Introduction to Parallel Computing

George Karypis
Graph Algorithms
Outline

- Graph Theory Background
- Minimum Spanning Tree
  - Prim’s algorithm
- Single-Source Shortest Path
  - Dijkstra’s algorithm
- All-Pairs Shortest Path
  - Dijkstra’s algorithm
  - Floyd’s algorithm
- Maximal Independent Set
  - Luby’s algorithm
Background

Figure 10.1 (a) An undirected graph and (b) a directed graph.

Figure 10.2 An undirected graph and its adjacency matrix representation.

Figure 10.3 An undirected graph and its adjacency list representation.
Minimum Spanning Tree

- Compute the minimum weight spanning tree of an undirected graph.

Figure 10.4 An undirected graph and its minimum spanning tree.
Prim’s Algorithm

- Prim’s Algorithm
  - Θ(n²) serial complexity for dense graphs.
    - why?
- How can we parallelize this algorithm?
- Which steps can be done in parallel?

1. procedure PRIM_MST(V, E, w, r)
2. begin
3.   V_T := {r};
4.   d[r] := 0;
5.   for all v ∈ (V – V_T) do
6.     if edge (r, v) exists set d[v] := w(r, v);
7.     else set d[v] := ∞;
8.   while V_T ≠ V do
9.     begin
10.    find a vertex u such that d[u] := min{d[v] | v ∈ (V – V_T)};
11.    V_T := V_T ∪ {u};
12.    for all v ∈ (V – V_T) do
13.       d[v] := min{d[v], w(u, v)};
14.   end
15. end PRIM_MST
Parallel Formulation of Prim’s Algorithm

- Parallelize the inner-most loop of the algorithm.
  - Parallelize the selection of the “minimum weight edge” connecting an edge in $V_T$ to a vertex in $V-V_T$.
  - Parallelize the updating of the $d[]$ array.

- What is the maximum concurrency that such an approach can use?

- How do we “implement” it on a distributed-memory architecture?

```plaintext
1. procedure PRIM_MST($V$, $E$, $w$, $r$)
2. begin
3.   $V_T := \{r\}$;
4.   $d[r] := 0$;
5.   for all $v \in (V - V_T)$ do
6.     if edge $(r, v)$ exists set $d[v] := w(r, v)$;
7.     else set $d[v] := \infty$;
8.   while $V_T \neq V$ do
9.     begin
10.        find a vertex $u$ such that $d[u] := \min[d[v] | v \in (V - V_T)]$;
11.        $V_T := V_T \cup \{u\}$;
12.        for all $v \in (V - V_T)$ do
13.            $d[v] := \min[d[v], w(u, v)]$;
14.     endwhile
15.   end PRIM_MST
```
Parallel Formulation of Prim’s Algorithm

- Decompose the graph $A$ (adjacency matrix) and vector $d$ vector using a 1D block partitioning along columns.
  
  - Why columns?

- Assign each block of size $n/p$ to one of the processors.

- How will lines 10 & 12—13 be performed?

- Complexity? Memory Complexity?

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

Find $d[]$ in $O(\log p)$ communication steps
- on shared memory
- on a distributed architecture

Parallel Formulation of Prim’s Algorithm

1. procedure PRIM_MST($V$, $E$, $w$, $r$)
2. begin
3. $V_T := \{r\}$;
4. $d[r] := 0$;
5. for all $v \in (V - V_T)$ do
6.   if edge $(r, v)$ exists set $d[v] := w(r, v)$;
7.   else set $d[v] := \infty$;
8. while $V_T \neq V$ do
9.   begin
10.      find a vertex $u$ such that $d[u] := \min[d[v]|v \in (V - V_T)]$;
11.      $V_T := V_T \cup \{u\}$;
12.      for all $v \in (V - V_T)$ do
13.         $d[v] := \min[d[v], w(u, v)]$;
14.   endwhile
15.   end PRIM_MST

Isoefficiency:
Single-Source Shortest Path

- Given a source vertex s find the shortest-paths to all other vertices.
- Dijkstra’s algorithm.
- How can it be parallelized for dense graphs?

Algorithm 10.2 Dijkstra’s sequential single-source shortest paths algorithm.
All-pairs Shortest Paths

- Compute the shortest paths between all pairs of vertices.

- Algorithms
  - Dijkstra’s algorithm
    - Execute the single-source algorithm \( n \) times.
  - Floyd’s algorithm
    - Based on dynamic programming.
All-Pairs Shortest Path
Dijkstra’s Algorithm

- Source-partitioned formulation
  - Partition the sources along the different processors.
    - Is it a good algorithm?
      - Computational & memory scalability
      - What is the maximum number of processors that it can use?

- Source-parallel formulation
  - Used when $p > n$.
  - Processors are partitioned into $n$ groups each having $p/n$ processors.
  - Each group is responsible for one single-source SP computation.
  - Complexity?

For $p \leq n$:
Complexity $T_P$?
Memory?

$$T_P = \Theta\left(\frac{n^3}{p}\right) + \Theta(n \log p).$$
Floyd’s Algorithm

- Solves the problem using a dynamic programming algorithm.
  - Let $d^{(k)}_{i,j}$ be the shortest path distance between vertices $i$ and $j$ that goes only through vertices $1,\ldots, k$.

$$d^{(k)}_{i,j} = \begin{cases} w(v_i,v_j) & \text{if } k = 0 \\ \min\left\{d^{(k-1)}_{i,j}, d^{(k-1)}_{i,k} + d^{(k-1)}_{k,j}\right\} & \text{if } k \geq 1 \end{cases}$$

- Complexity: $\Theta(n^3)$.
- Note: The algorithm can run in-place.

