Graph Algorithms on GPUs using CUDA

Shruti Padamata, Niveditha Raveendran

Georgia Institute of Technology, Atlanta

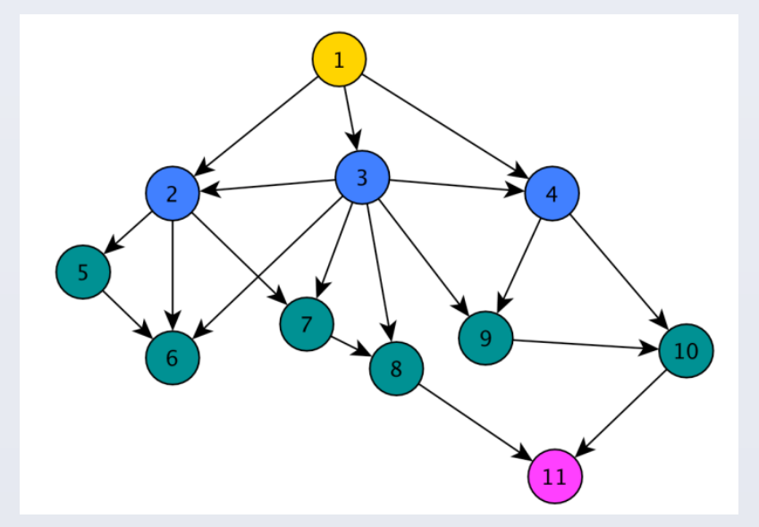
INTRODUCTION

Large graphs involving millions of vertices are common in many practical applications and are challenging to process. Graphics Processing Units (GPUs) of today have high computation power and recent work has demonstrated the plausibility of GPU graph traversal. Through this project we attempt to port two fundamental graph algorithms - breadth first search and all-pairs shortest path to GPUs, using CUDA.

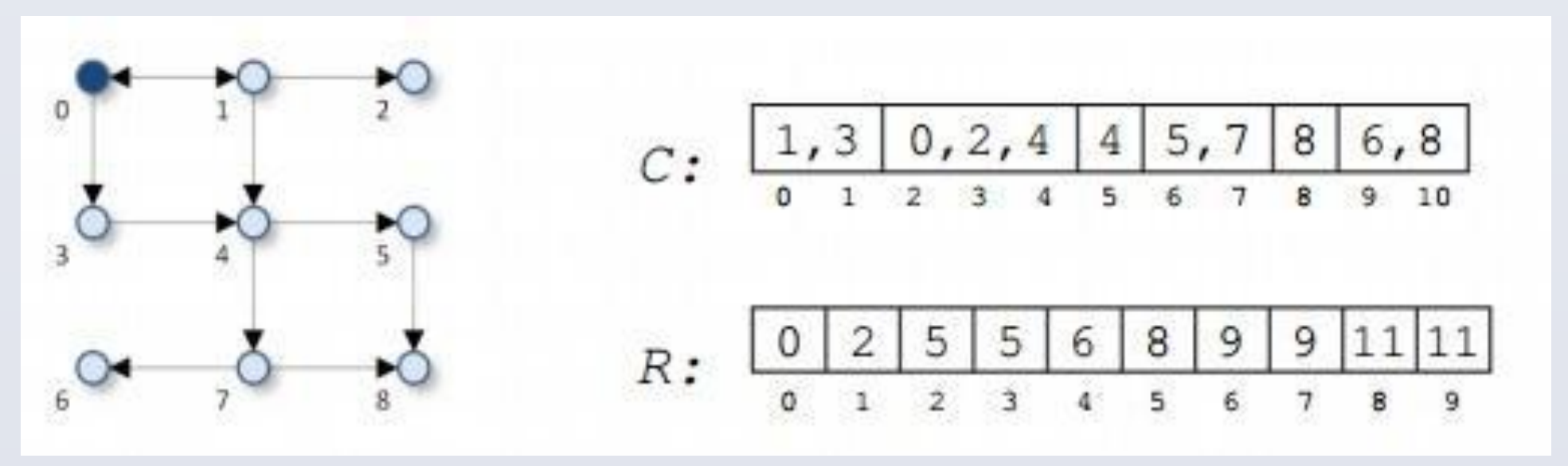
BREADTH FIRST SEARCH

Given a directed, un-weighted graph G(V,E) and a source vertex S, the breadth-first search(BFS) mechanism is used to search for a destination vertex V in G, by progressively exploring the neighboring nodes level-wise, starting from the source vertex S, as shown in the figure below.

The optimal sequential solution for this problem takes O(V+E) time.

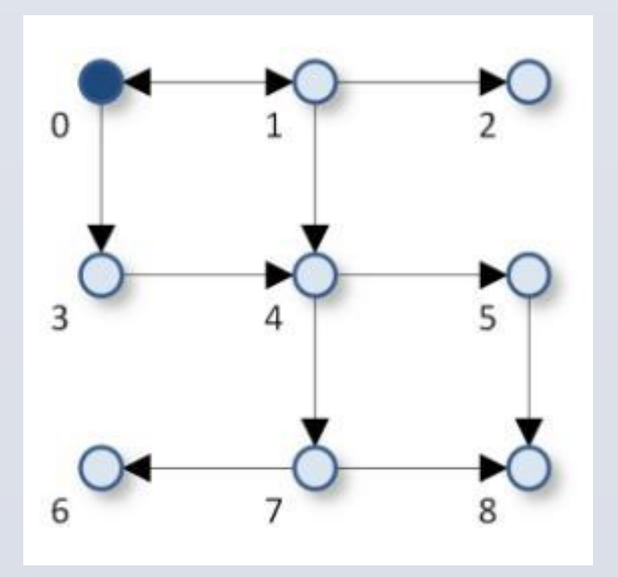


Compressed Sparse Row Representation


C: 1,3 | 0,2,4 | 4 | 5,7 | 8 | 6,8
0 1 2 3 4 5 6 7 8 9 10
R: 0 | 2 | 5 | 5 | 6 | 8 | 9 | 9 | 11 | 11
0 1 2 3 4 5 6 7 8 9

OPTIMIZED LINEAR WORK ALGORITHM

Example :



Iteration	Vertex Frontier	Next Frontier
1	{0}	{1, 3}
2	{1, 3}	{0, 2, 4, 4}
3	{2, 4}	{5, 7}
4	{5, 7}	{6, 8, 8}
5	{6, 8}	{}

IMPLEMENTATION

Initial Conditions

•Depth of source vertex set as 0

•Depth of all other vertices set to -1

•Initial Frontier contains only source node '0'.

