CPU-Based Shortest Path Algorithms

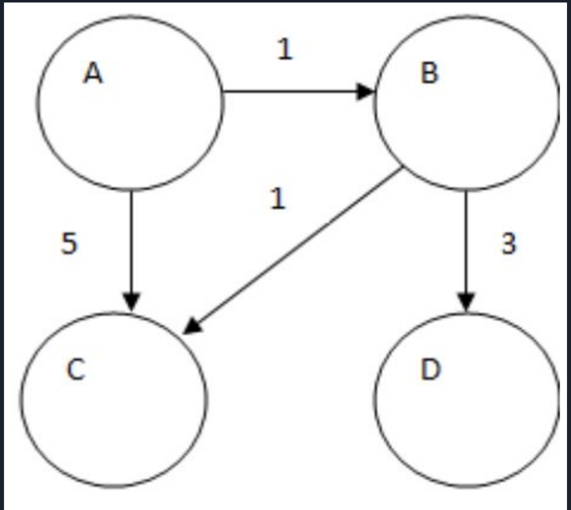
Brendan Ngo, Aravind Srinivasan

Definition: Shortest Path

Given a digraph *G* with non-negative weights on its edges, the shortest path from a source node *S*, and a destination node *T* is a directed path from *S* to *T* with the minimum total weight.

- Single-Source Shortest Path: *S* to V_i , $\forall\ V_i \in V$
- All-Pairs Shortest Path: V_i to V_j, $\forall\ V_i, V_j \in V$

