CSC254 Assignment#4
Junfei Liu(jliu137), Pinxin Liu(pliu23)

Speedup Analysis

How to run

We wrote a Makefile for you to run the code. Be sure to enter the src directory, and the following contains the sample codes to run:

Install:

$ make compile

Run the program and get the results for different threads as required:

$ make run
which test threads in range [0, 1, 2, 3, 4, 6, 8, 12, 16, 24, 32, 48]
with a = 0, n = 1000000, and default d

Run extra credits2:

$ make extra2
which test threads in range [0, 1, 2, 4, 8, 16]
and delta in range []
with a = 0, n = 1000000

Run extra credits4:

$ make extra4
which test threads in range [0, 1, 2, 4, 8, 16]
and # of vertices in range [1, 100, 1000, 10000, 100000, 1000000]
with a = 0, default d

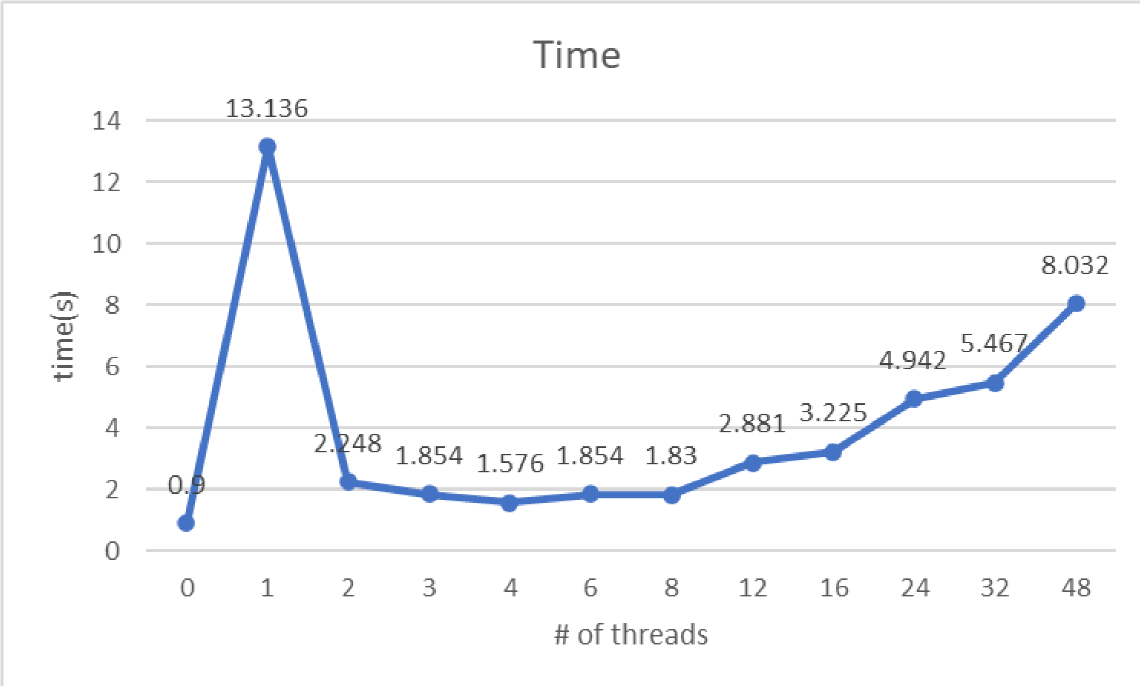
Experiments

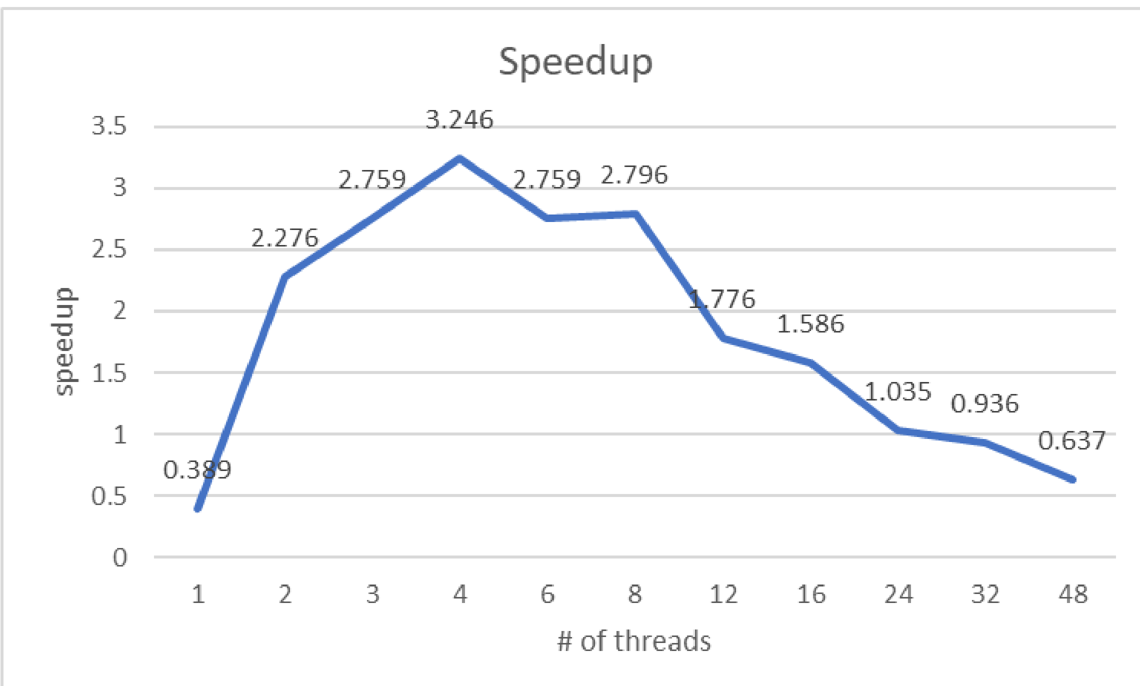
Project Requirement:

