Parallel Graph Algorithms
(Chapter 10)

Vivek Sarkar

Department of Computer Science
Rice University

vsarkar@cs.rice.edu
Schedule for rest of semester

- 4/11/08 - Deadline for finalizing Assignment #4
  —Send email to TA and me by tomorrow with your choices for
    - Parallel Language
    - Parallel Hardware
    - Sequential program that you’d like to parallelize
      It can be the same as one of the previous assignments, but now rewritten in a different parallel language

- 4/22/08 - In-class final exam

- 4/30/08 - Deadline for Assignment #4
Acknowledgments for today’s lecture

• Slides accompanying course textbook
  —http://www-users.cs.umn.edu/~karypis/parbook/

• John Mellor-Crummey --- COMP 422 slides from Spring 2007
Topics for Today

- Terminology and graph representations
- Minimum spanning tree, Prim's algorithm
- Shortest path, Dijkstra’s algorithm, Johnson’s algorithm
- Connected components
Terminology

• **Graph** $G = (V,E)$
  — $V$ is a finite set of points called *vertices*
  — $E$ is a finite set of *edges*

• **Undirected graph**
  — edge $e \in E$
    – unordered pair $(u,v)$, where $u,v \in V$

• **Directed graph**
  — edge $(u,v) \in E$
    – *incident from* vertex $u$
    – *incident to* vertex $v$

• **Path** from a vertex $u$ to $v$
  – a sequence $<v_0,v_1,v_2,\ldots,v_k>$ of vertices
    $v_0 = u$, $v_k = v$, and $(v_i,v_{i+1}) \in E$ for $i = 0, 1,\ldots, k-1$
  — path length = # of edges in a path
Directed and Undirected Graph Examples

undirected graph
directed graph
More Terminology

- **Connected** undirected graph
  —every pair of vertices is connected by a path.
- **Forest**: an acyclic graph
- **Tree**: a connected acyclic graph
- **Weighted graph**: graph with edge weights

- Common graph representations
  —adjacency matrix
  —adjacency list
Adjacency Matrix for Graph G = (V,E)

- $|V| \times |V|$ matrix
  - matrix element $a_{i,j} = 1$ if nodes $i$ and $j$ share an edge; 0 otherwise
  - for a weighted graph, $a_{i,j} = w_{i,j}$, the edge weight

- Requires $\Theta(|V|^2)$ space

Undirected graph

Adjacency matrix representation

adjacency matrix is symmetric about the diagonal
Adjacency List for Graph G = (V,E)

- An array $Adj[1..|V|]$ of lists
  - each list $Adj[v]$ is a list of all vertices adjacent to $v$
- Requires $\Theta(|E|)$ space
Topics for Today

• Terminology and graph representations
• Minimum spanning tree, Prim's algorithm
• Shortest path, Dijkstra’s algorithm, Johnson’s algorithm
• Connected components
Minimum Spanning Tree

- **Spanning tree** of a connected undirected graph $G$
  - subgraph of $G$ that is a tree containing all the vertices of $G$
  - if graph is not connected: spanning forest

- **Weight of a subgraph** in a weighted graph
  - sum of the weights of the edges in the subgraph

- **Minimum spanning tree** (MST) for weighted undirected graph
  - spanning tree with minimum weight
Minimum Spanning Tree

Undirected graph

Minimum spanning tree
Computing a Minimum Spanning Tree

Prim's sequential algorithm

1. procedure PRIM_MST(V, E, w, r)
2. begin
3. \[ V_T := \{ r \} \]: // initialize spanning tree vertices \( V_T \) with vertex \( r \), the designated root
4. \[ d[r] := 0; \] // compute \( d[:] \), the weight between
5. for all \( v \in (V - V_T) \) do // weight between
6. \[ \text{if edge } (r, v) \text{ exists set } d[v] := w(r, v); \] // r and each
7. \[ \text{else set } d[v] := \infty; \] // vertex outside \( V_T \)
8. while \( V_T \neq V \) do // while there are vertices outside \( T \)
9. begin
10. find a vertex \( u \) such that \( d[u] := \min\{d[v] | v \in (V - V_T)\} \); // recompute \( d[:] \) now
11. \[ V_T := V_T \cup \{ u \} ; \] // add \( u \) to \( T \)
12. for all \( v \in (V - V_T) \) do // that \( u \) is in \( T \)
13. \[ d[v] := \min\{d[v], w(u, v)\}; \] // use \( d[:] \) to find \( u \), // vertex closest to \( T \)
14. endwhile
15. end PRIM_MST
Parallel Formulation of Prim's Algorithm