Example



	A	B	C	D
A	0	1	2	4
B	∞	0	1	3
C	∞	∞	0	∞
D	∞	∞	∞	0

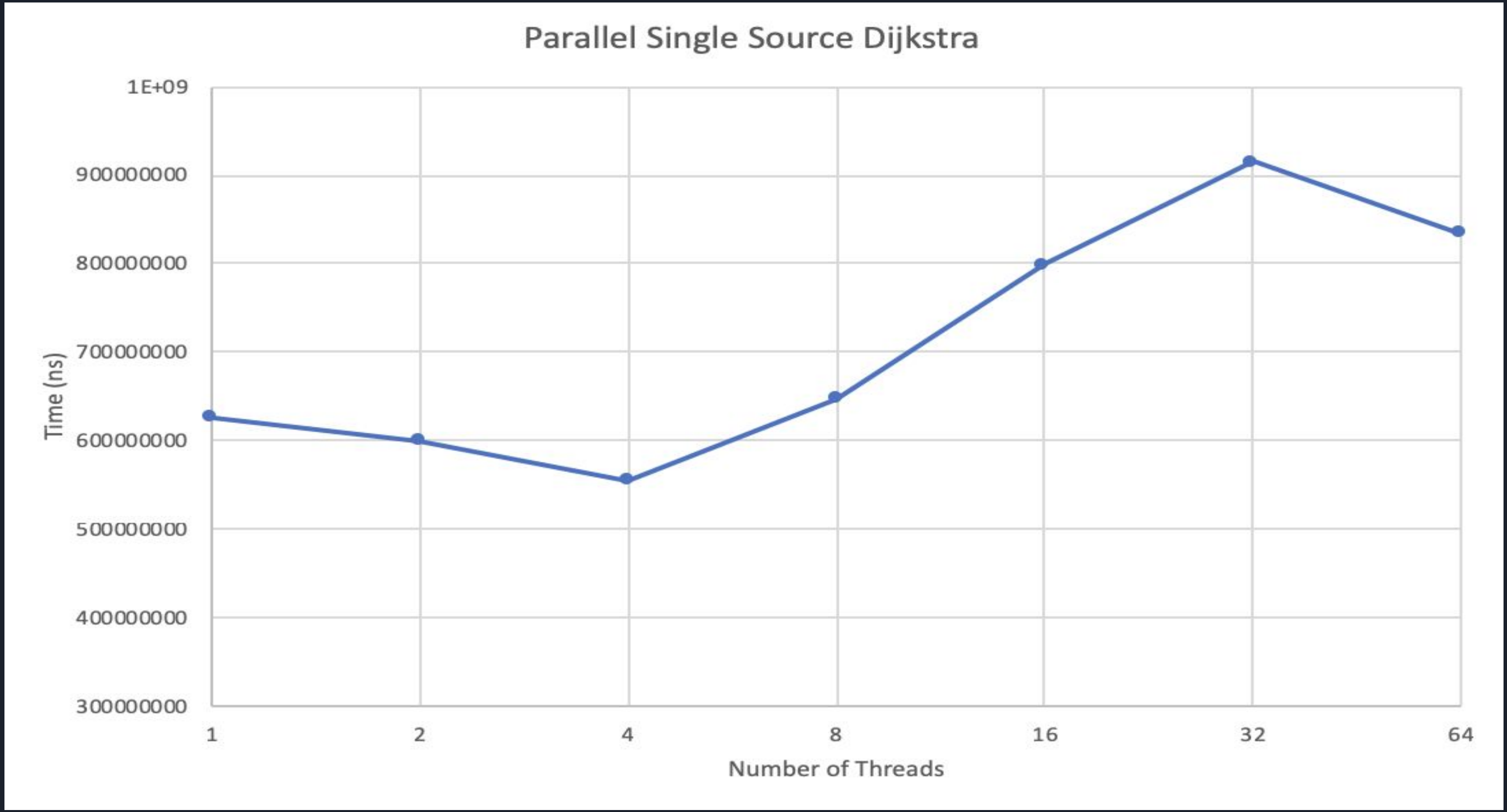
SSSP: Single-Source Shortest Path

1. Mark all nodes as unvisited and set their weights to ∞
2. Set current node weight to 0.
3. Remove node with smallest weight from unvisited set
4. Relax its outgoing edges - Given edge (u, v), dist(v) = min(dist(u) + weight(e), dist(v))
5. Repeat steps 3 and 4 until the unvisited set is empty.

Dijkstra’s Runtime: $O(E + V \lg V)$

Parallel Dijkstra’s finds the minimum in parallel with P threads.
Runtime: $O(E + V^2/P) \rightarrow O(E + V)$

SSSP: Single-Source Shortest Path



APSP: All-Pairs Shortest Path

Sequential Floyd-Warshall Algorithm $O(V^3)$:

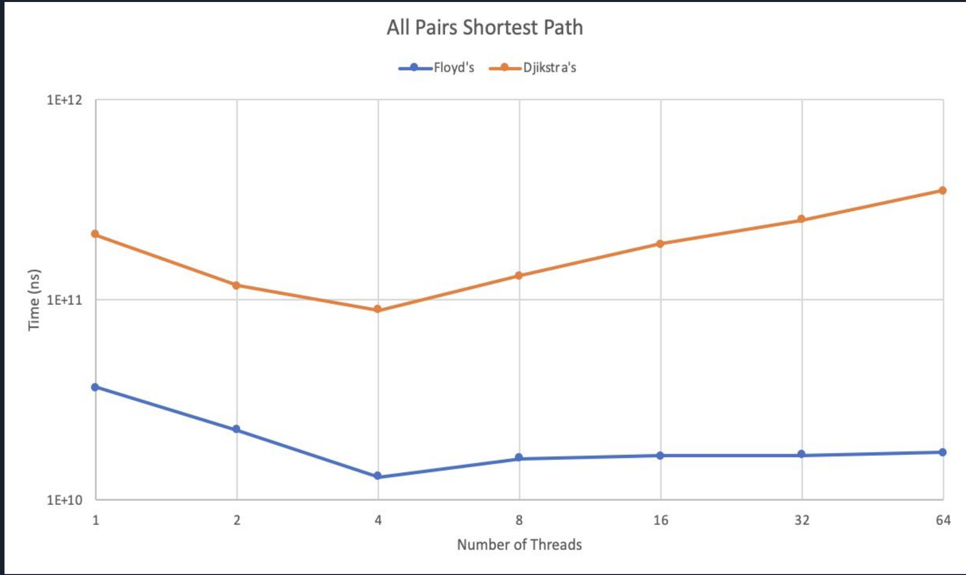
```
1 let dist be a |V| × |V| array of minimum distances initialized to ∞ (infinity)
2 for each edge (u,v)
3    dist[u][v] ← w(u,v)  // the weight of the edge (u,v)
4 for each vertex v
5    dist[v][v] ← 0
6 for k from 1 to |V|
7    for i from 1 to |V|
8       for j from 1 to |V|
9          if dist[i][j] > dist[i][k] + dist[k][j]
10             dist[i][j] ← dist[i][k] + dist[k][j]
11         end if
```