We run the benchmark of our delta-stepping algorithm on node2x18a.csug.rochester.edu The following data is obtained by testing threads in range [0, 1, 2, 3, 4, 6, 8, 12, 16, 24, 32, 48] with a = 0, n = 1000000, and default d:

# of Threads	Time(s)	Speedup
0	0.9	N/A
1	13.136	0.389
2	2.248	2.276
3	1.854	2.759
4	1.576	3.246
6	1.854	2.759
8	1.83	2.796
12	2.881	1.776
16	3.225	1.586
24	4.942	1.035
32	5.467	0.936
48	8.032	0.637

Where the speedup is calculated by the time it takes for the starter-code to execute with thread # = 1 (5.116s)/ the time it takes for Delta-stepping of different number of threads.


Time
time(s)
# of threads
0
1
2
3
4
6
8
12
16
24
32
48
0
2
4
6
8
10
12
14
0.9
13.136
2.248
1.854
1.576
1.854
1.83
2.881
3.225
4.942
5.467
8.032


Speedup
speedup
# of threads
1
2
3
4
6
8
12
16
24
32
48
0
0.5
1
1.5
2
2.5
3
3.5
0.389
2.276
2.759
3.246
2.759
2.796
1.776
1.586
1.035
0.936
0.637

We have the following observations:

