kind of parameters to improve our effectiveness.