Programming Assignment 2

STUDENT ID: 0660004

NAME: 王順興

2-opt

Concept

Check if the route can be shortened by reversing a segment of route; if so, reverse the route

Make it faster

- A reversed segment has the same route length, the total route distance is changed at the start and the end of the segment
- If every segment on a route is only processed by one thread, threads can work on the route simultaneously

Route Splitting

To process 2opt operation on every route:

```
for (depth = 1; depth < num_city - 1; ++depth) {
    for (start = 1; start < num_city - depth; ++start) {
        2opt_swap(start, start + depth);
    }
```

Idea: Split `start` among threads, and avoid "contention". The splitting is fair because each thread is processing segments with the same length.

```
for (depth = 1; depth < num_city - 1; ++depth) {
    #pragma omp parallel for schedule(static, chunk)
    for (start = 1; start < num_city - depth; ++start) {
        2opt_swap(start, start + depth);
    }
```

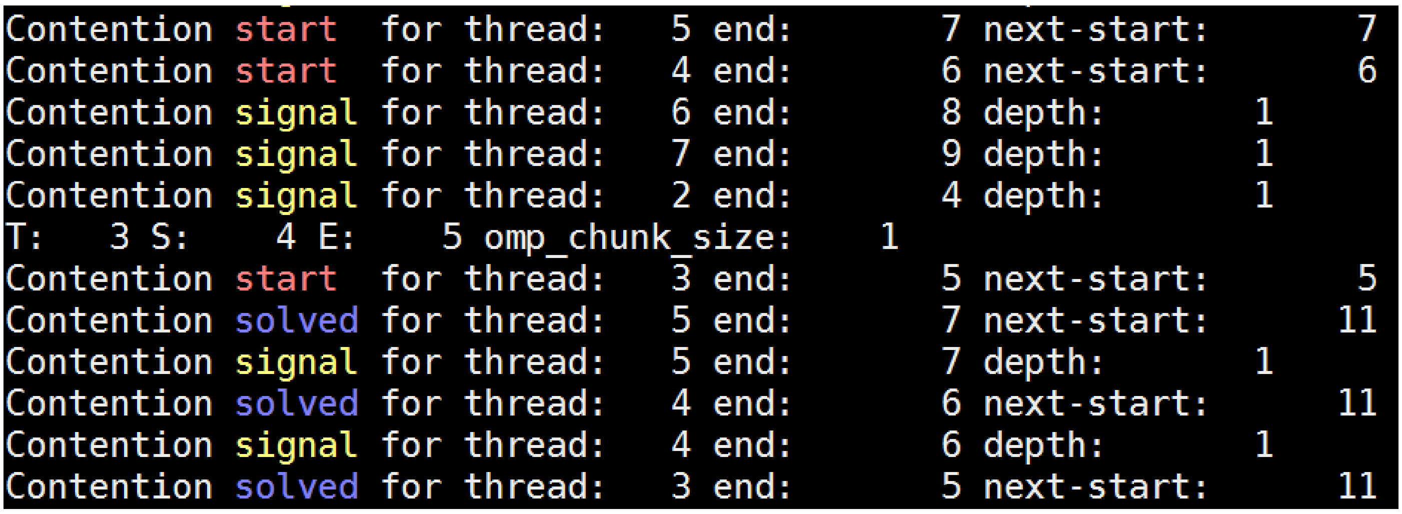
Contention

Contention is where two or more thread works on neighboring/crossed segment, and race condition may happen.

To avoid this, `start` is splitted in chunks, so contention will only happen between two consecutive threads.
With contention only happening between two consecutive threads, it's easy to check if contention happens, and the thread with the smaller thread_num can simply wait until the next thread moves on:

```
In thread(i):
while (segment_start_index[i + 1] <= segment_end_index[i] + 1)
    wait;
```

Example of Contention and Untangling:



```
Contention start  for thread:   5 end:       7 next-start:      7
Contention start  for thread:   4 end:       6 next-start:      6
Contention signal for thread:   6 end:       8 depth:       1
Contention signal for thread:   7 end:       9 depth:       1
Contention signal for thread:   2 end:       4 depth:       1
T:   3 S:    4 E:    5 omp_chunk_size:    1
Contention start  for thread:   3 end:       5 next-start:      5
Contention solved for thread:   5 end:       7 next-start:     11
Contention signal for thread:   5 end:       7 depth:       1
Contention solved for thread:   4 end:       6 next-start:     11
Contention signal for thread:   4 end:       6 depth:       1
Contention solved for thread:   3 end:       5 next-start:     11
```