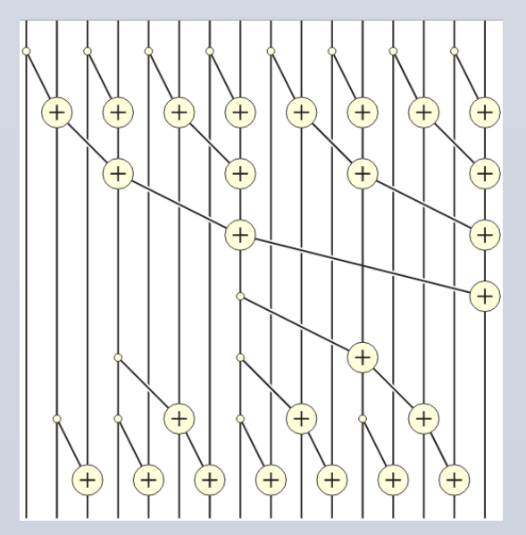
Frontier Expansion

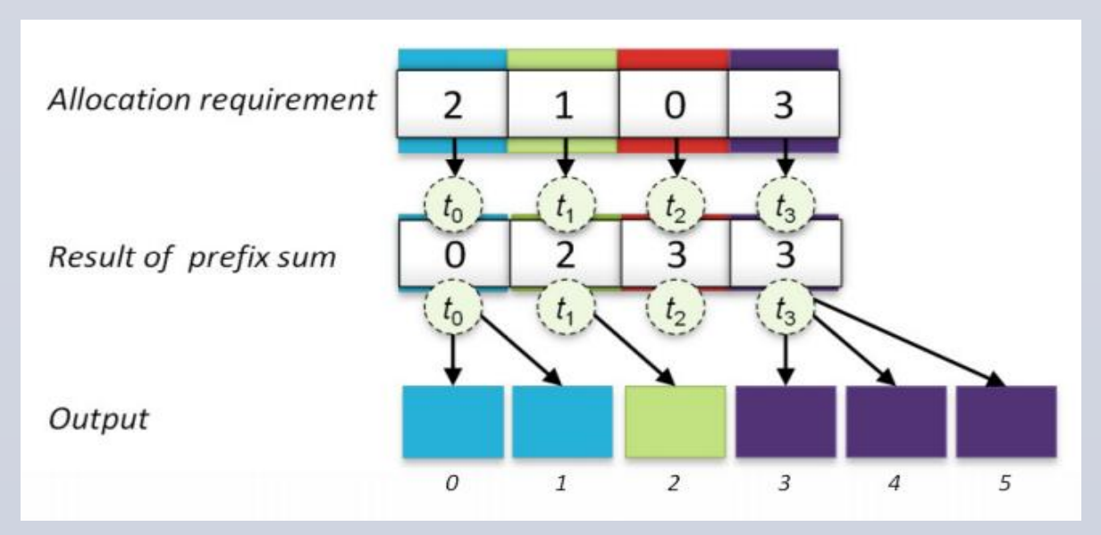
•Find the size of next frontier

•Allocate an array to hold vertices of next frontier

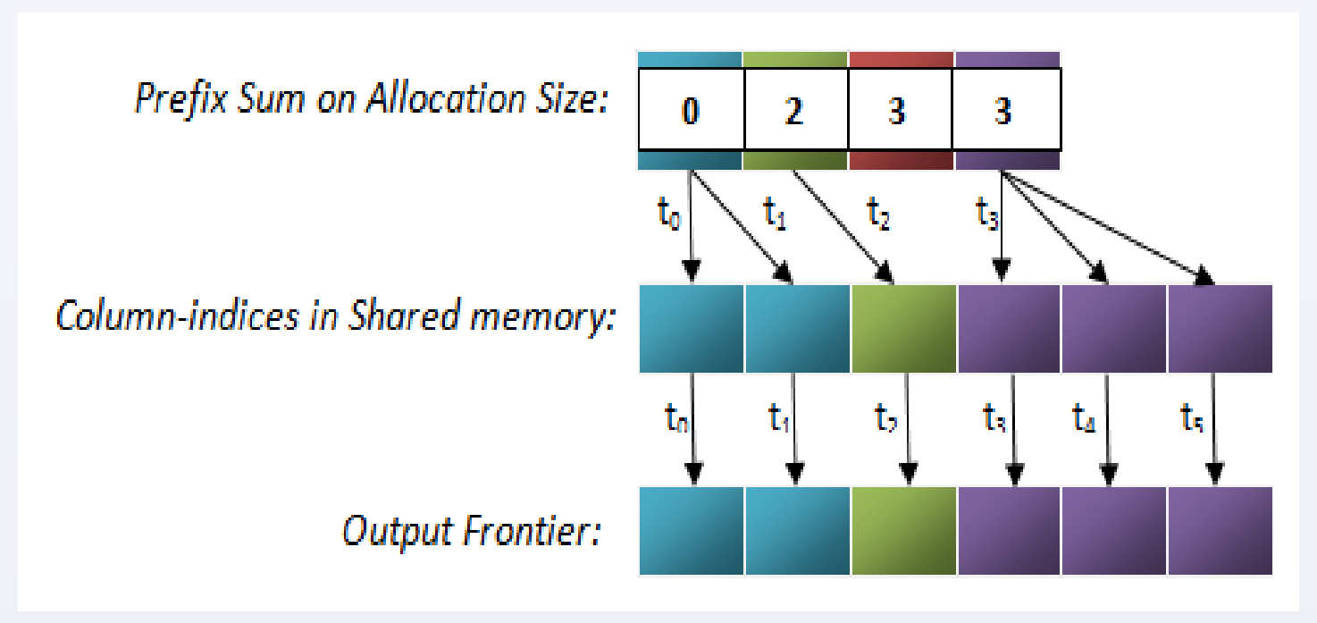
•Populate it with neighbors of each vertex in the current frontier

