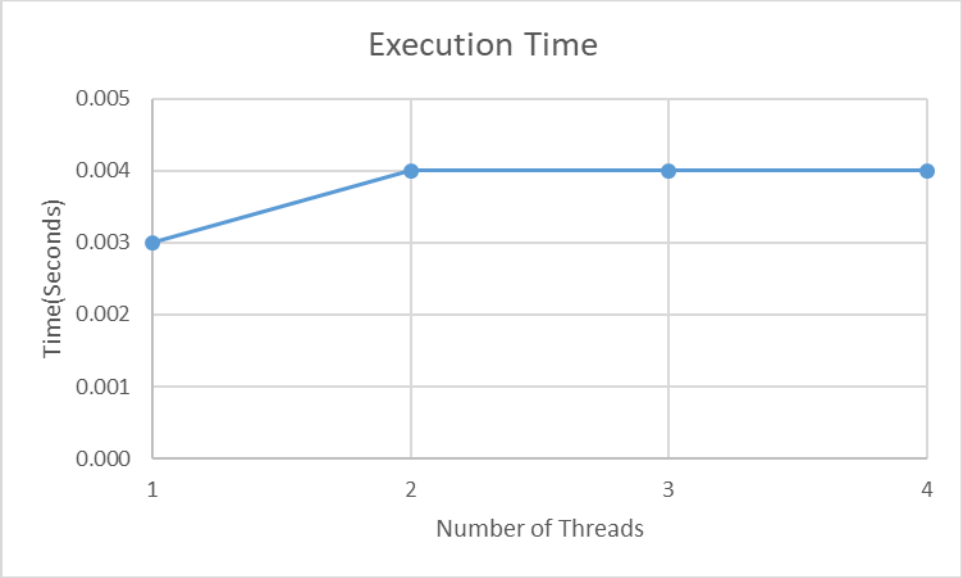
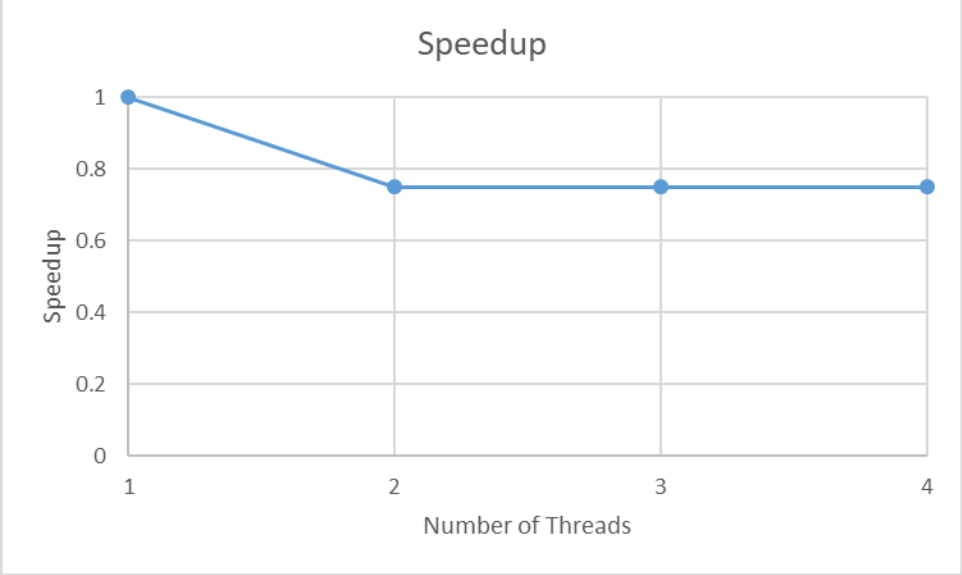
Lab 1: Parallel Travelling Salesman