• Parallelization prospects
  — outer loop (|V| iterations): hard to parallelize
    — adding 2 vertices to tree concurrently is problematic
  — inner loop: relatively easy to parallelize
    — consider which vertex is closest to MST in parallel

• Approach
  — data partitioning
    — partition adjacency matrix in a 1-D block fashion (blocks of columns)
    — partition distance vector d accordingly
  — in each step,
    — process first identifies the locally closest node
    — performs a global reduction to select globally closest node
    — leader inserts node into MST
    — broadcasts choice to all processes
    — each process updates its part of d vector locally based on choice
Parallel Formulation of Prim's Algorithm

**distance array** $d[1..n]$  
(a)

**adjacency matrix** $A$  
(b)

*partition $d$ and $A$ among $p$ processes*
Parallel Formulation of Prim's Algorithm

• Cost to select the minimum entry
  — $O(n/p)$: scan $n/p$ local part of $d$ vector on each processor
  — $O(\log p)$ all-to-one reduction across processors

• Broadcast next node selected for membership
  — $O(\log p)$

• Cost of locally updating $d$ vector
  — $O(n/p)$: replace $d$ vector with min of $d$ vector and matrix row

• Parallel time per iteration
  — $O(n/p + \log p)$

• Total parallel time
  — $O(n^2/p + n \log p)$
Minimum Spanning Tree: Prim's Algorithm

start with arbitrary root

2 iterations of Prim’s algorithm
Algorithms for Sparse Graphs

• Dense algorithms can be improved significantly if we make use of the sparseness

• Example: Prim’s algorithm complexity
  — can be reduced to $O(|E| \log n)$
    - use heap to maintain costs
    - outperforms original as long as $|E| = O(n^{2/ \log n})$

• Sparse algorithms: use adjacency list instead of matrix

• Partitioning adjacency lists is more difficult for sparse graphs
  — do we balance number of vertices or edges?

• Parallel algorithms typically make use of graph structure or degree information for performance
Algorithms for Sparse Graphs

Graph $G = (V,E)$ is sparse if $|E|$ is much smaller than $|V|^2$

Examples of sparse graphs: (a) a linear graph, in which each vertex has two incident edges; (b) a grid graph, in which each vertex has four incident vertices; and (c) a random sparse graph.
Topics for Today

• Terminology and graph representations
• Minimum spanning tree, Prim's algorithm
• Shortest path, Dijkstra’s algorithm, Johnson’s algorithm
• Connected components
Single-Source Shortest Paths

• Given weighted graph $G = (V,E,w)$

• Problem: *single-source shortest paths*
  — find the shortest paths from vertex $v \in V$ to all other vertices in $V$

• Dijkstra's algorithm: similar to Prim's algorithm
  — maintains a set of nodes for which the shortest paths are known
  — grows set by adding node closest to source using one of the nodes in the current shortest path set
Computing Single-Source Shortest Paths

Dijkstra’s sequential algorithm

1. procedure DIJKSTRA_SINGLE_SOURCE_SP($V$, $E$, $w$, $s$)
2. begin
3. $V_T := \{s\}$; // initialize tree vertices $V_T$ with vertex $s$, the designated src
4. for all $v \in (V - V_T)$ do // compute $l[\cdot]$, the
5. if $(s, v)$ exists set $l[v] := w(s, v)$; // weight between
6. else set $l[v] := \infty$; // $s$ and each vertex $\notin V_T$
7. while $V_T \neq V$ do // while some vertices are not in $V_T$
8. begin
9. find a vertex $u$ such that $l[u] := \min \{l[v] \mid v \in (V - V_T)\}$;
10. $V_T := V_T \cup \{u\}$; // add $u$ to $T$
11. for all $v \in (V - V_T)$ do // recompute $l[\cdot]$
12. $l[v] := \min \{l[v], l[u] + w(u, v)\}$; // now that $u$ is in $T$
13. endwhile
14. end DIJKSTRA_SINGLE_SOURCE_SP

// use $l[\cdot]$ to find $u$,
// next vertex closest
// src
Parallel Formulation of Dijkstra's Algorithm

Similar to parallel formulation of Prim's algorithm for MST

• Approach
  — data partitioning
    – partition weighted adjacency matrix in a 1-D block fashion
    – partition distance vector $L$ accordingly
  — in each step,
    – each process identifies its node closest to source
    – perform a global reduction to select globally closest node
    – broadcasts choice broadcast to all processes
    – each process updates its part of $L$ vector locally

