CMPE 213- Parallel Computing, Spring 2018

Report for pr1

Bijie Qiu

011834433

1. Source code is under the file named “dijkstra_mpi.c” and “Dijkstra_omp.c”

2. For OpenMP, I first printed out the times used for each loop, I found that the for loop copying l[i]=a(s,i) does not take much time, if I parallelize it, it will take more time. Therefore, I left it. In the for loop that tries to find local minimum, I used reduction. Since you also need the index, I kept a short array which keeps the indices for the minimum value from each thread, and then retrieve the index of the global minimum from all threads, it makes it faster than loop through all thousands of numbers to find the index. Last, I did another for loop to update l.

   For MPI, I used one node to read and send to other nodes. It reads a portion, send a portion, reads another portion and send etc. Each node only has one portion of a and l. But the m is synced from all nodes. I use all nodes to find local minimum and root node to find global minimum and broadcast to all. Each node updates its own l and root gather all to l as final output.

Time analysis for OpenMP

threads	1	2	4	8	16	20	28
100	0.000335	0.000453	0.0086	0.001255	0.015594	0.002042	0.016447
500	0.001202	0.0026	0.017982	0.00552	0.008726	0.012555	0.04293
1000	0.003505	0.007063	0.008602	0.011545	0.014441	0.018328	0.057917
5000	0.093513	0.076094	0.072608	0.068995	0.082841	0.099915	0.261178
10000	0.364673	0.245623	0.182754	0.144085	0.15417	0.208655	0.471076

Time analysis for MPI

Using 100 data

100		process				
		1	2	4	8	16
nodes	1	0.0001	0.0001	0.0001	0.0001	0.0001
	2	0.0003	0.0003	0.0003	0.0003	0.0003
	5	0.0006	0.0005	0.0006	0.0005	0.0005
	10	0.0009	0.0008	0.0008	0.0007	0.0007
	serial	0				

Using 1000 data

1000	1	0.004	0.004	0.0041	0.0042	0.004
	2	0.0027	0.0028	0.0027	0.0028	0.0027
	5	0.0026	0.0025	0.0027	0.0025	0.0026
	10	0.0035	0.0036	0.0035	0.0035	0.0036
	serial	0.01				

Using

10000 data

10000	1	0.895	0.8769	0.8748	0.8756	0.9324
	2	0.4527	0.4465	0.4457	0.4457	0.4555
	5	0.186	0.1843	0.1874	0.184	0.1864
	10	0.1216	0.121	0.1202	0.12	0.1246
	serial	0.88				

It looks like my openMP program works well for large dataset. It is interesting that the time first decreases when number of threads increases, but then the time increases again, I guess because the overhead is longer than the time it saved. Also, the time monotonically increases when number of threads increases for small dataset.

For my MPI, it seems that my time decreases using multiple number of nodes, but it performs similar using increasing process, it could be my program is not efficient or my bash code is wrong.

But in general, MPI does not work well if the data size is too small. The communication time is probably too long, it works very well when the dataset is large, for example, 10 nodes can achieve a speed up of 7.