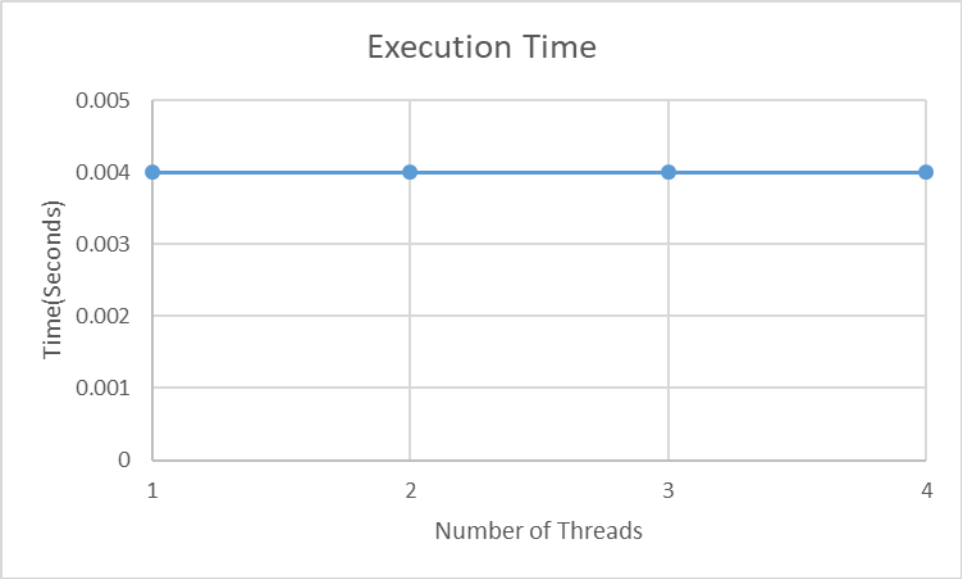
Jiyuan Lu jl11046@nyu.edu

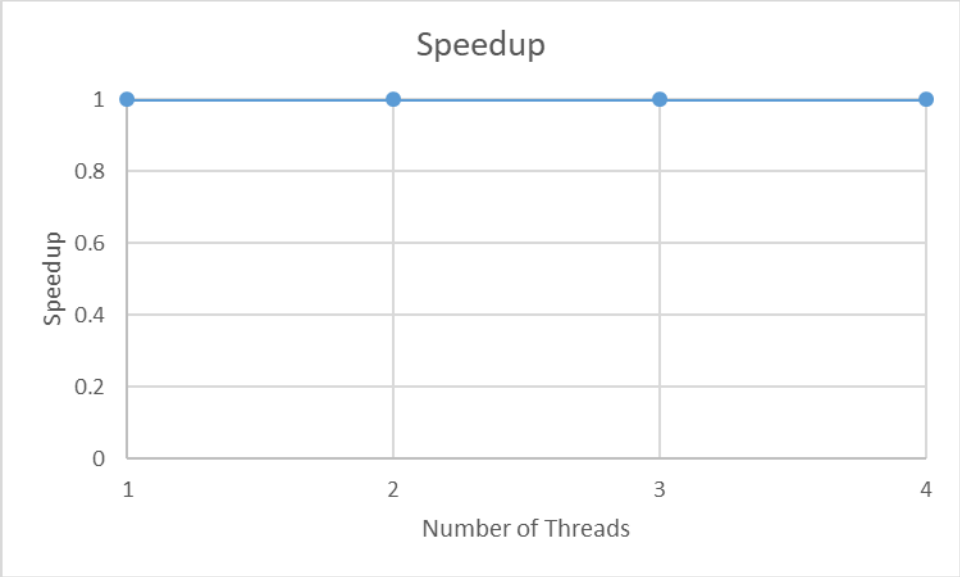
Speedup over single-thread version for a problem of size 4 cities:



