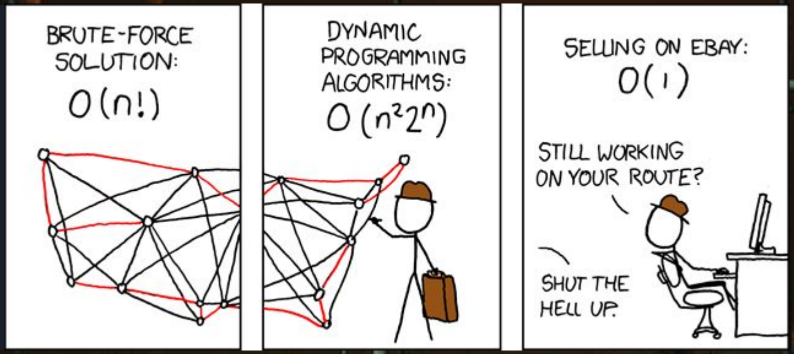
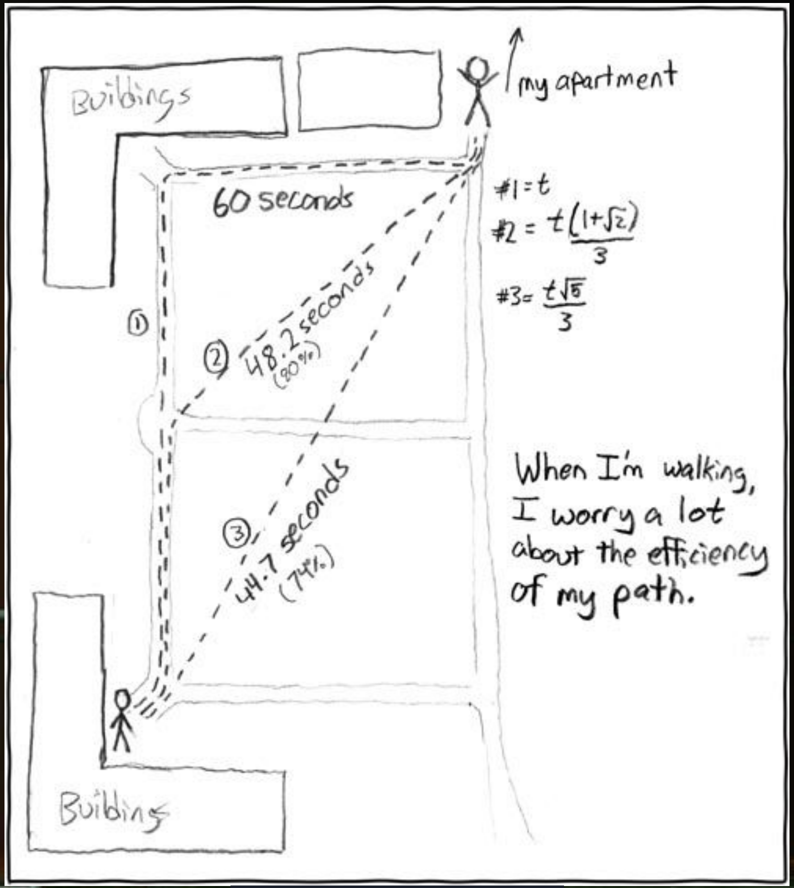
https://xkcd.com/1445/

https://xkcd.com/399/





https://xkcd.com/85/

Path Planning using Parallel Computing

High Performance Scientific Computing (ME 766)
Prof Shivasubramanian Gopalakrishnan
Spring 2021

Aaron John Sabu
Athul C D
Ayan Sharma
Vaibhav Malviya

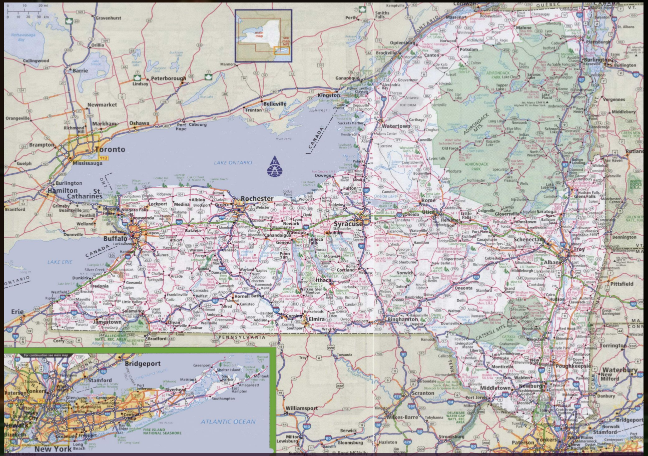
Our Goals

01 Implement path planning techniques for a roadmap-based problem from one point (node) to another using algorithms such as Dijkstra’s algorithm, Bellman-Ford algorithm, and Floyd-Warshall algorithm

