Compsci 590.3:
Introduction to Parallel Computing

Alvin R. Lebeck

Slides from the University of Oregon
Logistics

• Homework #4
• Homework #5
• Projects
• Project work days

Outline

• General Design Approach
• Gaussian Elimination
• Graph Algorithms
  – Minimum Spanning Tree
  – Shortest Path
• Map/Reduce

Compsci 590.3 Parallel Computing (Slides from UOregon)
Methodological Design

- **Partition**
  - Task/data decomposition
- **Communication**
  - Task execution coordination
- **Agglomeration**
  - Evaluation of the structure
- **Mapping**
  - Resource assignment

I. Foster, “Designing and Building Parallel Programs,” Addison-Wesley, 1995. Book is online, see webpage.
Partitioning

- Partitioning stage is intended to expose opportunities for parallel execution
- Focus on defining large number of small task to yield a fine-grained decomposition of the problem
- A good partition divides into small pieces both the computational tasks associated with a problem and the data on which the tasks operates
- Domain decomposition focuses on computation data
- Functional decomposition focuses on computation tasks
- Mixing domain/functional decomposition is possible
Domain and Functional Decomposition

- Domain decomposition of 2D / 3D grid

- Functional decomposition of a climate model
Partitioning Checklist

- Does your partition define at least an order of magnitude more tasks than there are processors in your target computer? If not, may loose design flexibility.
- Does your partition avoid redundant computation and storage requirements? If not, may not be scalable.
- Are tasks of comparable size? If not, it may be hard to allocate each processor equal amounts of work.
- Does the number of tasks scale with problem size? If not may not be able to solve larger problems with more processors.
- Have you identified several alternative partitions?
Communication (Interaction)

- Tasks generated by a partition must interact to allow the computation to proceed
  - Information flow: data and control

- Types of communication
  - Local vs. Global: locality of communication
  - Structured vs. Unstructured: communication patterns
  - Static vs. Dynamic: determined by runtime conditions
  - Synchronous vs. Asynchronous: coordination degree

- Granularity and frequency of communication
  - Size of data exchange

- Think of communication as interaction and control
  - Applicable to both shared and distributed memory parallelism
Types of Communication

- Point-to-point
- Group-based
- Hierarchical
- Collective
Communication Design Checklist

- Is the distribution of communications equal?
  - Unbalanced communication may limit scalability

- What is the communication locality?
  - Wider communication locales are more expensive

- What is the degree of communication concurrency?
  - Communication operations may be parallelized

- Is computation associated with different tasks able to proceed concurrently? Can communication be overlapped with computation?
  - Try to reorder computation and communication to expose opportunities for parallelism
Agglomeration

- Move from parallel abstractions to real implementation
- Revisit partitioning and communication
  - View to efficient algorithm execution
- Is it useful to agglomerate?
  - What happens when tasks are combined?
- Is it useful to replicate data and/or computation?
- Changes important algorithm and performance ratios
  - Surface-to-volume: reduction in communication at the expense of decreasing parallelism
  - Communication/computation: which cost dominates
- Replication may allow reduction in communication
- Maintain flexibility to allow overlap
Types of Agglomeration

- Element to column

- Element to block
  - Better surface to volume

- Task merging

- Task reduction
  - Reduces communication
Agglomeration Design Checklist

- Has increased locality reduced communication costs?
- Is replicated computation worth it?
- Does data replication compromise scalability?
- Is the computation still balanced?
- Is scalability in problem size still possible?
- Is there still sufficient concurrency?
- Is there room for more agglomeration?
- Fine-grained vs. coarse-grained?
Mapping

- Specify where each task is to execute
  - Less of a concern on shared-memory systems
- Attempt to minimize execution time
  - Place concurrent tasks on different processors to enhance physical concurrency
  - Place communicating tasks on same processor, or on processors close to each other, to increase locality
  - Strategies can conflict!
- Mapping problem is NP-complete
  - Use problem classifications and heuristics
- Static and dynamic load balancing
Mapping Algorithms

- **Load balancing (partitioning) algorithms**
- **Data-based algorithms**
  - Think of computational load with respect to amount of data being operated on
  - Assign data (i.e., work) in some known manner to balance
  - Take into account data interactions
- **Task-based (task scheduling) algorithms**
  - Used when functional decomposition yields many tasks with weak locality requirements
  - Use task assignment to keep processors busy computing
  - Consider centralized and decentralized schemes
Mapping Design Checklist

- Is static mapping too restrictive and non-responsive?
- Is dynamic mapping too costly in overhead?
- Does centralized scheduling lead to bottlenecks?
- Do dynamic load-balancing schemes require too much coordination to re-balance the load?
- What is the tradeoff of dynamic scheduling complexity versus performance improvement?
- Are there enough tasks to achieve high levels of concurrency? If not, processors may idle.
Outline