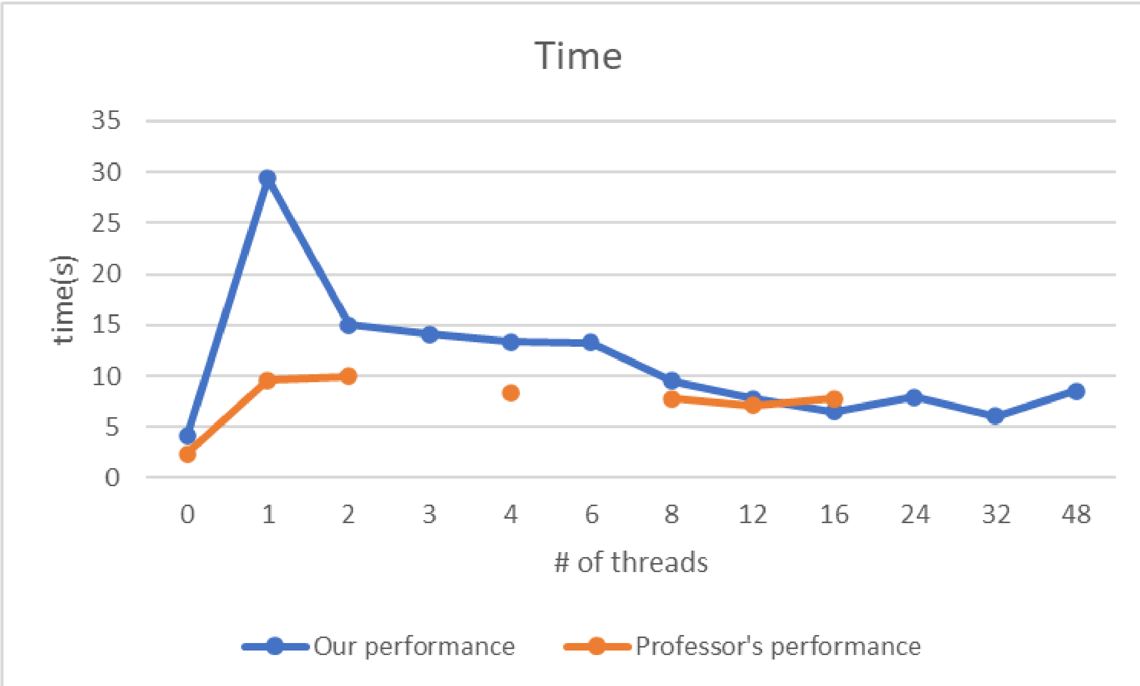
1. The parallelization is generally successful since the time cost with more than two threads are significantly smaller than the time cost with one thread.

2. The speedup increases with threads number increases from 1 to 4, shows no significant change with threads number 6 to 8, and then drops rapidly with more than 12 threads.

We did not achieve the ideal speed-up rate because the delta-stepping algorithm has lots of overhead when accessing message queues and barriers.

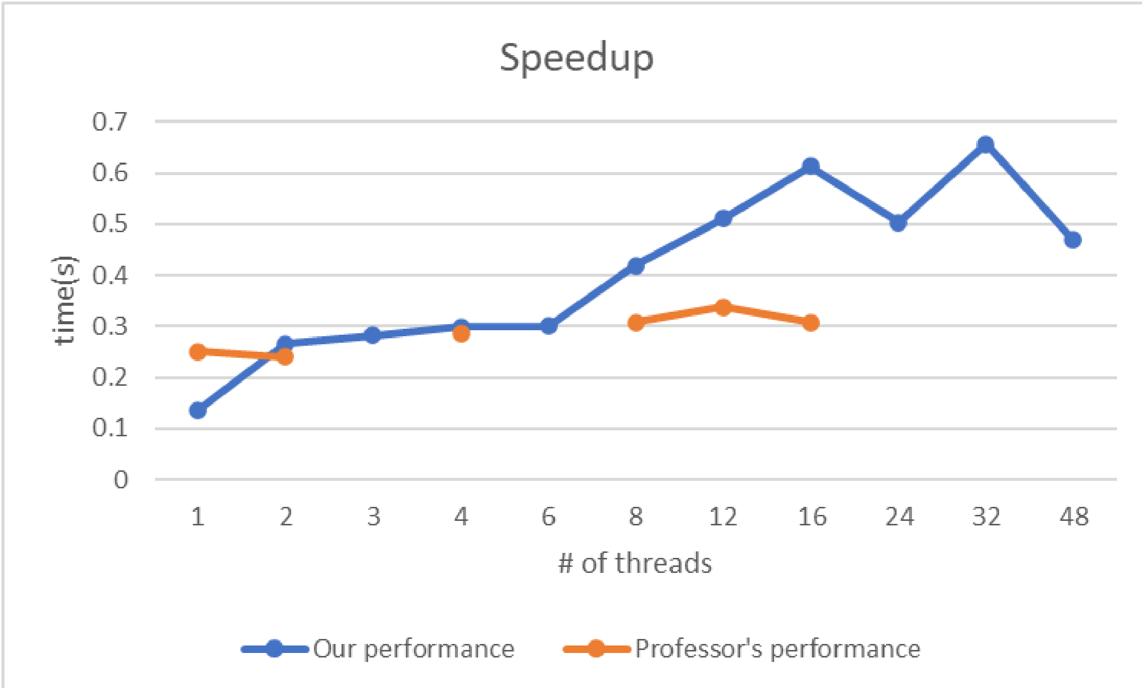
We also tested our delta-stepping algorithm with a = 0, n = 500000, and d = 100 to compare the performance with Professor's Scott's "solution".