02 Incorporate shared memory parallel computing elements into the program using the implementation of OpenMP at suitable locations

03 Compare the performance of the algorithms with each other along with their respective implementations on OpenMP and CUDA

New York State Roadmap...



264346 nodes
733846 edges

`a <start> <end> <distance>`

The Serial Algorithms

Dijkstra’s Algorithm

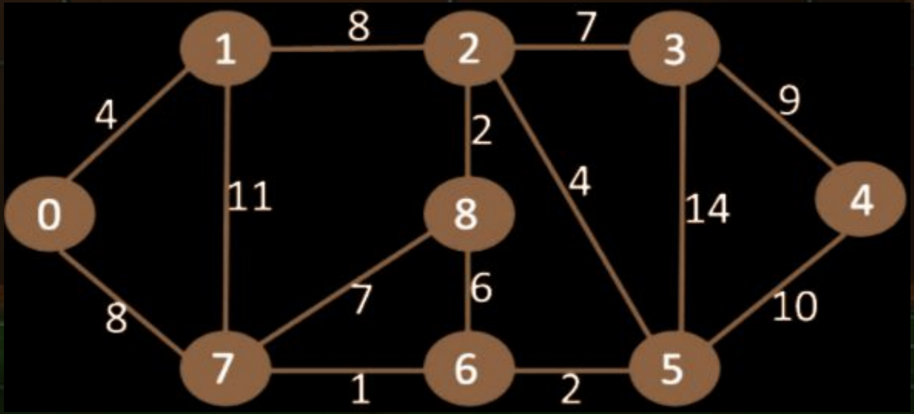
- Greedy Algorithm
- Select node with min distance and relax the connected edges

Bellman-Ford Algorithm

- Relaxes all the edges V-1 times
- Check for negative weight cycles in the end

Floyd-Warshall Algorithm

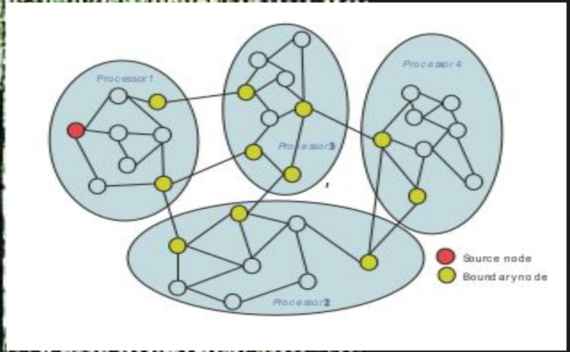
- Dynamic programming
- Shortest path in a directed weighted graph with positive or negative edge weights