Thread 5 waits for Thread 6

Thread 4 waits for Thread 5

Thread 6 moves on, Thread 5 can move on now

Thread 3 waits for Thread 4

Thread 5 moves on, Thread 4 can move on now

Thread 4 moves on, Thread 3 can move on now

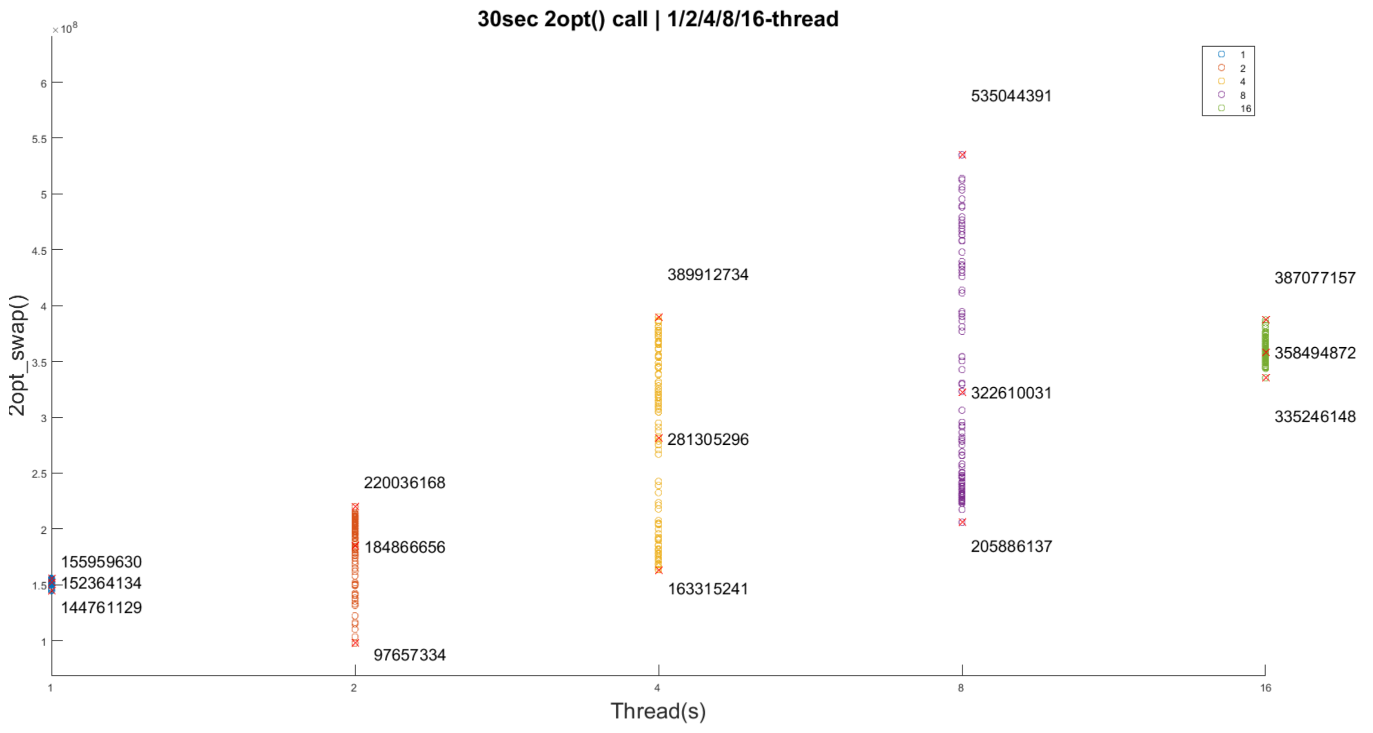
Thread 3 moves on

Signal: A thread finishes the chunk it's assigned to

2. Performance analysis with 1, 2, 4, 8, 16 thread(s)

How may 2-opt comparison do you perform within 30 seconds?

Plotting 100 test11 tests for 1/2/4/8/16 threads:



2opt_swap() in 30s for 1/2/4/8/16 threads			
	Min.	Avg.	Max.
1	144761129	152364134	155959630
2	97657334	184866656	220036168
4	163315241	281305296	389912734
8	205886137	322610031	535044391
16	335246148	358494872	387077157

For reasons I can’t find out, 2/4/8 thread’s performance varies a lot; while 1/16 threads performance are relatively consistent. No contention happens in any case, so the only overhead for multi-threading is to check if contention happens.

Even when removing operations in 2opt_swap(), leaving only counter, the situation persists.