Lines 7-11 can be parallelized with P threads for runtime $O(V^3/P) \rightarrow O(V)$ with V^2 threads

APSP: All-Pairs Shortest Path

Alternative: Run Djikstra's Algorithm in parallel $\forall V_i \in V$

Runtime: $O(E + V \lg V)$



As expected, Floyd-Warshall's is faster than running Parallel Dijkstra's

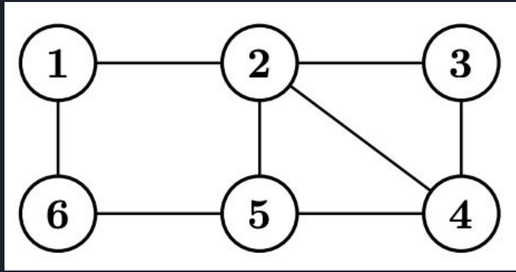
Trade off: Floyd-Warshall needs V^2 threads vs Dijkstra's needs V threads.

When we have large, sparse graphs with limited cores,

$O(V/P * (E + V\lg V)) \cong O(V^2 \lg V / P) \leq O(V^3/P)$

Matrix Multiplication Based Implementations: Distance Products

Given W, a nxn matrix containing the edge weights of the graph, W^k gives the distances realized by paths that use at most k edges.



1.

$$(\mathbf{A} \star \mathbf{B})_{ij} = \min_{k=1}^{n}\{a_{ik} + b_{kj}\}, 1 \leq i, j \leq n.$$

2.

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

$$A^2 = \begin{pmatrix} 2 & 0 & 1 & 1 & 2 & 0 \\ 0 & 4 & 1 & 2 & 1 & 2 \\ 1 & 1 & 2 & 1 & 2 & 0 \\ 1 & 2 & 1 & 3 & 1 & 1 \\ 2 & 1 & 2 & 1 & 3 & 0 \\ 0 & 2 & 0 & 1 & 0 & 2 \end{pmatrix}$$

$$B = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 \end{pmatrix}$$

Seidel's Algorithm

- APSP Algorithm for unweighted, undirected graphs
- Computes logarithmic number of distance matrices
- O(M(n)*log(n)), where M(n) is the cost of multiplying 2 matrices
  - Sequential: $O(n^{2.375477}\log(n))$
  - Parallel: $O(\log^2(n))$

Seidel(A)

Let $Z = A \cdot A$

Let B be an n x n 0-1 matrix,

$$\text{where } b_{ij} = \begin{cases} 1 & \text{if } i \neq j \text{ and } (a_{ij} = 1 \text{ or } z_{ij} > 0) \\ 0 & \text{otherwise} \end{cases}$$

IF $\forall i, j,\ i \neq j\ b_{ij} = 1$ then

Return D = 2B-A

Let T = Seidel(B)

Let $X = T \cdot A$

Return n x n matrix D,

$$\text{where } d_{ij} = \begin{cases} 2t_{ij} & \text{if } x_{ij} \geq t_{ij} \cdot degree(j) \\ 2t_{ij} - 1 & \text{if } x_{ij} < t_{ij} \cdot degree(j) \end{cases}$$

