

Some Parallel Shortest Path Algorithms

P.Y. Wang
Department of Computer Science 4A5
George Mason University
Fairfax VA 22030-4444 U.S.A.

All-Pairs Shortest Paths

- Given a graph $G = (V, E)$ having vertices $v_i \in V$ and costs $w_{i,j} > 0$ for edge $(v_i, v_j) \in E$, determine the cost of the shortest paths between all pairs of vertices $v_i, v_j \in V$ such that $v_i \neq v_j$.
- Output the results in a matrix $D = d_{i,j}$ such that $d_{i,j}$ is the cost of the shortest path from v_i to v_j .
- Three proposed parallel solutions can be based on:
 - Matrix-Multiplication (dynamic programming)
 - Floyd-Warshall's algorithm (dynamic programming)
 - Dijkstra's Single-Source algorithm

Matrix Multiplication Algorithm

- Assume that $G = (V, E)$ has no negative weight edges
- Define $P_{i,j}^{(k)}$ to be the shortest path from v_i to v_j using *at most* k edges, and let $d_{i,j}^{(k)}$ denote the cost of this path
- Thus, we set $d_{i,j}^{(1)} = w_{i,j}$ from adjacency matrix A
- We can recursively find each $d_{i,j}^{(k)}$ by using

$$d_{i,j}^{(k)} = \min\{d_{i,j}^{(k-1)}, \min_{1 \leq m \leq n} \{d_{i,m}^{(k-1)} + w_{m,j}\}\}$$

or, simply,

$$d_{i,j}^{(k)} = \min_{1 \leq m \leq n} \{d_{i,m}^{(k-1)} + w_{m,j}\}$$

The final solution to the problem is then given by the $d_{i,j}^{(n-1)}$.

Solution Strategy

- Let all the $d_{i,j}^{(k-1)}$ for a fixed k be stored as a matrix $D^{(k-1)}$
- Then the **product** $D^{(k-1)} \cdot A$, where A is the graph's adjacency matrix yields the desired $D_{i,j}^{(k)}$

$$d_{i,j}^{(k)} = \min_{1 \leq m \leq n} \{d_{i,m}^{(k-1)} + w_{m,j}\} \quad (\text{Shortest Path})$$

$$c_{i,j} = \sum_{k=1}^n a_{i,k} \cdot b_{k,j} \quad (\text{Matrix Multiply})$$

Hence, we replace \sum by \min , and \cdot by $+$ in the matrix multiply.

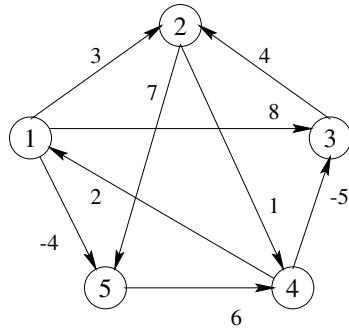
- Since $D^{(1)} = A$, $D^{(2)} = D^{(1)} \cdot A$, and so on, $D^{(k)} = A^k$, so we really just need to compute $A^k = A \cdot A \cdots A$.
- That can be done recursively: A^2, A^4, A^8, A^{16} , etc... so only $\lceil \lg(n-1) \rceil$ modified multiplications are needed.
- The serial time complexity is $O(n^3) \times O(\lg n)$.
- The calculation of the matrix products can be performed in parallel using Cannon or Fox's algorithm with checkboard partitioning.

The Floyd-Warshall Algorithm

- This dynamic programming approach keeps track of *intermediate vertices* through which a path can travel
- Let $d_{i,j}^{(k)}$ denote the minimum weight path from v_i to v_j using intermediate nodes $\{v_1, v_2, \dots, v_k\}$.
- Then

$$d_{i,j}^{(k)} = \begin{cases} w(i, j) & k = 0 \\ \min\{d_{i,j}^{(k-1)}, d_{i,k}^{(k-1)} + d_{k,j}^{(k-1)}\} & k \geq 1 \end{cases}$$

An Example



$$A = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

The matrix $D^{(0)}$ contains the minimum cost of paths that travel through no intermediate nodes:

$$D^{(0)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad D^{(0)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

$$d_{i,j}^{(1)} = \min\{d_{i,j}^{(0)}, d_{i,1}^{(0)} + d_{1,j}^{(0)}\}$$

$$D^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \begin{array}{l} \text{Row1: } d_{1,j}^{(1)} = \min\{d_{1,j}^{(0)}, d_{1,1}^{(0)} + d_{1,j}^{(0)}\} \\ \text{Row2: } d_{2,j}^{(1)} = \min\{d_{2,j}^{(0)}, d_{2,1}^{(0)} + d_{1,j}^{(0)}\} \\ \text{Row3: } d_{3,j}^{(1)} = \min\{d_{3,j}^{(0)}, d_{3,1}^{(0)} + d_{1,j}^{(0)}\} \\ \text{Row4: } d_{4,j}^{(1)} = \min\{d_{4,j}^{(0)}, d_{4,1}^{(0)} + d_{1,j}^{(0)}\} \\ \text{Row5: } d_{5,j}^{(1)} = \min\{d_{5,j}^{(0)}, d_{5,1}^{(0)} + d_{1,j}^{(0)}\} \end{array}$$

The matrix $D^{(1)}$ contains the minimum cost of paths that can travel through node 1 as an intermediate node.