•Use prefix sum for cooperative allocation, i.e., to compute the offsets for where each thread should start writing its output elements


Working of Prefix Sum



Working of Frontier Expansion using Serial Gathering


Prefix Sum on Allocation Size:
0 2 3 3
Column-indices in Shared memory:
Output Frontier:
Working of Frontier Expansion using Fine Grained Gathering

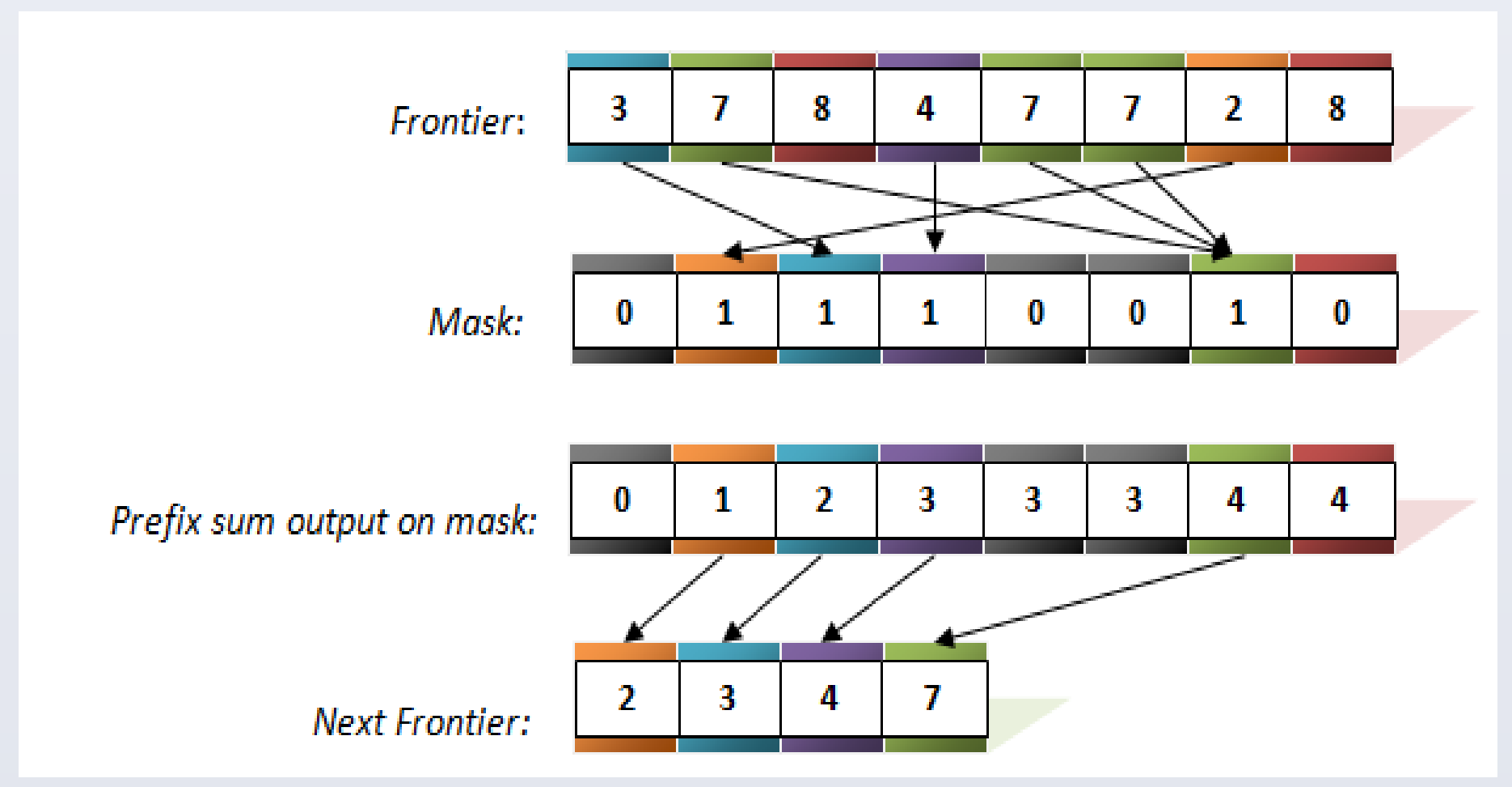
• Initial implementation used serial gathering : Unbalanced workload and divergence among threads

• Second implementation used fine grained gathering : Balanced workload among threads

Frontier Filtration

• Removal of vertices that have already been visited (those that already have their depth set)