Shoshan-Zwick Algorithm

- APSP algorithm that works on weighted, undirected graphs where existing edge weights are integers in the range of {1, 2, …, N}
- Computes a logarithmic number of distance products in order to determine shortest paths
- Algorithm is based on not allowing the range of elements in the matrix it is uses to increase
- Stores distance matrices as it iterates in order to recover the shortest path
- O(log(N*n)*N*M(n))
    - Sequential: $O(N*n^{2.375477}*\log(N*n))$
    - Parallel: $O(N*\log(n)\log^2 N*n))$

Function SHOSHAN-ZWICK-APSP($\mathbf{D}$)

$l = \lceil \log_2 n \rceil$.
$m = \log_2 M$.
for $(k = 1 \text{ to } m+1)$ **do**
  $\mathbf{D} = clip(\mathbf{D} \star \mathbf{D}, 0, 2 \cdot M)$.
end for
$\mathbf{A}_0 = \mathbf{D} - M$.
for $(k = 1 \text{ to } l)$ **do**
  $\mathbf{A}_k = clip(\mathbf{A}_{k-1} \star \mathbf{A}_{k-1}, -M, M)$.
end for
$\mathbf{C}_l = -M$.
$\mathbf{P}_l = clip(\mathbf{D}, 0, M)$.
$\mathbf{Q}_l = +\infty$.
for $(k = l-1 \text{ down to } 0)$ **do**
  $\mathbf{C}_k = [clip(\mathbf{P}_{k+1} \star \mathbf{A}_k, -M, M) \bigwedge \mathbf{C}_{k+1}] \bigvee [clip(\mathbf{Q}_{k+1} \star \mathbf{A}_k, -M, M) \bar{\bigwedge} \mathbf{C}_{k+1}]$.
  $\mathbf{P}_k = \mathbf{P}_{k+1} \bigvee \mathbf{Q}_{k+1}$.
  $\mathbf{Q}_k = chop(\mathbf{C}_k, 1-M, M)$.
end for
for $(k = 1 \text{ to } l)$ **do**
  $\mathbf{B}_k = (\mathbf{C}_k \geq 0)$.
end for
$\hat{\mathbf{B}}_0 = (-M < \mathbf{P}_0 < 0)$.
$\hat{\mathbf{R}} = \mathbf{P}_0$.
$\Delta = M \cdot \sum_{k=1}^{l} 2^k \cdot \mathbf{B}_k + 2 \cdot M \cdot \hat{\mathbf{B}}_0 + \hat{\mathbf{R}}$.
return Δ.

$$(clip(\mathbf{A}, a, b))_{ij} = \begin{cases} a & \text{if } a_{ij} < a \\ a_{ij} & \text{if } a \leq a_{ij} \leq b \\ +\infty & \text{if } a_{ij} > b \end{cases}$$

$$(chop(\mathbf{A}, a, b))_{ij} = \begin{cases} a_{ij} & \text{if } a \leq a_{ij} \leq b \\ +\infty & \text{otherwise} \end{cases}.$$