# of Threads	Our Time(s)	Professor's Time(s)
0	3.988	2.4
1	29.401	9.6
2	14.976	10
3	14.128	
4	13.322	8.4
6	13.302	
8	9.502	7.8
12	7.813	7.1
16	6.502	7.8
24	7.918	
32	6.068	
48	8.51	



Since starter code cannot set delta, we calculate speedup in terms of the time it takes for Dijkstra's algorithm to execute on respective machines / the time it takes for Delta-stepping of different number of threads.

# of Threads	Our Speedup	Professor's Speedup
1	0.136	0.25
2	0.266	0.24
3	0.282	
4	0.299	0.286
6	0.3	
8	0.42	0.308
12	0.51	0.338
16	0.613	0.308
24	0.504	
32	0.657	
48	0.469	

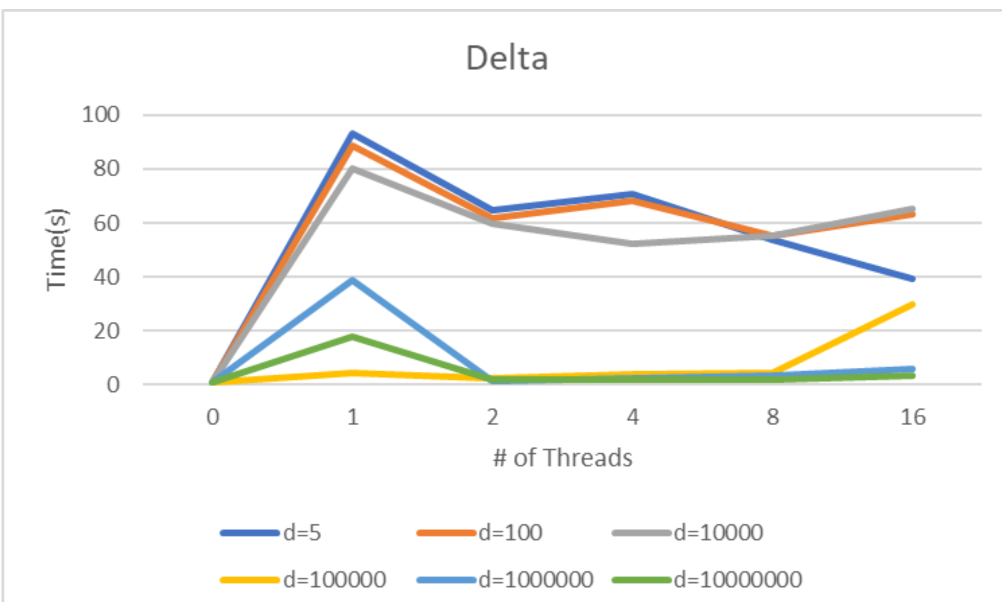


This set of data shows that our code has similar trends of speedup to professor's code, except when thread # = 1. It implies possibilities of improvement.