•Removal of duplicate copies of vertices

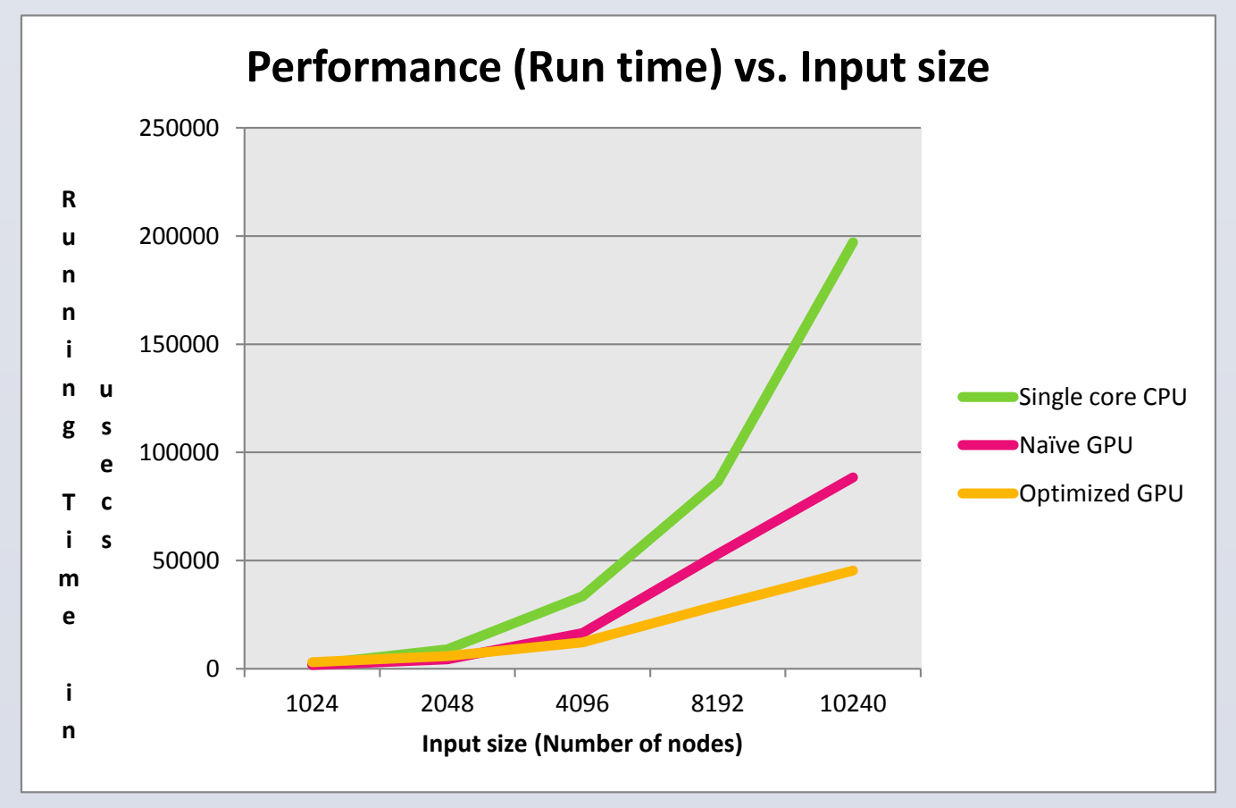


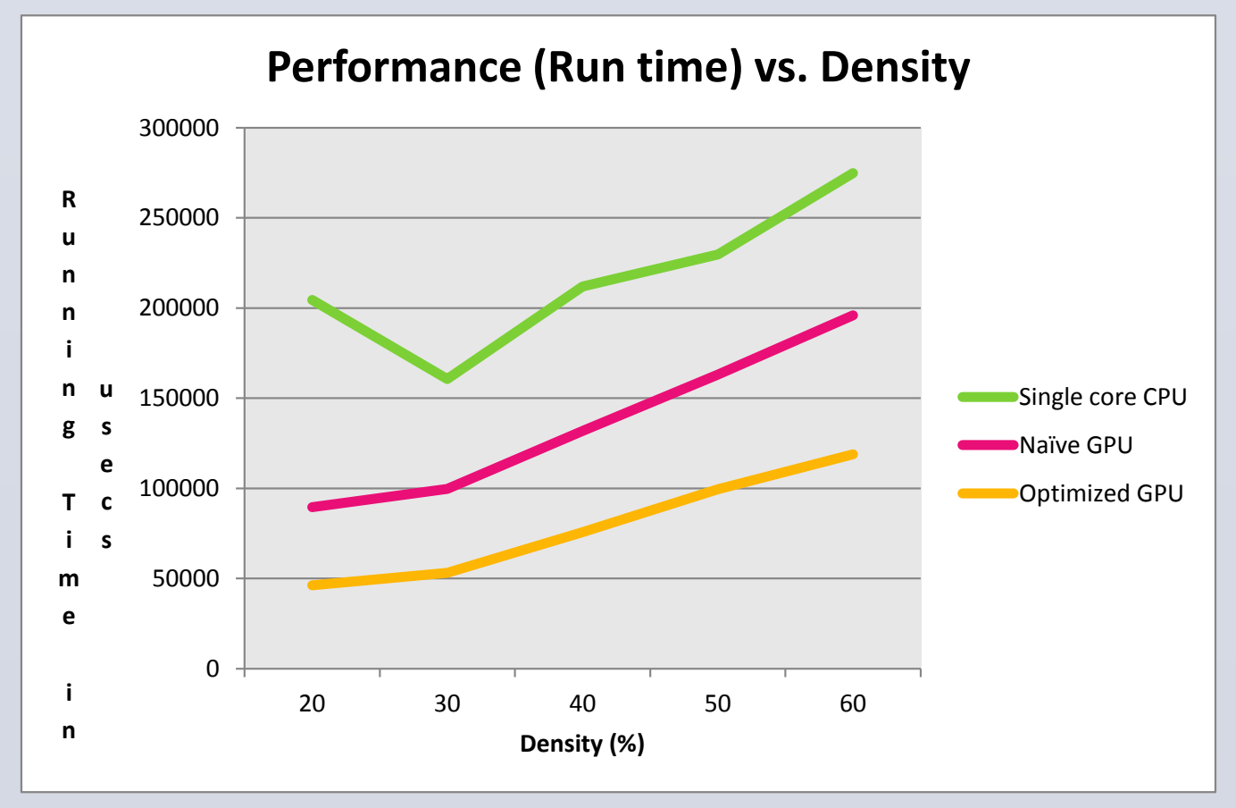
Removal of visited vertices and duplicate copies

