Introduction to Parallel Computing

George Karypis
Graph Algorithms
Previous Lectures

• Why concurrent and distributed programming?
• Amdahl’s vs. Gustafson's law
• Problem partitioning: SIMD vs. MIMD
• Memory architecture: UMA, NUMA and distributed
  • Memory-optimized programming
  • Memory latency vs. bandwidth
• Cache
  • Cache coherence
  • False sharing
  • Superlinear speedup
• Granularity and load balancing
  • Control: task management overhead
  • Synchronization overhead
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
Reminder (1)

- (a) An undirected graph
- (b) A directed graph
Reminder (2)

- An undirected graph and its adjacency list representation
Reminder (3)

• An undirected graph and its adjacency matrix representation
  • The number can also represent the weight of the edge.
  • What is the weight of vertexes with no edge between them?

![Graph](image)

\[ A = \begin{bmatrix}
0 & 1 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 1 \\
0 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 \\
0 & 1 & 1 & 1 & 0
\end{bmatrix} \]
Reminder (4)

• Which representation is better?
Minimum Spanning Tree

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

An undirected graph

Its minimum spanning tree
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. \hspace{1em} if edge \((r, v)\) exists set \( d[v] := w(r, v); \)
7. \hspace{1em} else set \( d[v] := \infty; \)
8. while \( V_T \neq V \) do
9. \hspace{1em} begin
10. \hspace{2em} find a vertex \( u \) such that \( d[u] := \min\{d[v] | v \in (V - V_T)\}; \)
11. \hspace{2em} \( V_T := V_T \cup \{u\}; \)
12. \hspace{2em} for all \( v \in (V - V_T) \) do
13. \hspace{3em} \( d[v] := \min\{d[v], w(u, v); \)
14. \hspace{2em} endwhile
15. end PRIM_MST
Prim’s Algorithm

- Θ(n²) serial complexity for dense graphs.
- 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.         endwhile
15.     end PRIM_MST
```

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

• Parallelize the inner-most loops of the algorithm.
  • Parallelize the initialization
  • Parallelize the selection of the “minimum weight edge”.
  • 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?

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

Initialization

Minimum selection

Update d[]

<table>
<thead>
<tr>
<th>d[]</th>
<th>a  b  c  d  e  f</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>d[ ]</th>
<th>a  b  c  d  e  f</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
</tr>
<tr>
<td>c</td>
<td>3</td>
</tr>
<tr>
<td>d</td>
<td>\infty</td>
</tr>
<tr>
<td>e</td>
<td>\infty</td>
</tr>
<tr>
<td>f</td>
<td>2</td>
</tr>
</tbody>
</table>
Parallel Formulation of Prim’s Algorithm

• Domain decomposition
  • Decompose the vector \( d \)
  • Decompose the graph’s adjacency matrix \( (A) \) along the columns
    • Why columns?
  • Assign each block of size \( \frac{n}{p} \) to one of the processors.

• How will lines 5-7, 10 & 12-13 be performed?
  • Complexity?

\[
T_P = \Theta \left( \frac{n^2}{p} \right) + \Theta(n \log p).
\]
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?
  • Very similar to Prim’s parallel version
  • The same parallel performance of Prim’s

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

**procedure** DIJKSTRA_SINGLE_SOURCE_SP\((V, E, \omega, s)\)

1. \( V_T := \{s\} \)
2. \( l[r] := 0 \)
3. \( \text{for all } v \in (V - V_T) \text{ do} \)
4. \( \text{if edge}(r, v) \text{ exists, set } l[v] := \omega(r, v) \)
5. \( \text{else set } l[v] := \infty \)
6. \( \text{while } V_T \neq V \text{ do} \)
7. \( \text{find a vertex } u \text{ such that } l[u] := \min\{l[v] | v \in (V - V_T)\} \)
8. \( V_T := V_T \cup \{u\} \)
9. \( \text{for all } v \in (V - V_T) \text{ do} \)
10. \( l[v] := \min\{l[v], l[u] + \omega(u, v)\} \)
All-Pairs Shortest Path Dijkstra’s Algorithm

- Compute the shortest paths between all pairs of vertices.
- Execute the single-source algorithm $n$ times. $T = \Theta(n^3)$

- 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 $\frac{p}{n}$ processors.
  - Each group is responsible for one single-source computation.
  - Complexity?

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

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

  $$d_{i,j}^{(k)} = \begin{cases} w(v_i, v_j) & \text{if } k = 0 \\ \min_{1 \leq i, j \leq n} \{d_{i,j}^{k-1}, d_{i,k}^{k-1} + d_{k,j}^{k-1} \} & \text{if } k \geq 1 \end{cases}$$

• Complexity: $\Theta(n^3)$
• Note: The algorithm can run in-place.
• How can we parallelize it?