- How can we parallelize it?
Parallel Formulation of Floyd’s Algorithm

- Distribute the matrix using a 2D block decomposition.
- Parallelize the double inner-most loop.

1. procedure FLOYD_ALL_PAIRS.SP(A)
2. begin
3. \( D^{(0)} = A \);
4. for \( k := 1 \) to \( n \) do
5. \( i := 1 \) to \( n \) do
6. \( j := 1 \) to \( n \) do
7. \( d_{i,j}^{(k)} := \min(d_{i,j}^{(k-1)}, d_{i,k}^{(k-1)} + d_{k,j}^{(k-1)}) \);
8. end FLOYD_ALL_PAIRS.SP

Communication pattern?
Complexity?

\[ T_p = \Theta \left( \frac{n^3}{p} \right) + \Theta \left( \frac{n^2}{\sqrt{p}} \log p \right). \]

\( n \) iterations. Each process performs per iteration:
- Communication - bcast \( n/\sqrt{p} \) values from \( k^{th} \) col/row in \( \log p \) steps
- Computation - modify \( n^{1/2}/p \) values

![Communication patterns used in the 2-D block mapping. When computing \( d_{i,j}^{(k)} \), information must be sent to the highlighted process from two other processes along the same row and column. (b) The row and column of \( \sqrt{p} \) processes that contain the \( k^{th} \) row and column send them along process columns and rows.](image-url)
1. procedure FLOYD_2DBLOCK\(D^{(0)}\)
2. begin
3.     for \(k := 1\) to \(n\) do
4.         begin
5.             each process \(P_{i,j}\) that has a segment of the \(k^{th}\) row of \(D^{(k-1)}\);
6.                 broadcasts it to the \(P_{*,j}\) processes;
7.             each process \(P_{i,j}\) that has a segment of the \(k^{th}\) column of \(D^{(k-1)}\);
8.                 broadcasts it to the \(P_{i,*}\) processes;
9.             each process \(P_{i,j}\) waits to receive the needed segments;
10.         end
11.     end FLOYD_2DBLOCK

**Algorithm 10.4** Floyd's parallel formulation using the 2-D block mapping. \(P_{*,j}\) denotes all the processes in the \(j^{th}\) column, and \(P_{i,*}\) denotes all the processes in the \(i^{th}\) row. The matrix \(D^{(0)}\) is the adjacency matrix.
## Comparison of All-Pairs SP Algorithms

### Table 10.1
The performance and scalability of the all-pairs shortest paths algorithms on various architectures with $O(p)$ bisection bandwidth. Similar run times apply to all $k - d$ cube architectures, provided that processes are properly mapped to the underlying processors.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Maximum Number of Processes for $E = \Theta(1)$</th>
<th>Corresponding Parallel Run Time</th>
<th>Isoefficiency Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dijkstra source-partitioned</td>
<td>$\Theta(n)$</td>
<td>$\Theta(n^2)$</td>
<td>$\Theta(p^3)$</td>
</tr>
<tr>
<td>Dijkstra source-parallel</td>
<td>$\Theta(n^2 / \log n)$</td>
<td>$\Theta(n \log n)$</td>
<td>$\Theta((p \log p)^{1.5})$</td>
</tr>
<tr>
<td>Floyd 1-D block</td>
<td>$\Theta(n / \log n)$</td>
<td>$\Theta(n^2 \log n)$</td>
<td>$\Theta((p \log p)^3)$</td>
</tr>
<tr>
<td>Floyd 2-D block</td>
<td>$\Theta(n^2 / \log^2 n)$</td>
<td>$\Theta(n \log^5 n)$</td>
<td>$\Theta(p^{1.5} \log^3 p)$</td>
</tr>
<tr>
<td>Floyd pipelined 2-D block</td>
<td>$\Theta(n^2)$</td>
<td>$\Theta(n)$</td>
<td>$\Theta(p^{1.5})$</td>
</tr>
</tbody>
</table>
Maximal Independent Sets

- Find the maximal set of vertices that are not adjacent to each other.

Figure 10.15  Examples of independent and maximal independent sets.
Serial Algorithms for MIS

- Practical MIS algorithms are incremental in nature.
  - Start with an empty set.
  1. Add the vertex with the smallest degree.
  2. Remove adjacent vertices
  3. Repeat 1—2 until the graph becomes empty.
- These algorithms are impossible to parallelize.
  - Why?
- Parallel MIS algorithms are based on the ideas initially introduced by Luby.
Luby’s MIS Algorithm

- Randomized algorithm.
  - Starts with an empty set.
  - Assigns random numbers to each vertex.
  - Vertices whose random number are smaller than all of the numbers assigned to their adjacent vertices are included in the MIS.
  - Vertices adjacent to the newly inserted vertices are removed.
  - Repeat steps 1—3 until the graph becomes empty.

- This algorithm will terminate in $O(\log (n))$ iterations.

- Why is this a good algorithm to parallelize?
- How will the parallel formulation proceed?
  - Shared memory
  - Distributed memory

Figure 10.16 The different augmentation steps of Luby’s randomized maximal independent set algorithm. The numbers inside each vertex correspond to the random number assigned to the vertex.