• Parallel performance of Dijkstra's algorithm
  — identical to that of Prim's algorithm
    – parallel time per iteration: $O(n/p + \log p)$
    – total parallel time: $O(n^2/p + n \log p)$
All-Pairs Shortest Paths

• Given weighted graph $G(V,E,w)$

• Problem: all-pairs shortest paths
  — find the shortest paths between all pairs of vertices $v_i, v_j \in V$

• Several algorithms known for solving this problem
All-Pairs Shortest Path

Serial formulation using Dijkstra’s algorithm

• Execute $n$ instances of the single-source shortest path —one for each of the $n$ source vertices

• Sequential time per source: $O(n^2)$

• Total sequential time complexity: $O(n^3)$
All-Pairs Shortest Path

Parallel formulation using Dijkstra’s algorithm

Two possible parallelization strategies

• **Source partitioned**: execute each of the $n$ shortest path problems on a different processor

• **Source parallel**: use a parallel formulation of the shortest path problem to increase concurrency
All-Pairs Shortest Path Dijkstra's Algorithm

“Source partitioned” parallel formulation

• Use $n$ processors
  — each processor $P_i$ finds the shortest paths from vertex $v_i$ to all other vertices
    – use Dijkstra's sequential single-source shortest paths algorithm

• Analysis
  — requires no interprocess communication
    – provided adjacency matrix is replicated at all processes
  — parallel run time: $\Theta(n^2)$

• Algorithm is cost optimal
  — asymptotically same # of ops in parallel as in sequential version

• However: can only use $n$ processors (one per source)
"Source parallel” parallel formulation

- Each of the shortest path problems is executed in parallel — can therefore use up to \( n^2 \) processors.
- Given \( p \) processors (\( p > n \)) — each single source shortest path problem is executed by \( p/n \) processors.
- Recall time for solving one instance of all-pair shortest path — \( O(n^2/p + n \log p) \)
- Considering the time to do one instance on \( p/n \) processors
  
  \[
  T_P = \Theta \left( \frac{n^3}{p} \right) + \Theta(n \log p).
  \]
- Represents total time since each instance is solved in parallel
• Dijkstra's algorithm, modified to handle sparse graphs is called Johnson's algorithm.

• The modification accounts for the fact that the minimization step in Dijkstra's algorithm needs to be performed only for those nodes adjacent to the previously selected nodes.

• Johnson's algorithm uses a priority queue $Q$ to store the value $l[v]$ for each vertex $v \in (V - V_T)$. 
Single-Source Shortest Paths: 
Johnson's Algorithm

1. procedure \texttt{JOHNSON\_SINGLE\_SOURCE\_SP}(V, E, s)
2. begin
3. \hspace{1em} \texttt{Q} := V;
4. \hspace{1em} for all \texttt{v} \in \texttt{Q} do
5. \hspace{2em} \texttt{l}[v] := \infty;
6. \hspace{1em} \texttt{l}[s] := 0;
7. \hspace{1em} while \texttt{Q} \neq \emptyset do
8. \hspace{2em} begin
9. \hspace{3em} \texttt{u} := \texttt{extract\_min}(\texttt{Q});
10. \hspace{3em} for each \texttt{v} \in \texttt{Adj}[u] do
11. \hspace{4em} if \texttt{v} \in \texttt{Q} and \texttt{l}[u] + \texttt{w}(u, v) < \texttt{l}[v] then
12. \hspace{5em} \texttt{l}[v] := \texttt{l}[u] + \texttt{w}(u, v);
13. \hspace{4em} endwhile
14. \hspace{2em} end \texttt{JOHNSON\_SINGLE\_SOURCE\_SP}
Single-Source Shortest Paths: Parallel Johnson's Algorithm

- Maintaining strict order of Johnson's algorithm generally leads to a very restrictive class of parallel algorithms.

- We need to allow exploration of multiple nodes concurrently. This is done by simultaneously extracting \( p \) nodes from the priority queue, updating the neighbors' cost, and augmenting the shortest path.

- If an error is made, it can be discovered (as a shorter path) and the node can be reinserted with this shorter path.
Single-Source Shortest Paths: Parallel Johnson's Algorithm

An example of the modified Johnson's algorithm for processing unsafe vertices concurrently.