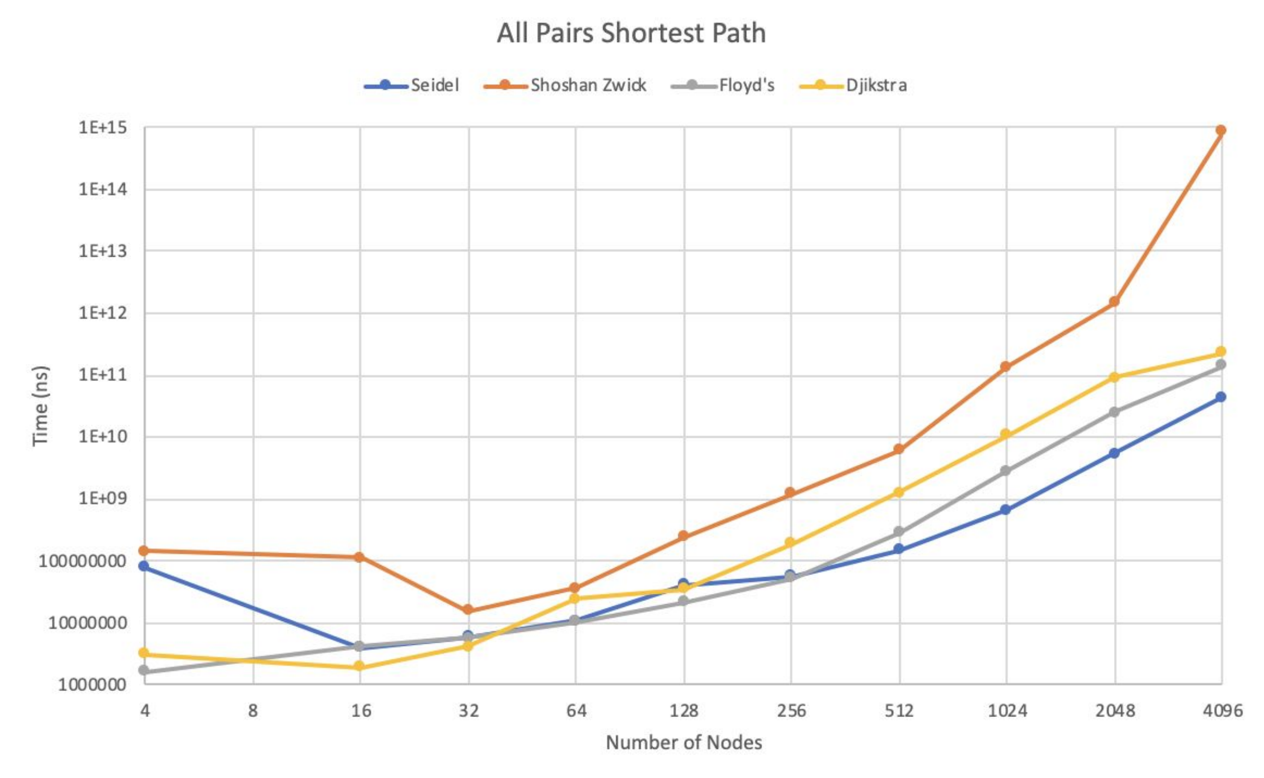
$$\left(\mathbf{A} \bigwedge \mathbf{B}\right)_{ij} = \begin{cases} a_{ij} & \text{if } b_{ij} < 0 \\ +\infty & \text{otherwise} \end{cases}$$

$$\left(\mathbf{A} \bar{\bigwedge} \mathbf{B}\right)_{ij} = \begin{cases} a_{ij} & \text{if } b_{ij} \geq 0 \\ +\infty & \text{otherwise} \end{cases}$$

$$\left(\mathbf{A} \bigvee \mathbf{B}\right)_{ij} = \begin{cases} a_{ij} & \text{if } a_{ij} \neq +\infty \\ b_{ij} & \text{if } a_{ij} = +\infty, b_{ij} \neq +\infty \\ +\infty & \text{if } a_{ij} = b_{ij} = +\infty \end{cases}$$

$$(\mathbf{C} \geq 0)_{ij} = \begin{cases} 1 & \text{if } c_{ij} \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$(0 \leq \mathbf{P} \leq M)_{ij} = \begin{cases} 1 & \text{if } 0 \leq p_{ij} \leq M \\ 0 & \text{otherwise} \end{cases}$$