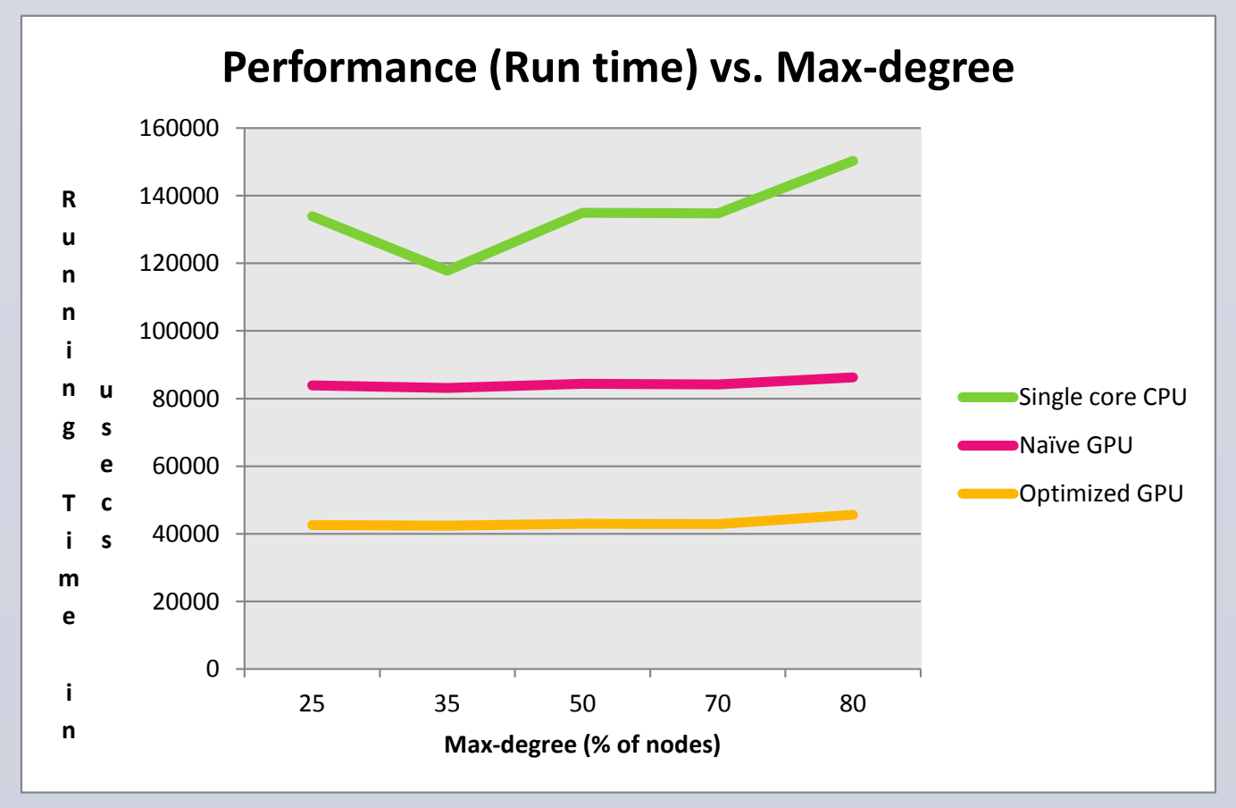
Depth Setting

• Set the current depth to all vertices in the frontier to current_depth using the mask array

EXPERIMENTS AND RESULTS







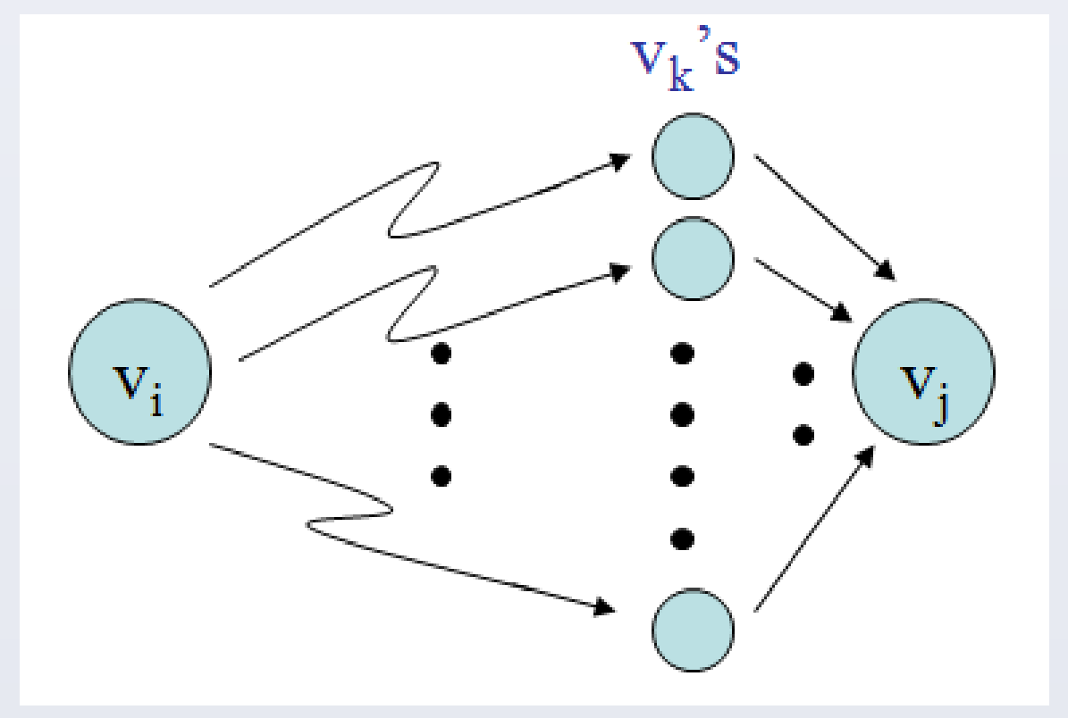
ALL PAIRS SHORTEST PATH

Given a weighted, directed graph G = (V, E) with a weight function w: E→R that maps edges to real-valued weights, the all pairs shortest path problem aims to find a shortest (least-weight) path from u to v for every pair of vertices u, v ∈ V, where the weight of a path is the sum of the weights of its constituent edges.