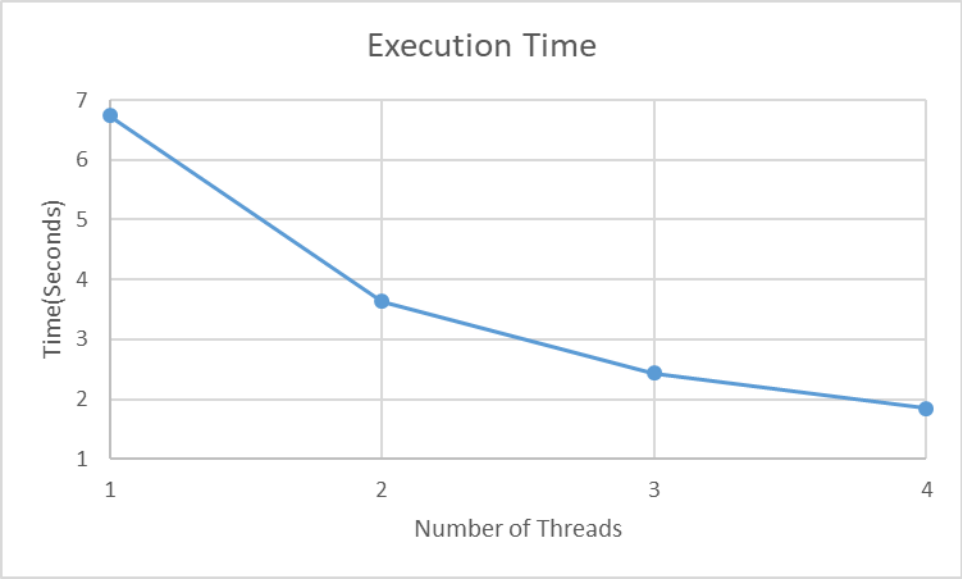


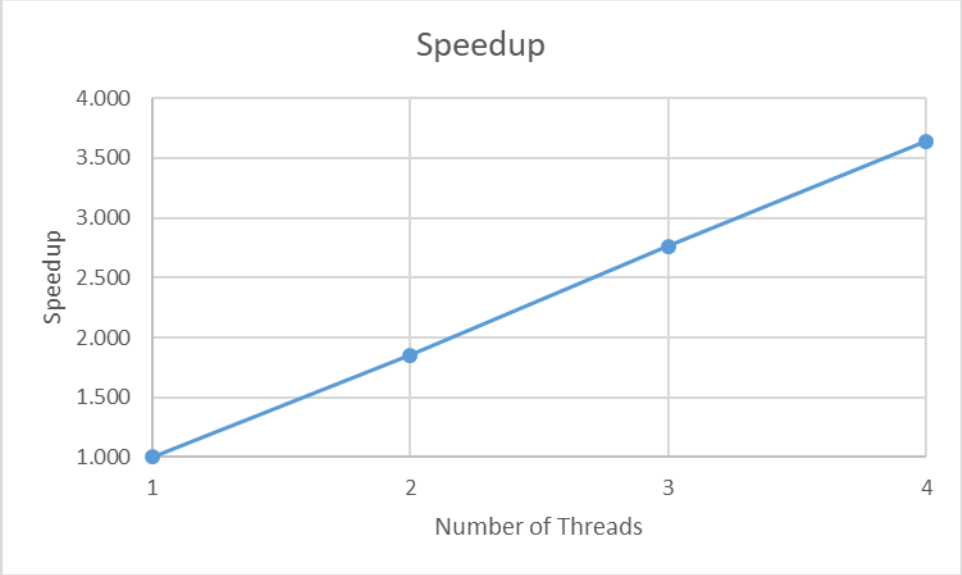
Speedup over single-thread version for a problem of size 8 cities:





Speedup over single-thread version for a problem of size 12 cities:





Analysis of the graphs

a) For 4 cities, there is no speedup. In fact, it takes more time when using multiple threads instead of a single thread. This is because the problem size is too small, and there is not too much to be parallelized (most of the program is the sequential part). And there is more time wasted on things like creating threads and synchronizations than the time saved from parallel processing

   The following shows that the most part of the program is sequential for a problem size of 4:

```
[jl11046@crunchy1 lab1]$ time ./ptsm 4 1 cities4.txt
Best path: 0 1 3 2
Distance: 23
It took 0 ticks (0.000000 seconds) to execute the program.
It took 0 ticks (0.000000 seconds) to execute the sequential part of the program.
It took 0 ticks (0.000000 seconds) to execute the parallel part of the program.
It took 0.000404 seconds to execute the program.
It took 0.000386 seconds to execute the sequential part.
It took 0.000018 seconds to execute the parallel part.

real    0m0.004s
user    0m0.000s
sys     0m0.004s
[jl11046@crunchy1 lab1]$
```

b) For 8 cities, there is no speedup. In fact, it takes about the same time when using multiple threads as using a single thread. This is because the problem size is still small. And the gain from parallel processing are close to the loss incurred from things like creating threads and synchronizations.

The following shows that the sequential part and the parallel part of the program are close for a problem size of 8:

```
[jl11046@crunchy1 lab1]$ time ./ptsm 8 1 cities8.txt
Best path: 0 3 1 7 4 5 2 6
Distance: 26
It took 0 ticks (0.000000 seconds) to execute the program.
It took 0 ticks (0.000000 seconds) to execute the sequential part of the program.
It took 0 ticks (0.000000 seconds) to execute the parallel part of the program.
It took 0.001300 seconds to execute the program.
It took 0.000635 seconds to execute the sequential part.
It took 0.000665 seconds to execute the parallel part.

real    0m0.004s
user    0m0.001s
sys     0m0.003s
```

c) For 12 cities, there is significant speedup. The speedup is close to the number of threads used. (1.852 for 2 threads, 2.764 for 3 threads, and 3.640 for 4 threads). We can also calculate the efficiency of the program (0.926 for 2 threads, 0.921 for 3 threads, and 0.910 for 4 threads). This indicates that the problem is strongly scalable for a problem size of 12. This is because the problem size is now big enough and the most part of the program is parallelizable.

The following shows that the most part of the program is parallelizable for a problem size of 12:

```
[jl11046@crunchy1 lab1]$ time ./ptsm 12 1 cities12.txt
Best path: 0 1 2 8 10 11 9 3 7 4 5 6
Distance: 44
It took 6660000 ticks (6.660000 seconds) to execute the program.
It took 0 ticks (0.000000 seconds) to execute the sequential part of the program.
It took 6660000 ticks (6.660000 seconds) to execute the parallel part of the program.
It took 6.662848 seconds to execute the program.
It took 0.000673 seconds to execute the sequential part.
It took 6.662175 seconds to execute the parallel part.

real    0m6.670s
user    0m6.662s
sys     0m0.004s
[jl11046@crunchy1 lab1]$
```

Data

- For execution time, I conducted 10 experiments for each (n, t) pair where n is the number of cities and t is the number of threads. Then I took the average of the 10 experiments.