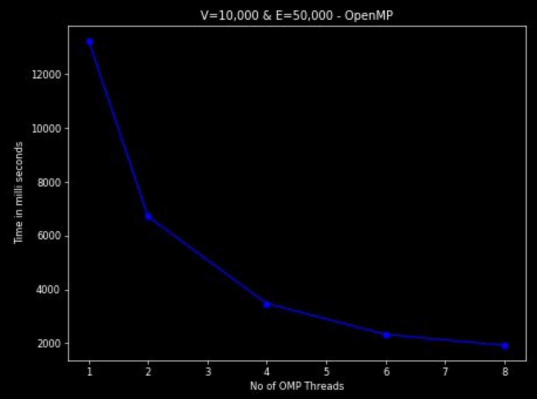


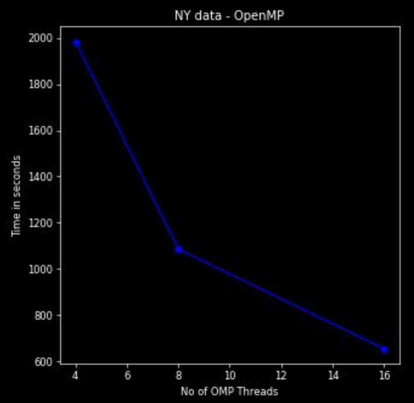
Algorithm 01

Dijkstra’s Algorithm

- A very efficient serial implementation (using C++’s STL) of the algorithm just takes 3s to run on entire NY dataset.
- We developed a parallelizable implementation of the algorithm by distributing vertices to different threads of OpenMP.
- Speed up: 1.97, 3.81, 5.68, 6.89 for 2, 4, 6, 8 threads
- Parallel algorithm on NY dataset with 4, 8, and 16 threads took 1984, 1086, and 653.8sec respectively.
- Complexity: O((V+E)logV)



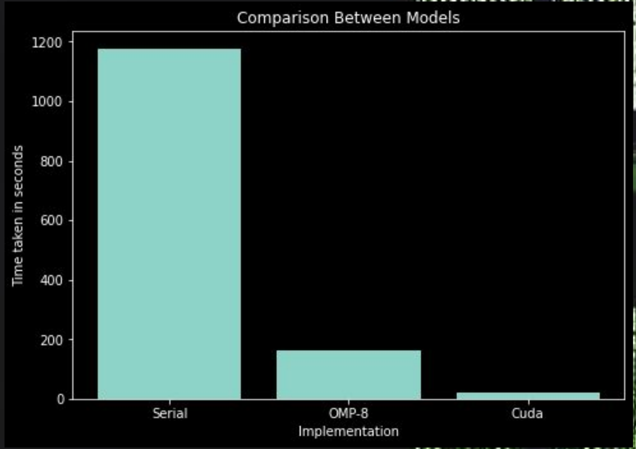




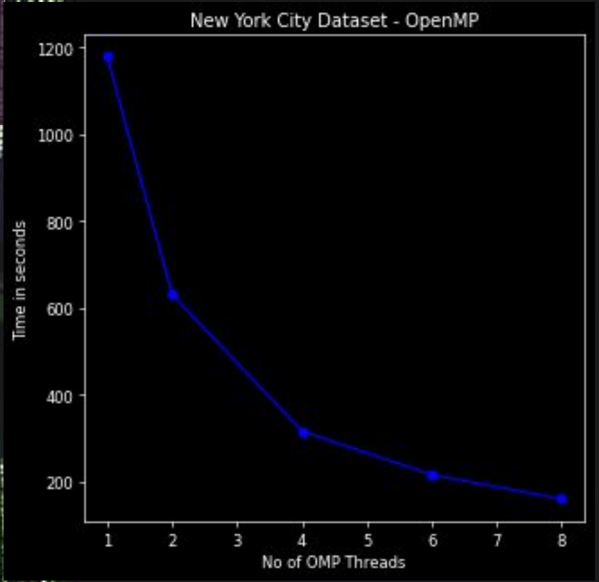
Algorithm 02

Bellman-Ford Algorithm

- Slower than Dijkstra's algo, but faster than Floyd-Warshall algo **O(VE)**
- Can detect and report negative cycles
- Not greedy: Processes all edges
- Not highly parallelizable: key loop is Sequential
- Speed up: achievable by non-sequential operations on edges
  - Running on a GPU/CUDA → significant speed up



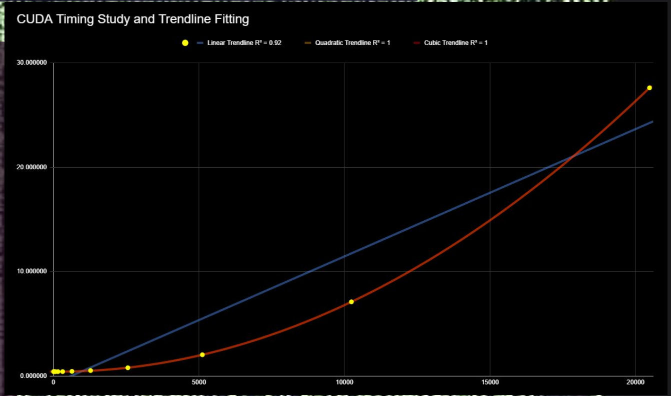
- OpenMP also gives speed up by dividing the edges across different cores
- Speed up: 1.86, 3.72, 5.44, 7.33 for 2, 4, 6, 8 threads