**Priority Queue**

1. \( b:1, d:7, c:\text{inf}, e:\text{inf}, f:\text{inf}, g:\text{inf}, h:\text{inf}, i:\text{inf} \)
2. \( e:3, c:4, g:10, f:\text{inf}, h:\text{inf}, i:\text{inf} \)
3. \( h:4, f:6, i:\text{inf} \)
4. \( g:5, i:6 \)

**Array l[]**

\[
\begin{array}{cccccccc}
  a & b & c & d & e & f & g & h & i \\
  \infty & 1 & \infty & 7 & \infty & \infty & \infty & \infty & 3 \\
  0 & 1 & 4 & 7 & 3 & 10 & \infty & \infty & 4 \\
  0 & 1 & 4 & 7 & 3 & 6 & 10 & 4 & \infty \\
  0 & 1 & 4 & 7 & 3 & 6 & 5 & 4 & 6 \\
\end{array}
\]
• Even if we can extract and process multiple nodes from the queue, the queue itself is a major bottleneck.

• For this reason, we use multiple queues, one for each processor. Each processor builds its priority queue only using its own vertices.

• When process $P_i$ extracts the vertex $u \in V_i$, it sends a message to processes that store vertices adjacent to $u$.

• Process $P_j$, upon receiving this message, sets the value of $l[v]$ stored in its priority queue to $\min\{l[v], l[u] + w(u, v)\}$.
• If a shorter path has been discovered to node $v$, it is reinserted back into the local priority queue.

• The algorithm terminates only when all the queues become empty.

• A number of node paritioning schemes can be used to exploit graph structure for performance.
Topics for Today

• Terminology and graph representations
• Minimum spanning tree, Prim's algorithm
• Shortest path, Dijkstra’s algorithm, Johnson’s algorithm
• Connected components
**Connected Components**

*Definition*: equivalence classes of vertices under the “is reachable from” relation for undirected graphs

Example: graph with three connected components

\{1,2,3,4\}, \{5,6,7\}, and \{8,9\}
Connected Components

Serial depth-first search based algorithm

- Perform DFS on a graph to get a forest
  — each tree in the forest = separate connected component.

Depth-first forest above obtained from depth-first traversal of the graph at top. Result = 2 connected components
Connected Components Parallel Formulation

• Partition the graph across processors

• Step 1
  — run independent connected component algorithms on each processor
  — result: $p$ spanning forests.

• Step 2
  — merge spanning forests pairwise until only one remains
Connected Components Parallel Formulation

1. Partition adjacency matrix of the graph $G$ into two parts

2. Each process gets a subgraph of graph $G$

3. Each process computes the spanning forest of its subgraph of $G$

4. Merge the two spanning trees to form the final solution
Connected Components Parallel Formulation

- Merge pairs of spanning forests using disjoint sets of vertices
- Consider the following operations on the disjoint sets
  - `find(x)`
    - returns pointer to representative element of the set containing `x`
    - each set has its own unique representative
  - `union(x, y)`
    - merges the sets containing the elements `x` and `y`
    - the sets are assumed disjoint prior to the operation
Connected Components Parallel Formulation

• To merge forest $A$ into forest $B$
  — for each edge $(u,v)$ of $A$,
    – perform $\textit{find}$ operations on $u$ and $v$
      determine if $u$ and $v$ are in same tree of $B$
    – if not, then union the two trees (sets) of $B$ containing $u$ and $v$
      result: $u$ and $v$ are in same set, which means they are connected
    – else, no $\textit{union}$ operation is necessary.

• Merging forest $A$ and forest $B$ requires at most
  — $2(n-1)$ $\textit{find}$ operations
  — $(n-1)$ $\textit{union}$ operations

  at most $n-1$ edges must be considered because $A$ and $B$ are forests
Connected Components

Analysis of parallel 1-D block mapping

• Partition an $n \times n$ adjacency matrix into $p$ blocks

• Each processor computes local spanning forest: $\Theta(n^2/p)$

• Merging approach
  — embed a logical tree into the topology
    – $\log p$ merging stages
    – each merge stage takes time $\Theta(n)$
  — total cost due to merging is $\Theta(n \log p)$

• During each merging stage
  — spanning forests are sent between nearest neighbors
  — $\Theta(n)$ edges of the spanning forest are transmitted

• Parallel execution time

$$T_P = \Theta\left(\frac{n^2}{p}\right) + \Theta(n \log p).$$
Summary

• Terminology and graph representations
• Minimum spanning tree, Prim's algorithm
• Shortest path, Dijkstra’s algorithm, Johnson’s algorithm
• Connected components