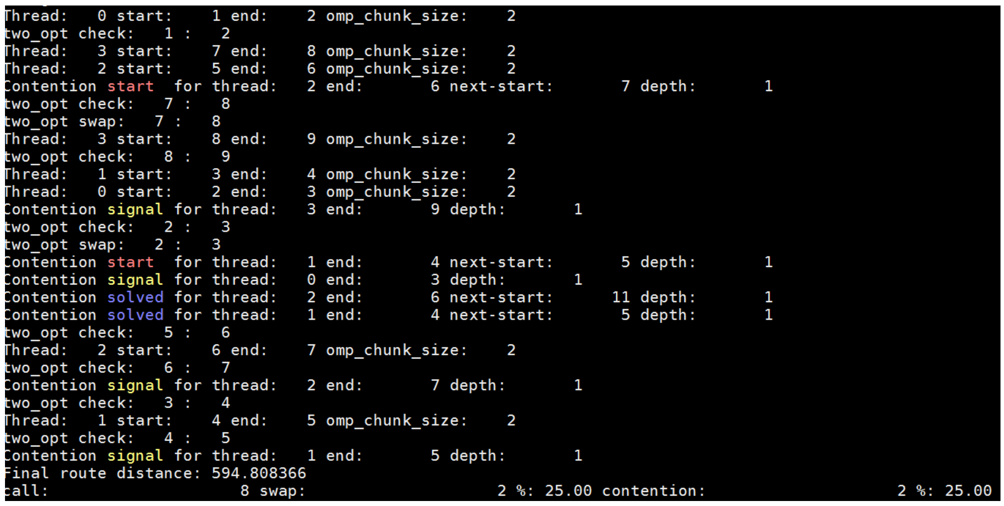
3. Compare your results with the un-parallel version of your code (with test8)

Do you use any random functions in your code? **No**

If not, then your answer should be equal. Otherwise, there are **<u>race condition</u>** unhandled in your code.

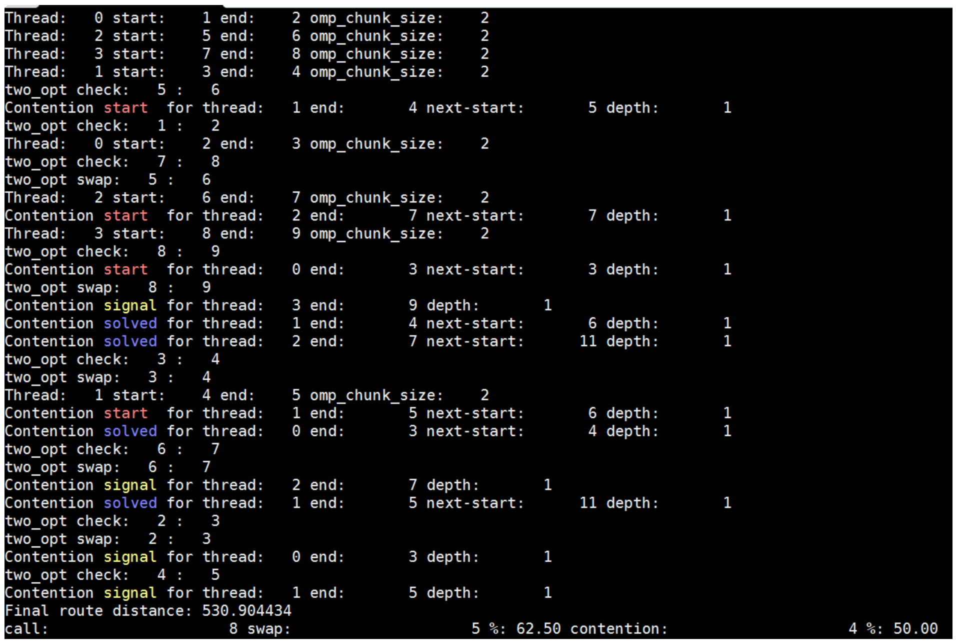
The distance is not equal because in my program, a thread can process the part of route that other thread is working on, even though there's no race condition.

Log 1: 4- thread / Test 0 / 2opt_swap(): 8 / Final distance: 594



```
Thread:   0 start:    1 end:    2 omp_chunk_size:    2
two_opt check:   1 :   2
Thread:   3 start:    7 end:    8 omp_chunk_size:    2
Thread:   2 start:    5 end:    6 omp_chunk_size:    2
Contention start  for thread:   2 end:       6 next-start:       7 depth:       1
two_opt check:   7 :   8
two_opt swap:   7 :   8
Thread:   3 start:    8 end:    9 omp_chunk_size:    2
two_opt check:   8 :   9
Thread:   1 start:    3 end:    4 omp_chunk_size:    2
Thread:   0 start:    2 end:    3 omp_chunk_size:    2
Contention signal for thread:   3 end:       9 depth:       1
two_opt check:   2 :   3
two_opt swap:   2 :   3
Contention start  for thread:   1 end:       4 next-start:       5 depth:       1
Contention signal for thread:   0 end:       3 depth:       1
Contention solved for thread:   2 end:       6 next-start:      11 depth:       1
Contention solved for thread:   1 end:       4 next-start:       5 depth:       1
two_opt check:   5 :   6
Thread:   2 start:    6 end:    7 omp_chunk_size:    2
two_opt check:   6 :   7
Contention signal for thread:   2 end:       7 depth:       1
two_opt check:   3 :   4
Thread:   1 start:    4 end:    5 omp_chunk_size:    2
two_opt check:   4 :   5
Contention signal for thread:   1 end:       5 depth:       1
Final route distance: 594.808366
call:                      8 swap:                    2 %: 25.00 contention:                  2 %: 25.00
```