Extra credit #2:

We added a command-line argument to control the Δ choice and experimented with its impact. The following data is obtained by testing threads in range [0, 1, 2, 4, 8, 16] with a = 0, n = 1000000, and delta in range [5, 100, 10000, 100000, 1000000, 10000000]:

# of threads	0	1	2	4	8	16
D=5	1.13	93.086	64.783	70.917	53.957	39.42
D=100	0.931	88.634	61.888	68.133	55.432	63.269
D=10000	0.984	80.095	59.95	52.209	55.307	65.026
D=100000	0.916	4.583	2.145	4.048	4.535	29.989
D=1000000	0.928	38.866	1.502	2.282	3.427	5.745
D=10000000	0.977	18.029	1.86	1.895	2.032	3.305

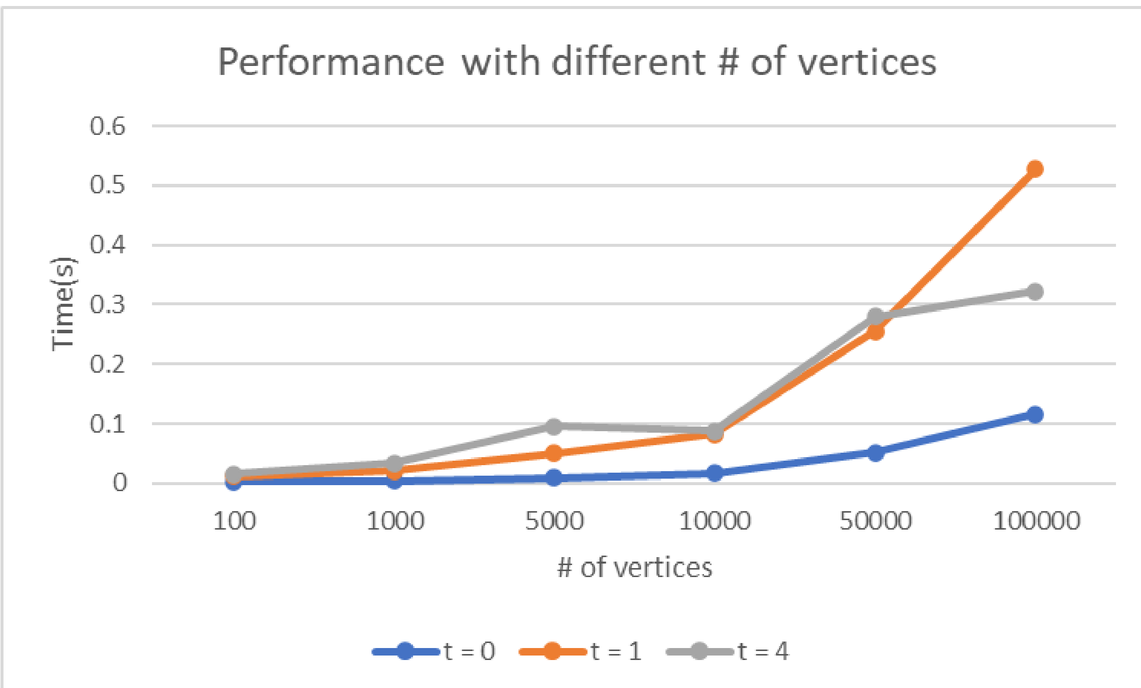


This set of data shows that larger delta generally shows better performance.

Extra credit #4:

For execution on some fixed number of threads, we obtained a set of run time versus the number of vertices. The following data is obtained by testing threads in range [0, 1, 4] with a = 0, n = [100, 1000, 5000, 10000, 50000, 100000]:

# of vertices	100	1000	5000	10000	50000	100000
t = 0	0.003	0.004	0.009	0.017	0.052	0.116
t = 1	0.011	0.02	0.05	0.082	0.255	0.528
t = 4	0.016	0.034	0.096	0.088	0.28	0.323



For execution with sequential method, the run time increases linearly with the increase in the number of vertices. However, the run time varies in a different pattern when executed with multiple threads.