```procedure FLOYD_ALLPAIRS_SP(A):
  1. $D^0 = A$
  2. for $k := 1$ to $n$ do
  3.     for $i := 1$ to $n$ do
  4.         for $j := 1$ to $n$ do
  5.             $d_{i,j}^{(k)} = \min\{d_{i,j}^{k-1}, d_{i,k}^{k-1} + d_{k,j}^{k-1} \}$
```
Parallel Formulation of Floyd’s Algorithm (1)

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

```
procedure FLOYD_ALL_PAIRS_SP(A):
1.  \( D^0 = A \)
2.  for \( k := 1 \) to \( n \) do
3.    for \( i := 1 \) to \( n \) do
4.      for \( j := 1 \) to \( n \) do
5.        \( d_{i,j}^{(k)} = \min\{d_{i,j}^{k-1}, d_{i,k}^{k-1} + d_{k,j}^{k-1}\} \)
```

Matrix \( D^{(k)} \) distributed by 2-D block mapping into \( \sqrt{p} \times \sqrt{p} \) subblocks

The sub-block of \( D^{(k)} \) assigned to process \( P_{i,j} \)
Parallel Formulation of Floyd’s Algorithm (2)

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

```plaintext
procedure FLOYD_ALL_PAIRS_SP(A):
1. \(D^0 = A\)
2. for \(k := 1\) to \(n\) do
3. for \(i := 1\) to \(n\) do
4. for \(j := 1\) to \(n\) do
5. \(d_{i,j}^{(k)} = \min\{d_{i,j}^{k-1}, d_{i,k}^{k-1} + d_{k,j}^{k-1}\}\)
```

**Figure 10.8** (a) 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.
Parallel Formulation of Floyd’s Algorithm (3)

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 \(D^{(k-1)}\); broadcasts it to the \(P_{*,*}\) processes;
6. each process \(P_{i,j}\) that has a segment of the \(k\)th column of \(D^{(k-1)}\); broadcasts it to the \(P_{i,*}\) processes;
7. each process waits to receive the needed segments;
8. each process \(P_{i,j}\) computes its part of the \(D^{(k)}\) matrix;
9. end
10. 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.
Parallel Formulation of Floyd’s Algorithm

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

```plaintext
procedure FLOYD_ALL_PAIRS_SP(A):
1. \( D^0 = A \)
2. for \( k := 1 \) to \( n \) do
3.     for \( i := 1 \) to \( n \) do
4.         for \( j := 1 \) to \( n \) do
5.             \( d^{(k)}_{i,j} = \min\{d^{k-1}_{i,j}, d^{k-1}_{i,k} + d^{k-1}_{k,j}\} \)
```

- Communication pattern? Complexity?

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

Figure 10.7 (a) Matrix \( D^{(k)} \) distributed by 2-D block mapping into \( \sqrt{p} \times \sqrt{p} \) subblocks, and (b) the subblock of \( D^{(k)} \) assigned to process \( P_{i,j} \).

Figure 10.8 (a) Communication patterns used in the 2-D block mapping. When computing \( d^{(k)}_{i,j} \), 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.
Maximal Independent Sets (MIS)

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

\{a, d, i, h\} is not an independent set
\{a, d\} is an independent set, but not maximal

\{a, c, j, f, g\} is a maximal independent set
\{a, d, h, f\} is a maximal independent set
Serial Algorithms for MIS

• Practical MIS algorithms are incremental in nature.

procedure MIS(V, E):
1. \( I = \{\} \)
2. while \( V \) is not empty do
3. choose a node \( v \in V \)
4. \( I = I \cup \{v\} \)
5. remove from \( V \) the node \( v \) and all its neighbors
6. return \( I \)

• The runtime is \( O(n) \) since in the worst case we have to check all nodes.

• How can we parallelize it?
  • Can we achieve \( O\left(\frac{n}{p}\right) \) complexity?
Parallel Algorithm for MIS

• These algorithms are impossible to parallelize.
  • Why?
• Parallel MIS algorithms are based on the ideas initially introduced by Luby.

```plaintext
procedure MIS(V, E):
1.  I = {}
2.  while V is not empty do
3.      choose a node ν ∈ V
4.      I = I ∪ {ν}
5.      remove from V the node ν and all its neighbors
6.  return I
```
Luby’s MIS Algorithm (Randomized Algorithm)

• Starts with an empty set.
1. Assigns random numbers to each vertex.
2. Vertices whose random number are smaller than all of the numbers assigned to their adjacent vertices are included in the MIS.
3. Vertices adjacent to the newly inserted vertices are removed.
4. Repeat steps 1—3 until the graph becomes empty.
Luby’s MIS Algorithm
Parallel Implementation

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

• On average, this algorithms 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