Shonte Amato-Grill
Parallel Computing
April 22, 2014

Lab 2: Open MP Travelling Salesman

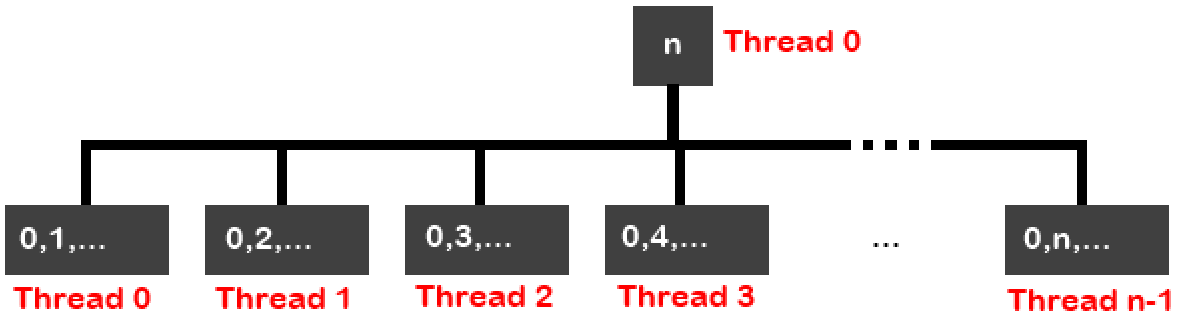
How is it Parallelized?

Two algorithms were tried, both very similar, but one with memory improvements.

1. **Brute Force**
   This algorithm simply generates *all* possible permutations (starting with 0) of the given cities, storing them in an array of size *(number_of_cities-1)!*. It then loops over them, checking each permutation's path distance against the previously found smallest. It overwrites the old shortest path if the new one is of smaller distance.
   Parallelism was achieved by using `#pragma omp for` with several different thread configurations. After a while, this algorithm was dropped, due to exorbitant overuse of memory.
2. **One By One**
   Obviously enough, this required a more clever approach. Instead of pre-generating all possible permutations, we have $(\text{number\_of\_cities} - 1)$ threads, each generating one permutation at a time, starting with $\{0,\text{thread\_id} + 1, \dots\}$. In this way, each thread can lexicographically generate all possible permutations beginning with its distinct prefix.

   Step 1: Calculate the length of the current permutation.
   Step 2: If the current length is smaller than the previously found path, store it (critical section).
   Step 3: Generate the next, lexicographic permutation, and loop back to *Step 1*.

   So, the parallelism can be represented thusly:



The Input

The input file for this program *must* follow this format:

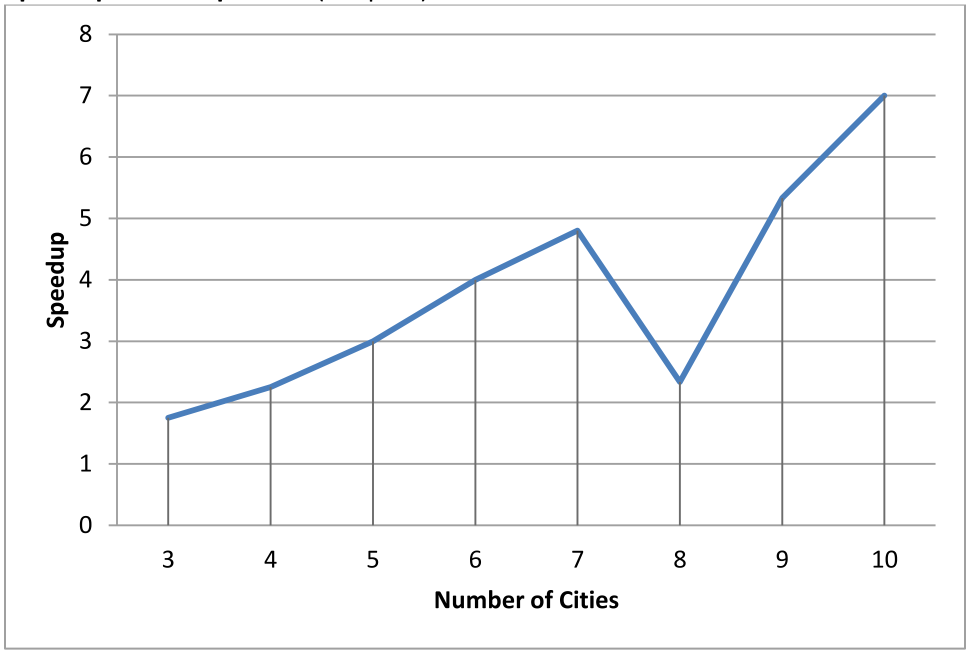
```
Num_of_cities
Distance_matrix
```

So, for instance, with 3 cities:

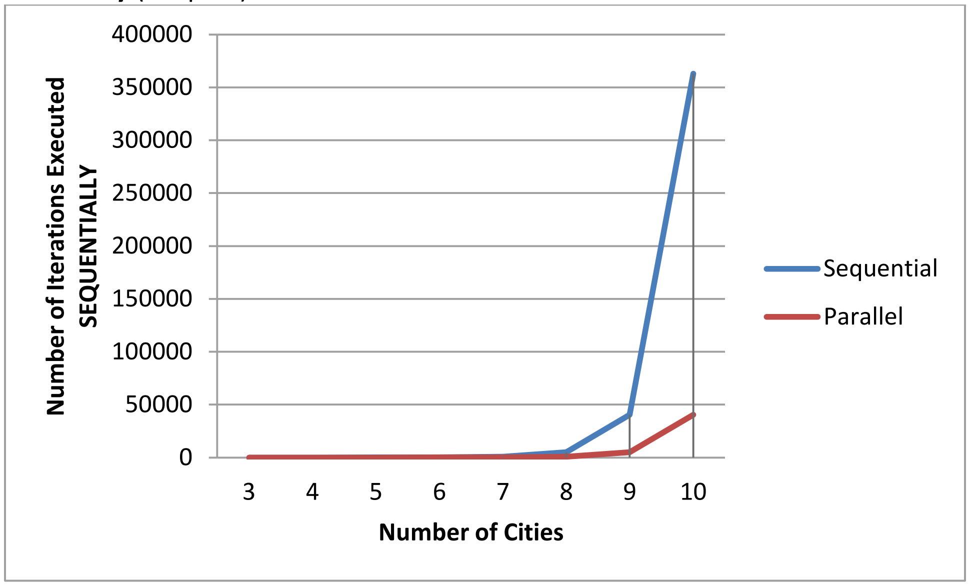
```
3
0 2 2
2 0 3
2 3 0
```

Analysis

Speedup over Sequential (Graph 1)



Scalability (Graph 2)



Rather evidently, the speedup on this parallel algorithm when compared to its pseudo-sequential sibling is rather good. Since, in this implementation, there are always $(\text{number\_of\_cities} - 1)$ threads, and $(\text{number\_of\_cities} - 1)!$ possible paths (permutations), $\lim_{n\to\infty} \frac{(\mathrm{n}-1)!}{n-1} = \infty$. Or, the greater the number of cities, the more iterations required (Graph 2), and thusly, the more effectiveness gained by the parallel implementation and the greater the speedup over the sequential version (Graph 1).
There is a sudden drop in speedup at 8 cities, but this could easily be caused by server load, and other uncontrolled variables.