Multicore Processors: Architecture & Programming, Programming Assignment - 1

Tulsi Jain

1 Travelling Salesman Problem Statement

There are a group of n cities (from 0 to n-1). The traveling salesman must start from city number 0, visit each city once, and does not have to come back to the initial city. Distance between each city (cities are fully connected to each other) is given. Find a shortest path with shortest distance. Note that there may be several paths that satisfy the above conditions, you just need to find one of them.

2 Approach

Travelling salesman can be approached in multiple way. Dynamic programming algorithms is correct best known algorithms in sequential running. But in parallel programming using same algorithms as sequential running is not always a good idea as to get benefit in parallel programming threads should execute independent of each other but in dynamic programming information is pass over to next iteration hence this algorithm is not a good candidate to run in parallel. Best algorithm in parallel programming for travelling salesman is based on brute force approach, examine all (n - 1)! possible paths and select path with minimum path. One simple pruning technique would provide a little speed up is to explore path until it does not exceed the best known path distance. Multiple experiments have been run using three different approached to determine the best algorithms. Among-st all approaches **iterative based** approach is observed to perform best (least time to solve the problem).

3 Experiments

Three experiment were executed but one based iterative approach is the fastest. Below are the detailed description of all three approach mentioned in-order of worst to best.

3.1 Memory based

In this approach, all possible paths are store in two-dimensional array by creating all permutations of n-1 cities before looking for best possible path. Then all paths are evaluated using iterative approach. This algorithm took 0.491 seconds to complete for 10 cities when run on 10 threads in parallel and turned out to worst performed approach. One possible reason is two-pass run first to create all possible paths and to store possible paths in memory and second is to look for minimum path distance. Writing and reading in memory led to its worst performance.

3.2 Recursion based

This is different from memory based approach in a way. Instead of storing a path in two-dimensional array it is evaluated right away be it is better than best known path then best known path is updated otherwise this path is discarded. This algorithms took 20 This performed better than memory based approach but it still taking much time and there is still a scope for improvement.

3.3 Iterative based (best approach)

This approach is best performed approach and is implemented by improvements on top of memory based and recursion based. First, this uses iteration to permute over all the possibles paths instead of recursion which is used in memory and recursion based approaches. Like in last approach, path are evaluated right away and best known path are updated only if this better has minimum distance then best known path. This approach took 0.08 seconds to complete for 10 cities when run on 10 threads in parallel.

No cities	Memory based	Recursion based	Iterative based (best approach)
5	0.014	0.068	**0.044**
10	0.300	0.110	**0.080**