Log 2: 4- thread / Test 0 / 2opt_swap(): 8 / Final distance: 530



```
Thread:   0 start:    1 end:    2 omp_chunk_size:    2
Thread:   2 start:    5 end:    6 omp_chunk_size:    2
Thread:   3 start:    7 end:    8 omp_chunk_size:    2
Thread:   1 start:    3 end:    4 omp_chunk_size:    2
two_opt check:   5 :   6
Contention start  for thread:   1 end:       4 next-start:       5 depth:       1
two_opt check:   1 :   2
Thread:   0 start:    2 end:    3 omp_chunk_size:    2
two_opt check:   7 :   8
two_opt swap:   5 :   6
Thread:   2 start:    6 end:    7 omp_chunk_size:    2
Contention start  for thread:   2 end:       7 next-start:       7 depth:       1
Thread:   3 start:    8 end:    9 omp_chunk_size:    2
two_opt check:   8 :   9
Contention start  for thread:   0 end:       3 next-start:       3 depth:       1
two_opt swap:   8 :   9
Contention signal for thread:   3 end:       9 depth:       1
Contention solved for thread:   1 end:       4 next-start:       6 depth:       1
Contention solved for thread:   2 end:       7 next-start:      11 depth:       1
two_opt check:   3 :   4
two_opt swap:   3 :   4
Thread:   1 start:    4 end:    5 omp_chunk_size:    2
Contention start  for thread:   1 end:       5 next-start:       6 depth:       1
Contention solved for thread:   0 end:       3 next-start:       4 depth:       1
two_opt check:   6 :   7
two_opt swap:   6 :   7
Contention signal for thread:   2 end:       7 depth:       1
Contention solved for thread:   1 end:       5 next-start:      11 depth:       1
two_opt check:   2 :   3
two_opt swap:   2 :   3
Contention signal for thread:   0 end:       3 depth:       1
two_opt check:   4 :   5
Contention signal for thread:   1 end:       5 depth:       1
Final route distance: 530.904434
call:                      8 swap:                    5 %: 62.50 contention:                  4 %: 50.00
```

Both example the execute 2opt_swap(i, i+1) once, so 2opt_swap() is called 8 times each (10 node with the first node untouched). Yet the route changes and final distance are different due to the randomness in thread job scheduling by OS.

4. Record your distance every 30 seconds with **test11**

Please observe if the distance improves within 10 minute.

12 threads, cherry picking the best in 10 test

Time	Distance
0m 30s	551164176631
1m 00s	387634583985
1m 30s	311517458361
2m 00s	268252542044
2m 30s	241759607366
3m 00s	221083873462
3m 30s	205001031814
4m 00s	189691735775
4m 30s	179632732100
5m 00s	170423093091
5m 30s	161875623748
6m 00s	154019443817
6m 30s	148311675690
7m 00s	142885325261
7m 30s	138252766707
8m 00s	133360874930
8m 30s	129622332157
9m 00s	126164889332
9m 30s	123165547789
10m 00s	120124885742

5. Discussion

- If there’s any race condition in your code, please describe how you handle them?

See *Contention*

- What is the most difficult part of this assignment?

Dealing with the edge cases regarding contention and debugging the problem mentioned in part 4.

- What is the bottle neck of your program? Do you think that it can be improved?

There’s bottleneck caused contention checking and because the parallel region is in the inner loop, a thread has to wait for all other thread to finish the work so it can move on.

The later can be improved by modifying the contention checking from

```
While(segment_start_index[i + 1] <= segment_end_index[i] + 1)
```

to

```
while(segment_start_index[i+1 ... N] <= segment_end_index[i] + 1)
```

But that means more overhead so the former problem become worse.

- Can the number of thread be more than the number of core, please justify your answer?

Yes, the OS will map logical thread to physical thread/core

- Which do you prefer, pthread or OpenMP, why? What are their pros and cons?

If I want to make stuff fast: OpenMP, otherwise Pthread. Pthread is easier to customize but more complicated to write; OpenMP is easier to write but the implicit stuff will trip you up.