SHORTEST PATHS & MATRIX MULTIPLICATION

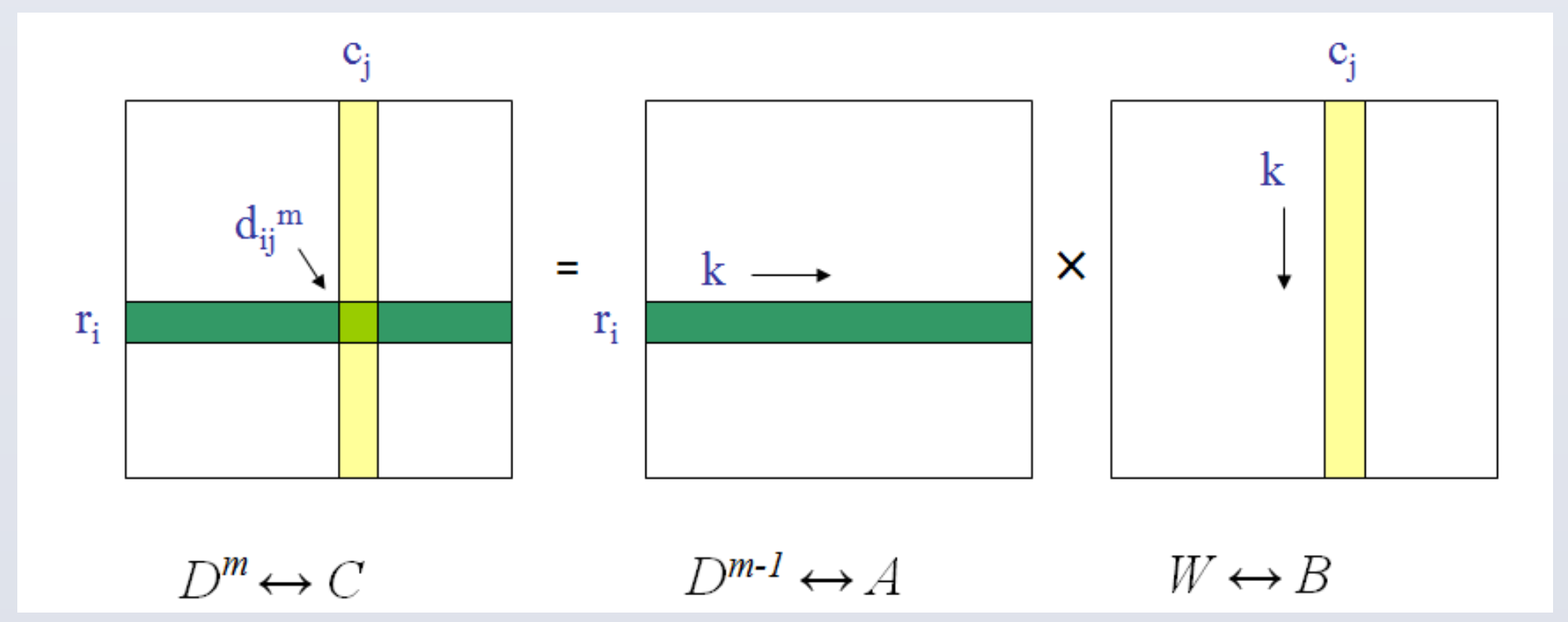
d_{ij}^{m} = minimum weight of any path from v_i to v_j that contains at most "*m*" edges.

To consider all possible shortest paths with ≤ m edges from v_i to v_j, consider shortest path with ≤ m -1 edges, from v_i to v_k, where $v_k \in R_{V_i}$ and $(v_k, v_j) \in E$



Given $W = D^1$, compute a series of matrices D^2, D^4 ….. D^n

How matrix multiplication is related to APSP?



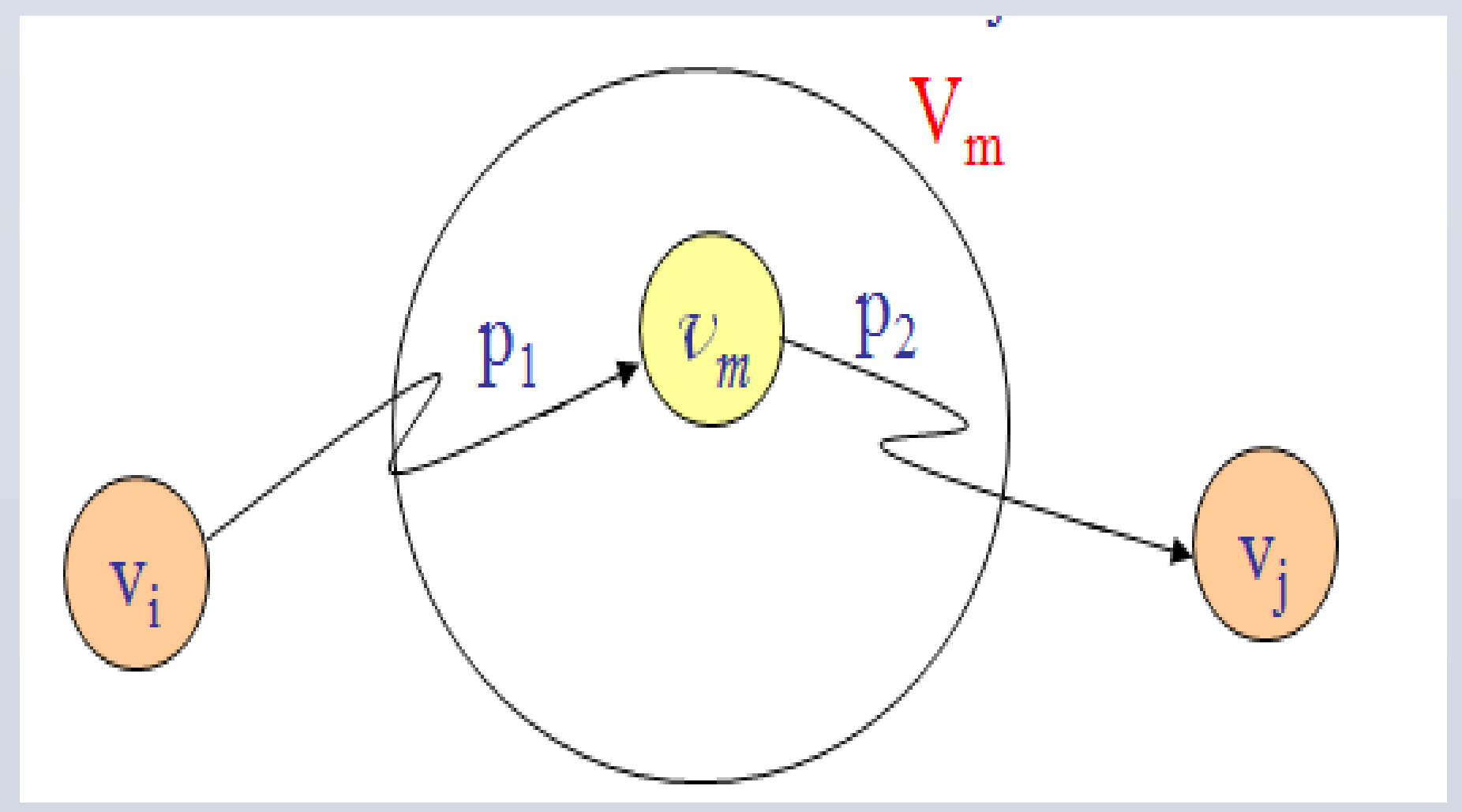
Replace + with min() and * with + in matrix multiplication.