3.4 Graph

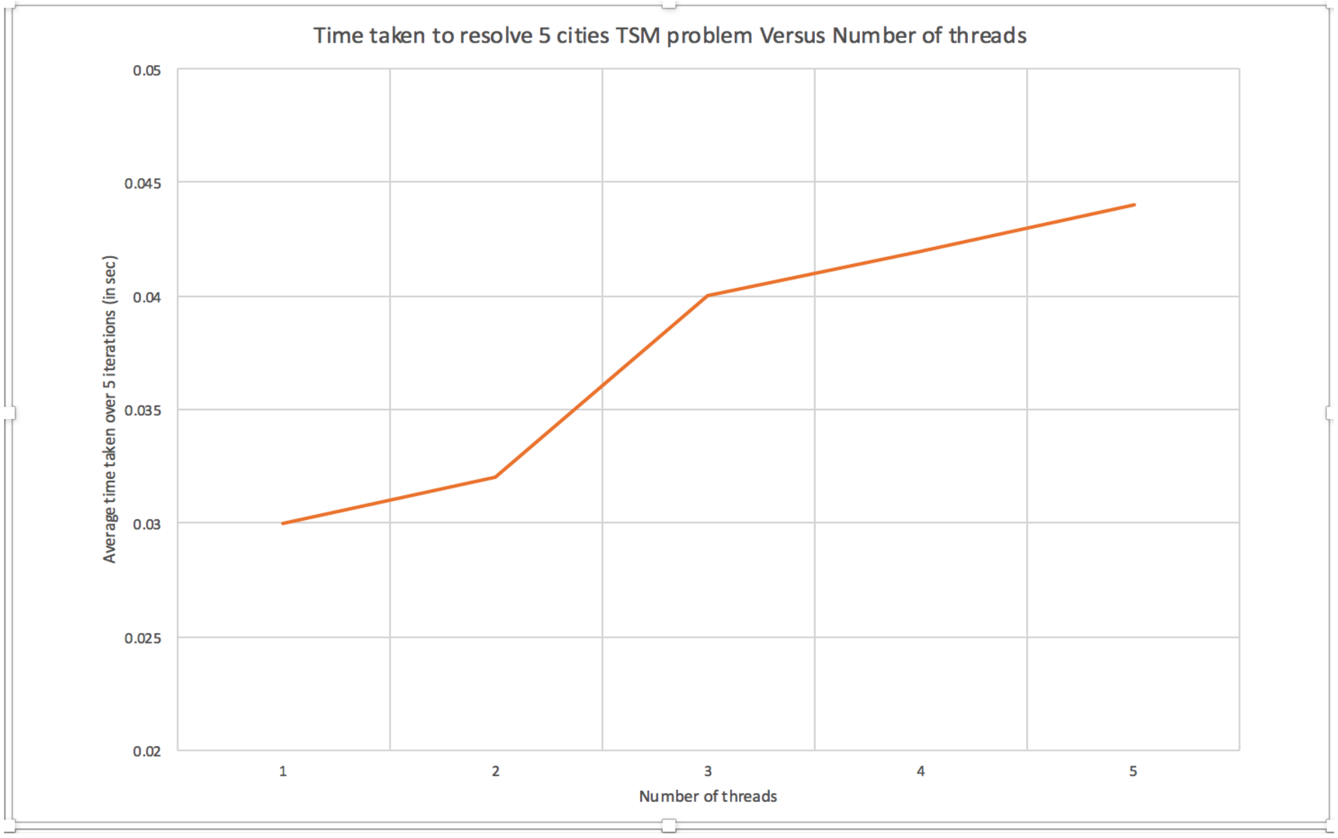

Figure 1: Time taken to resolve 5 cities TSM problem Versus Number of threads

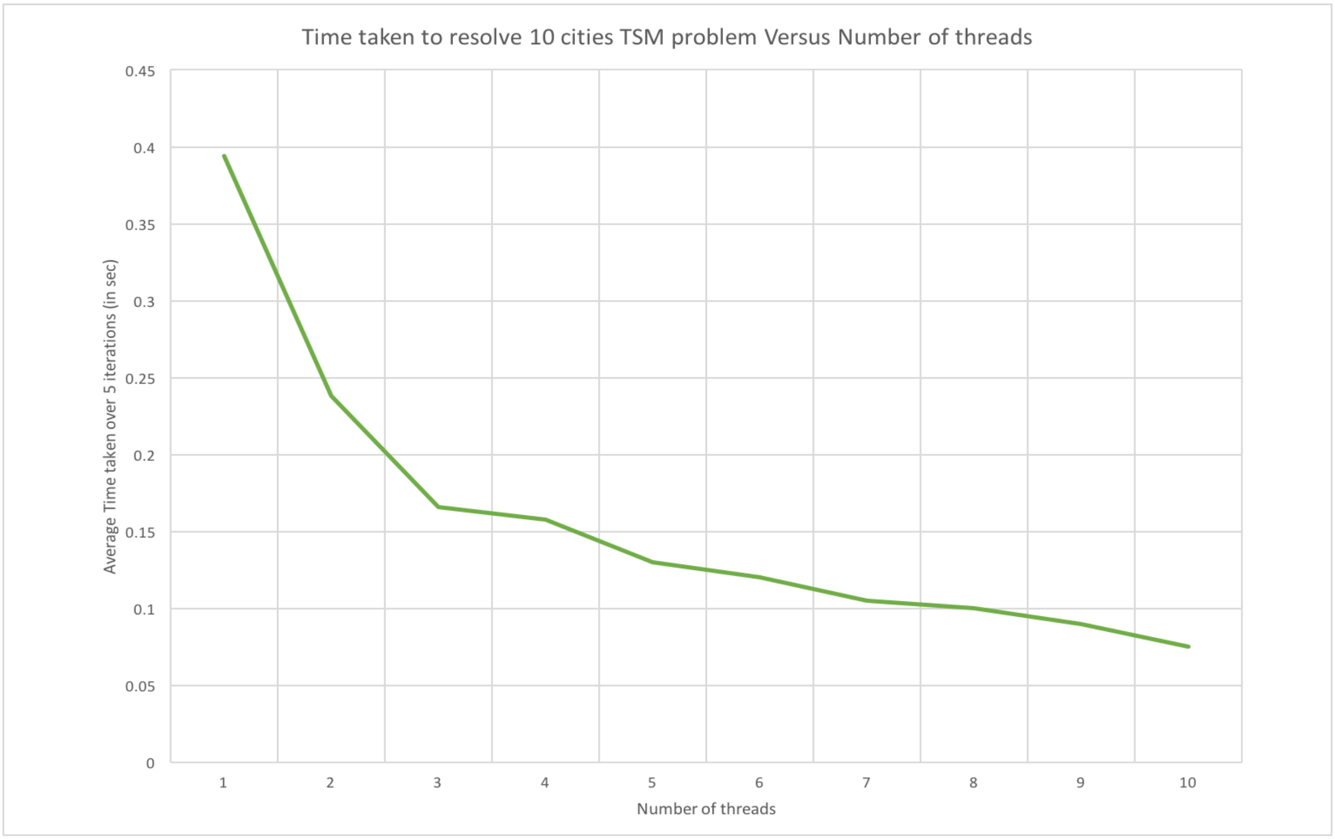

Figure 2: Time taken to resolve 10 cities TSM problem Versus Number of threads

4 Machine Specification

Above shown results are achieved on **Crunchy1** server and following is machine configuration retrived by running cat proc cpuinfo.

Siblings: 16
Model name: AMD Opteron(TM) Processor 6272
Number processors: 64
Cpu cores: 8
Bogomips: 4199.44
TLB size: 1536 4K pages
Clflush size: 64
Cache alignment: 64
Address sizes: 48 bits physical, 48 bits virtual
Power management: ts ttp tm 100mhzsteps hwpstate cpb

There are 64 logical processors present in the system. So, 64 threads can execute in parallel. For this problem statement we would not run more than 10 threads in parallel.
Five cities
Now our parallelization does not result in any speed up for a small problem of 5 cities because execution time is very less for each thread. Number of states is only 4! = 24. Now parallelizing this tiny number of states does no good and overhead of creating many threads is not favourable and thus we slowdown instead of speedup is observed.

Ten cities
But while solving the TSP problem for 10 cities in parallel over ten threads resulted in significant, approx 7 times speed up over sequential version. This is because of now each thread creation overhead is shadowed by instructions executed by a thread. But one may think that speed up should be 10 times as we are running on 10 threads in parallel but it is impossible to achieve because of threads creation overhead, present of critical section and barrier section. Results shown are in both graphs can be described logically and matches with machine specification.

5 Miscellaneous

I evaluated my program for number of cities from 3 to 20 against tsmoptimal and it passes all the test cases outputting the same minimum distance but different path because of parallel excution.

Programming code is structured as following
Single thread read input file and does all the required work before evaluating any possible path.
Multiple threads in parallel execute all the states and update the best path and minimum distance using **critical section** API.
barrier API is used until all the threads finish their work.