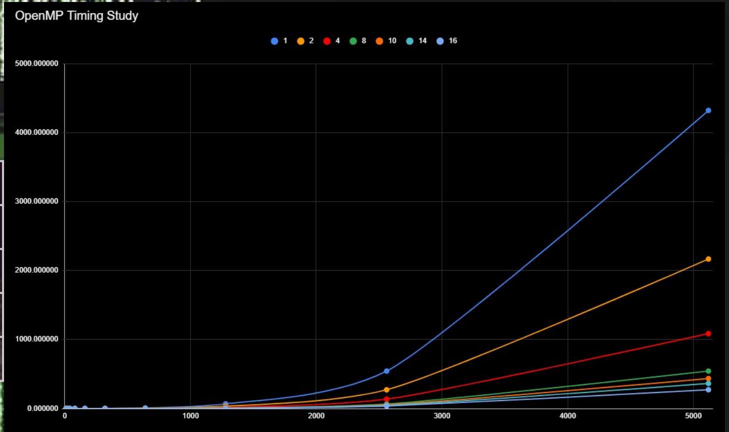
Algorithm 03

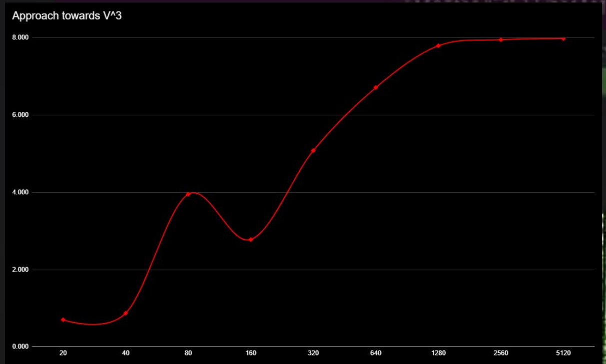
Floyd-Warshall Algorithm

- Single execution → Shortest path lengths between all vertex pairs
- Worst time complexity **$O(V^3)$**
- Usefulness:
  - Small dense graphs
  - All-pairs minimum distance



R^2 for CUDA		
k	R^2	Adj. R^2
Linear	0.918447	0.908253
Quadratic	0.999986	0.999982
Cubic	0.999988	0.999983





R^2 for OpenMP (numThreads = 8)		
k	R^2	Adj. R^2
Linear	0.847018	0.827895
Quadratic	0.997321	0.996555
Cubic	0.999999	0.999998

A Comparison of the Algorithms

	Dijkstra's	Bellman-Ford	Floyd-Warshall
Time Complexity	O((V+E)log(V))	O(VE)	$O(V^3)$
Can Negative Cycles ?	No	Yes	Yes
Suitable for :	Large/Medium	Medium/Small	Small
For NY Dataset:	Running time: 3sec	Running time: 1178s	Running time: ∞
Speed Up (8 threads):	6.89	7.33	7.965

Future Prospects

- ❖ More algorithms : A* RRT
  D* Delta stepping
- ❖ More Versatile Platforms : OpenCL
  MPI
- ❖ Multi-Agent Path Planning
  + Road Constraints
- ❖ Parallelization + Proper Hardware
  = Real-time Path Generation

References

- ➢ Tang et al., A Parallel Shortest Path Algorithm Based on Graph-Partitioning and Iterative Correcting, IEEE HPSC
- ➢ Crauser et al., A Parallelization of Dijkstra's Shortest Path Algorithm
- ➢ Li et al., Multi-Agent Path Finding for Large Agents
- ➢ Hong et al., Efficient Parallel Graph Exploration on Multi-Core CPU and GPU
- ➢ Agarwal et al., New Approach of Bellman Ford Algorithm on GPU using Compute Unified Design Architecture (CUDA)
- ➢ https://people.sc.fsu.edu/~jburkardt/c_src/dijkstra_openmp/dijkstra_openmp.html

Who did what?

AARON

Bellman-Ford Algorithm
- Serial, OpenMP, CUDA

ATHUL

Data Analysis

Test Cases

Floyd-Warshall Algorithm
- Serial, OpenMP, CUDA

Presentation

Report

Code Base

VAIBHAV

Dijkstra’s Algorithm
- Serial, OpenMP

AYAN


Thank
You