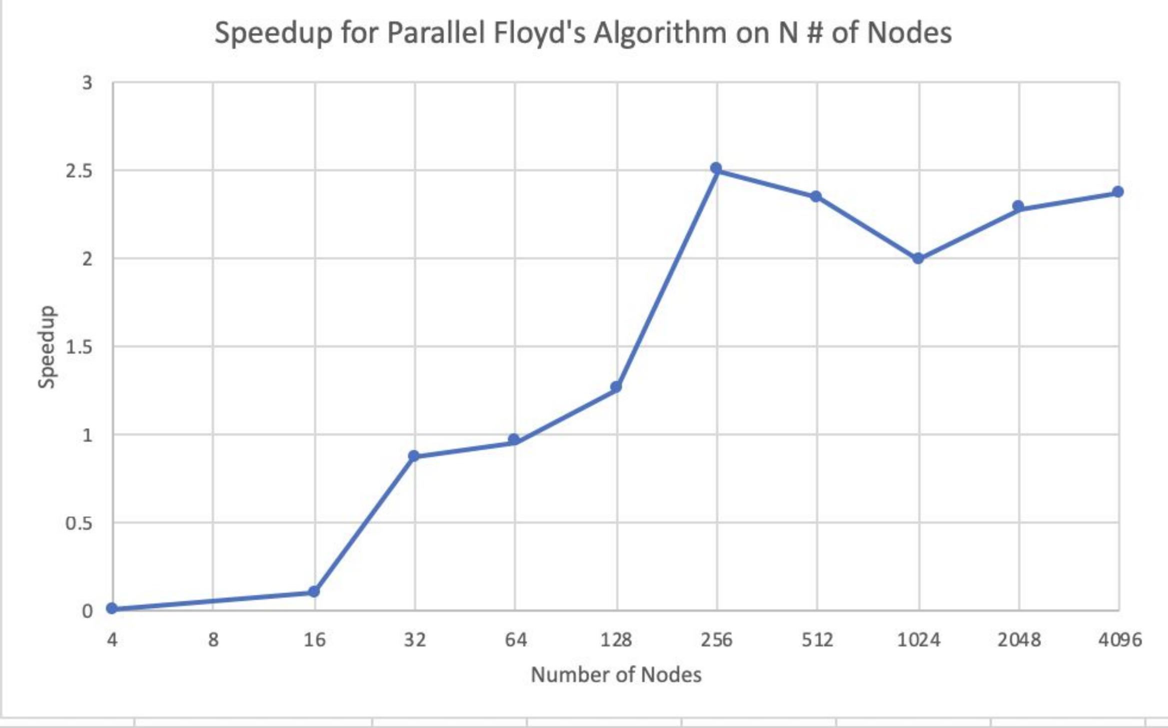
Analysis of Matrix Based Implementations

Conclusion

- Shortest Path Algorithms can be improved using parallelization on CPU
- CPU-based implementations efficiency capped by number of available processors
  - Maximum speedup when num of threads = number of available processors (4)
- Speedups are only realized for larger number of nodes
  - For smaller inputs, benefits of parallelization are negligible and cost of setting up parallelization too high
- Matrix-based algorithms do not perform well in practice on CPU

References

[1] Djikstra’s Algorithm, Dijkstra, E. W. (1959). "A note on two problems in connexion with graphs"
[2] Floyd-Warshall Algorithm, Floyd, Robert W. (June 1962). "Algorithm 97: Shortest Path". *Communications of the ACM*. 5 (6): 345.
[3] Seidel’s Algorithm, Seidel, R. (1995). "On the All-Pairs-Shortest-Path Problem in Unweighted Undirected Graphs". *Journal of Computer and System Sciences*. **51** (3): 400–403.
[4] Strassen Matrix Multiplication Algorithm, Strassen, Volker, *Gaussian Elimination is not Optimal*, Numer. Math. 13, p. 354-356, 1969
[5] Coppersmith-Winograd Matrix Multiplication Algorithm, Coppersmith, Don; Winograd, Shmuel (1990), "Matrix multiplication via arithmetic progressions" (PDF), *Journal of Symbolic Computation*, **9** (3): 251
[7] Shoshan-Zwick Algorithm, A. Shoshan and U. Zwick. All pairs shortest paths in undirected graphs with integer weights. In FOCS, pages 605–614, 1999.
[8] Correction of the Shoshan-Zwick Algorithm, Pavlos Eirinakis, Matthew Williamson, and K. Subramani. On the Shoshan-Zwick algorithm for the all-pairs shortest path problem. J. Graph Algorithms Appl., 21(2):177–181, 2017.



Parallelization is not helpful for smaller graphs