Implemented by calling matrix multiplication – like function log(n-1) times.

Running time is $\Theta(n^3 \lg n)$

FLOYD WARSHALLS ALGORITHM

p_{ij}^{m} : weight of a shortest path from v_i to v_j with all intermediate vertices from $V_m = \{ v_1, v_2, \ldots, v_m \}$



```
FLOYD-WARSHALL(W)
1 n ← rows[W]
2 $D_{(0)}$ ← W
3 for k ← 1 to n
4   do for i ← 1 to n
5      do for j ← 1 to n
6         do $d_{ij}^{k} \leftarrow \min\{ d_{ij}^{k-1}, d_{ik}^{k-1} + d_{jk}^{k-1} \}$
7 return D(n)
```

BLOCKED FLOYD WARSHALLS ALGORITHM

• Divide the n x n matrix W into $(n/B)^2$ sub matrices of size B x B where B is the blocking factor

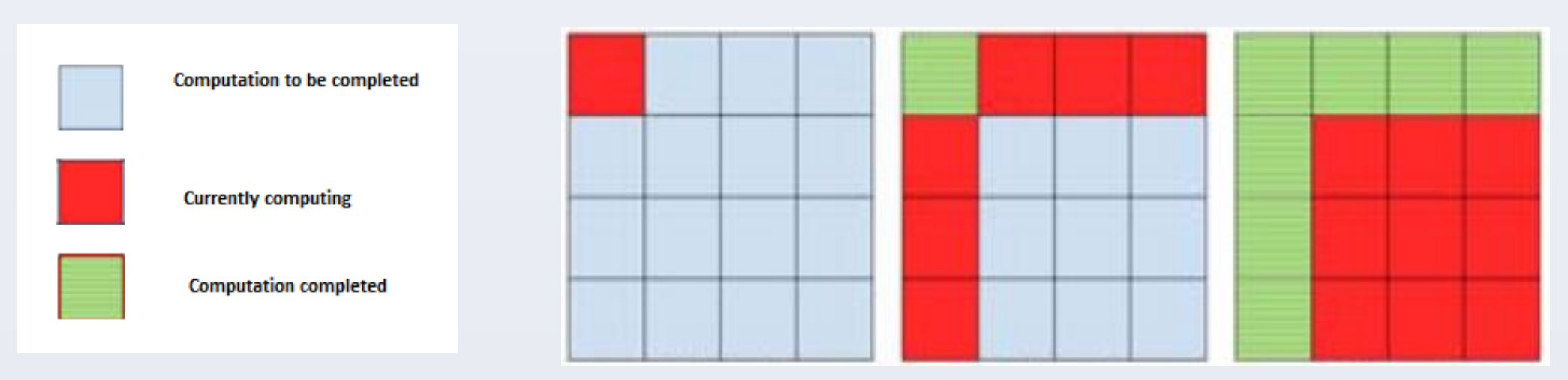
• In phase 1, compute results of FW for B iterations for the ith diagonal block