$$D^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad D^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

$$d_{i,j}^{(2)} = \min\{d_{i,j}^{(1)}, d_{i,2}^{(1)} + d_{2,j}^{(1)}\}$$

$$D^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \begin{array}{l} \text{Row1 : } d_{1,j}^{(2)} = \min\{d_{1,j}^{(1)}, d_{1,2}^{(1)} + d_{2,j}^{(1)}\} \\ \text{Row2 : } d_{2,j}^{(2)} = \min\{d_{2,j}^{(1)}, d_{2,2}^{(1)} + d_{2,j}^{(1)}\} \\ \text{Row3 : } d_{3,j}^{(2)} = \min\{d_{3,j}^{(1)}, d_{3,2}^{(1)} + d_{2,j}^{(1)}\} \\ \text{Row4 : } d_{4,j}^{(2)} = \min\{d_{4,j}^{(1)}, d_{4,2}^{(1)} + d_{2,j}^{(1)}\} \\ \text{Row5 : } d_{5,j}^{(2)} = \min\{d_{5,j}^{(1)}, d_{5,2}^{(1)} + d_{2,j}^{(1)}\} \end{array}$$

The matrix $D^{(2)}$ contains the minimum cost of paths that can travel through nodes 1 and 2 as intermediate nodes.

Time Complexity

- If $D^{(k)}$ denotes the matrix of $d_{i,j}^{(k)}$ values, then we need to compute these matrices

$$D^{(0)}, D^{(1)}, D^{(2)}, D^{(3)}, \dots, D^{(n)}$$

- This requires storage for each successive pair of matrices: $D^{(k-1)}$ and $D^{(k)}$.
- The serial computation would take $\Theta(n^3)$ time.

Simple Parallel Formulations

Using a checkboard approach (with one $d_{i,j}$ per processor) on $\sqrt{p} \times \sqrt{p}$ processors:

- Note that computing $d_{i,j}^{(k)}$ requires all the values of $d_{i,-}^{(k-1)}$ and $d_{-,j}^{(k-1)}$
- During the k^{th} iteration, we could have all \sqrt{p} processors send their data to the $\sqrt{p} - 1$ processors in the same row
- Then repeat for columns, thus requiring One-to-All Broadcasts twice

Using a checkboard approach (with $\frac{n}{p}$ of the $d_{i,j}$ per processor):

- Broadcasts (with the mesh mapped onto a *hypercube*) and $\frac{n}{\sqrt{p}}$ message size:

$$\Theta\left(\frac{n}{\sqrt{p}} \lg p\right)$$

- Each processor computes n^2/p elements of the $D^{(k)}$ matrix:
- Repeating n times:

$$\text{Total time} = \Theta\left(\frac{n^3}{p}\right) + \Theta\left(\frac{n^2}{\sqrt{p}} \lg p\right)$$

- The time can be improved to

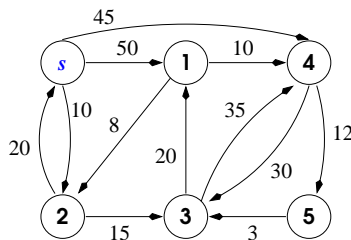
$$\text{Time} = O\left(\frac{n^3}{p}\right) + O(n)$$

if a pipelined block partitioned approach is used (which is not required for this project.)

Single-Source Shortest Paths: Dijkstra's Algorithm

Given a weighted directed graph and a *start* node s :

- Let the cost of a path between two nodes be the sum of the weights of the edges in the path.
- How can we find the minimum cost (shortest) path from the s to *all* the other nodes in the graph?



Important Property

The minimum cost path between two nodes contains other minimum cost paths within it.

Using this property and the same greedy approach used in Prim's algorithm for minimum spanning trees, we can formulate an algorithm for the *single-source* minimum cost path problem.

Let $w_{i,j}$ denote the **cost** or weight of the edge (i, j) and let S denote the set of nodes we have already processed.

Define $d[i]$ to denote the cost of the cheapest path from node s to any node i using the vertices in S .

Algorithm Dijkstra (G, w, s)

Initialize $S \leftarrow \{s\}$ and $d[s] \leftarrow 0$

for v in $V - \{s\}$ **do** $d[v] \leftarrow w_{s,v}$

while $S \neq V$ **do**

 Choose vertex u such that $d[u] = \min d[w]$ for all w in $V - S$

 Add u to the set S

for every w in $V - S$ **do**

$d[w] \leftarrow \min(d[w], d[u] + w_{u,w})$

endfor

endwhile

$\Rightarrow O(n^2)$ time complexity if an adjacency matrix is used

Using Dijkstra's Algorithm

- We can utilize Dijkstra's Single-Source Shortest Path algorithm to find all pairs of shortest paths
- There are two ways to adapt the method:
 - Source partition
 - Source parallel

Source Partitioned Adaptation

- Use n processors, P_i , for $|V| = n$
- Each P_i runs the Single-Source Shortest Path algorithm to find the solution for its assigned vertex v_i
- There is no communication needed between nodes
- Time complexity $O(n^2)$ in parallel
- What if $p < n$?? Each processor executes the algorithm for a set of starting vertices.
- If $p > n$ a source-parallel adaptation could be used (which is not part of this programming assignment).