My Parallel (1 thread)

Problem Size (n) / Time(seconds)	Weight	Mean	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8	Test 9	Test 10
1	0	0.0031	0.004	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003
2	6	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003
3	11	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003
4	23	0.0031	0.003	0.004	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003
5	25	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003
6	19	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003
7	32	0.0033	0.003	0.006	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003
8	26	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004
9	47	0.0091	0.009	0.009	0.009	0.009	0.009	0.01	0.009	0.009	0.009	0.009
10	30	0.0568	0.056	0.057	0.056	0.056	0.057	0.058	0.058	0.057	0.056	0.057
11	39	0.5765	0.576	0.577	0.574	0.574	0.576	0.576	0.58	0.574	0.581	0.577
12	44	6.7339	6.542	6.643	6.664	7.453	6.695	6.642	6.765	6.587	6.636	6.712

My Parallel (2 threads)

Problem Size (n) / Time(seconds)	Weight	Mean	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8	Test 9	Test 10
1	0	0.0041	0.005	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004
2	6	0.0041	0.004	0.005	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004
3	11	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004
4	23	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004
5	25	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004
6	19	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004
7	32	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004
8	26	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004
9	47	0.007	0.007	0.007	0.007	0.007	0.007	0.007	0.007	0.007	0.007	0.007
10	30	0.034	0.034	0.034	0.034	0.034	0.034	0.034	0.035	0.034	0.033	0.034
11	39	0.2902	0.292	0.292	0.29	0.292	0.289	0.288	0.289	0.289	0.289	0.292
12	44	3.6356	3.626	3.646	3.619	3.627	3.694	3.613	3.625	3.65	3.622	3.634

My Parallel (3 threads)

Problem Size (n) / Time(seconds)	Weight	Mean	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8	Test 9	Test 10
1	0	0.0042	0.005	0.005	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004
2	6	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004
3	11	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004
4	23	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004
5	25	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004
6	19	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004
7	32	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004
8	26	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004
9	47	0.006	0.006	0.006	0.006	0.006	0.006	0.006	0.006	0.006	0.006	0.006
10	30	0.0222	0.022	0.023	0.022	0.023	0.022	0.022	0.022	0.022	0.022	0.022
11	39	0.2352	0.233	0.235	0.237	0.232	0.231	0.232	0.235	0.23	0.233	0.254
12	44	2.4361	2.43	2.44	2.433	2.436	2.433	2.445	2.44	2.43	2.432	2.442

My Parallel (4 threads)												
Problem Size (n) / Time(seconds)	Weight	Mean	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8	Test 9	Test 10
1	0	0.0041	0.005	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004
2	6	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004
3	11	0.0041	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.005	0.004	0.004
4	23	0.0039	0.004	0.004	0.004	0.004	0.003	0.004	0.004	0.004	0.004	0.004
5	25	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004
6	19	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004
7	32	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004
8	26	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004
9	47	0.0054	0.006	0.006	0.006	0.006	0.005	0.005	0.005	0.005	0.005	0.005
10	30	0.0224	0.023	0.023	0.022	0.022	0.023	0.022	0.022	0.022	0.023	0.022
11	39	0.1785	0.177	0.177	0.187	0.188	0.176	0.176	0.176	0.177	0.176	0.175
12	44	1.8503	1.843	1.832	2.021	1.831	1.821	1.833	1.832	1.825	1.835	1.83

Some Notes

- The program was built on crunchy1.cims.nyu.edu and compiled with gcc-9.2.
- Problem about the clock() function:
    - The clock() function as given in the Hint does not work well because the problem size is too small. It always gives me 0 ticks estimation for the sequential execution time. Therefore, I used omp_get_wtime() instead to get a finer-grained estimation of both the sequential and the parallel execution time of the program.
- Advantages over the reference implementation:
    - My program, even when executed sequentially, is much faster than the reference program when the problem size is above 10.
    - The reference program has invalid output for distance when the problem size is 1. I fixed the problem by outputting 0 for the distance when the problem size is 1.
- How I iterated on building the program:
    - I first implemented a serial version, then built upon it to get a parallelized version.
    - At first, there was a bug in my parallel program because I did not create private variables for each thread, and all threads were competing for the shared variable. The program was essentially sequentially executed, and the performance was even worse than a sequential version.
-