• In phase 2, compute results of FW for B iterations for the blocks in the ith

•row and ith column

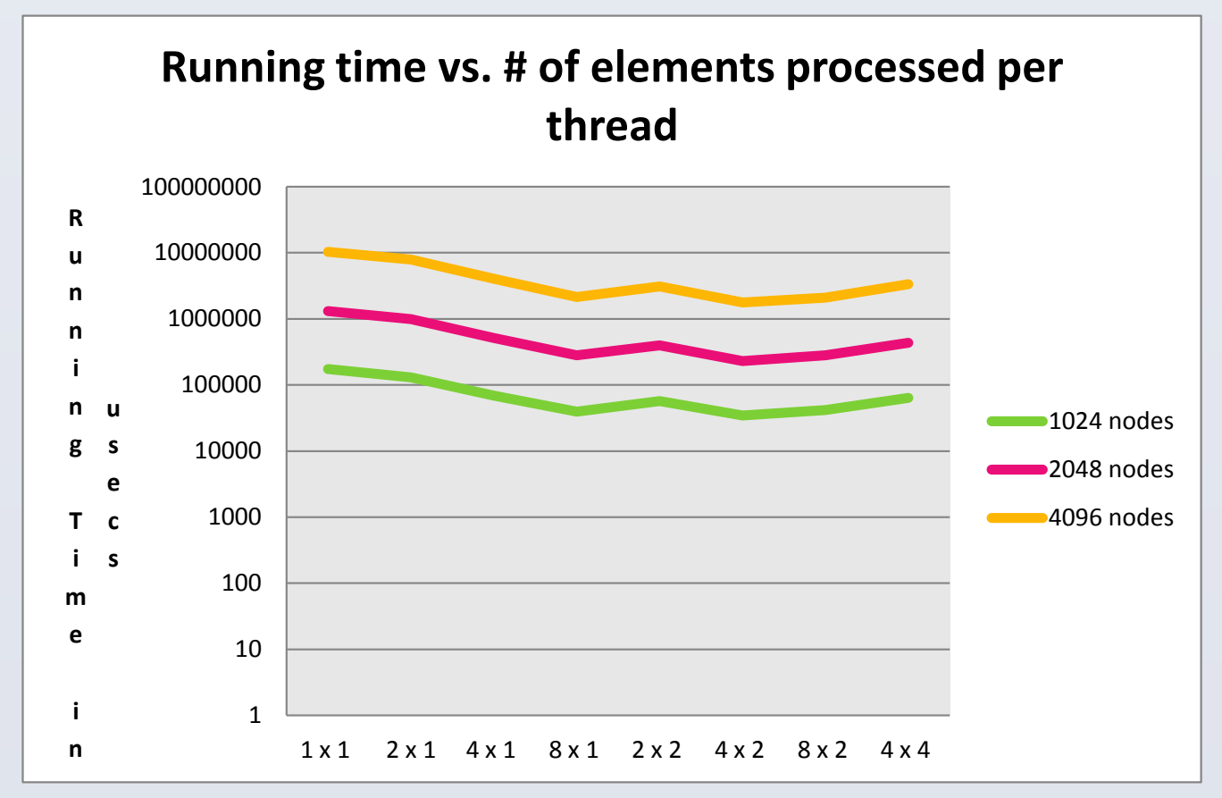
• In phase 3, compute results of FW for B iterations for the remaining blocks

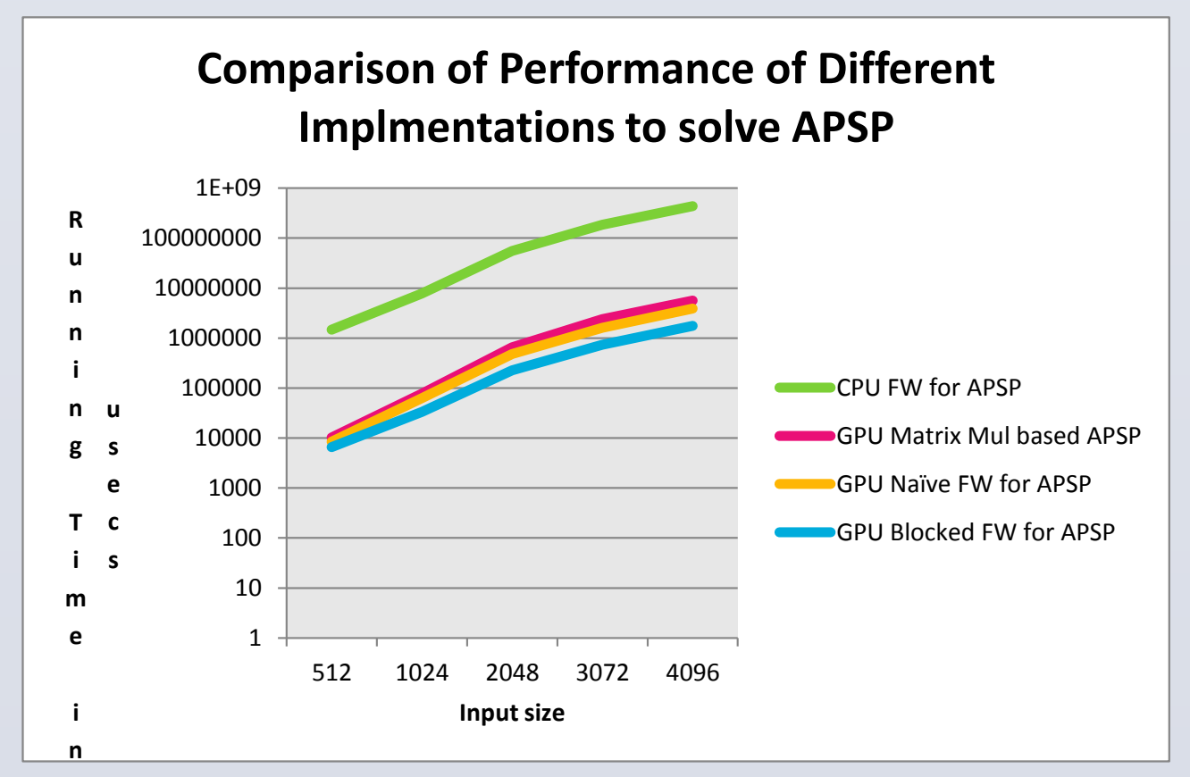
• Repeat the three phases n/B times with I varying from 0 to (n/B) - 1



Phases when block (1,1) is the self dependent block

EXPERIMENTS & RESULTS





CONCLUSION

The results from the experiments demonstrate that GPUs are well-suited for both sparse graph traversal as well as Floyd Warshall's Algorithm. Future work could involve porting these graph algorithms for computation across multiple GPUs.

REFERENCES

[1] "Scalable GPU Graph Traversal" Duane Merrill (NVIDIA), Michael Garland (NVIDIA), Andrew Grimshaw (University of Virginia), in 17th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'12), Feb 2012.

[2] Gary J. Katz and Joseph T. Kider, Jr. 2008. All-pairs shortest-paths for large graphs on the GPU. In *Proceedings of the 23rd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware*(GH '08). Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 47-55.