- General Design Approach
- Gaussian Elimination
- Graph Algorithms
  - Minimum Spanning Tree
  - Shortest Path
- Map/Reduce
Dense Matrix Algorithms

- Great deal of activity in algorithms and software for solving linear algebra problems
  - Solution of linear systems (Ax = b)
  - Least-squares solution of over- or under-determined systems (min ||Ax-b||)
  - Computation of eigenvalues and eigenvectors (Ax=λx)
  - Driven by numerical problem solving in scientific computation

- Solutions involves various forms of matrix computations

- Focus on high-performance matrix algorithms
  - Key insight is to maximize computation to communication
Solving a System of Linear Equations

- $Ax=b$
  
  $a_{0,0}x_0 + a_{0,1}x_1 + \ldots + a_{0,n-1}x_{n-1} = b_0$
  $a_{1,0}x_0 + a_{1,1}x_1 + \ldots + a_{1,n-1}x_{n-1} = b_1$
  
  $\ldots$
  
  $a_{n-1,0}x_0 + a_{n-1,1}x_1 + \ldots + a_{n-1,n-1}x_{n-1} = b_{n-1}$

- Gaussian elimination (classic algorithm)
  
  - Forward elimination to $Ux = y$ ($U$ is upper triangular)
    - without or with partial pivoting
  
  - Back substitution to solve for $x$
  
  - Parallel algorithms based on partitioning of $A$
Sequential Gaussian Elimination

1. procedure GAUSSIAN ELIMINATION (A, b, y)
2. Begin
3.   for k := 0 to n - 1 do /* Outer loop */
4.     begin
5.       for j := k + 1 to n - 1 do
7.       y[k] := b[k] / A[k, k];
8.       A[k, k] := 1;
9.       for i := k + 1 to n - 1 do
10.      begin
11.         for j := k + 1 to n - 1 do
13.         b[i] := b[i] - A[i, k] * y[k];
15.       endfor; /*Line9*/
16.     endfor; /*Line3*/
17.   end GAUSSIAN ELIMINATION
Computation Step in Gaussian Elimination

\[ 5x + 3y = 22 \]
\[ 8x + 2y = 13 \]

\[ x = \frac{(22 - 3y)}{5} \]
\[ \frac{8(22 - 3y)}{5} + 2y = 13 \]
\[ y = \frac{(13 - 176/5)}{(24/5 + 2)} \]
Rowwise Partitioning on Eight Processes

<table>
<thead>
<tr>
<th></th>
<th>P_0</th>
<th>P_1</th>
<th>P_2</th>
<th>P_3</th>
<th>P_4</th>
<th>P_5</th>
<th>P_6</th>
<th>P_7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>(0,1)</td>
<td>(0,2)</td>
<td>(0,3)</td>
<td>(0,4)</td>
<td>(0,5)</td>
<td>(0,6)</td>
<td>(0,7)</td>
</tr>
<tr>
<td>P_0</td>
<td></td>
<td>0</td>
<td>1</td>
<td>(1,2)</td>
<td>(1,3)</td>
<td>(1,4)</td>
<td>(1,5)</td>
<td>(1,6)</td>
</tr>
<tr>
<td>P_1</td>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>(2,3)</td>
<td>(2,4)</td>
<td>(2,5)</td>
<td>(2,6)</td>
</tr>
<tr>
<td>P_2</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>(3,3)</td>
<td>(3,4)</td>
<td>(3,5)</td>
<td>(3,6)</td>
</tr>
<tr>
<td>P_3</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>(4,3)</td>
<td>(4,4)</td>
<td>(4,5)</td>
<td>(4,6)</td>
</tr>
<tr>
<td>P_4</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>(5,3)</td>
<td>(5,4)</td>
<td>(5,5)</td>
<td>(5,6)</td>
</tr>
<tr>
<td>P_5</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>(6,3)</td>
<td>(6,4)</td>
<td>(6,5)</td>
<td>(6,6)</td>
</tr>
<tr>
<td>P_6</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>(7,3)</td>
<td>(7,4)</td>
<td>(7,5)</td>
<td>(7,6)</td>
</tr>
</tbody>
</table>

(a) Computation:

(i) $A[k,j] := A[k,j]/A[k,k]$ for $k < j < n$

(ii) $A[k,k] := 1$

(b) Communication:

One-to-all broadcast of row $A[k,*]$
### Rowwise Partitioning on Eight Processes

<table>
<thead>
<tr>
<th>P_0</th>
<th>1 (0,1) (0,2) (0,3) (0,4) (0,5) (0,6) (0,7)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P_1</td>
<td>0 1 (1,2) (1,3) (1,4) (1,5) (1,6) (1,7)</td>
</tr>
<tr>
<td>P_2</td>
<td>0 0 (2,3) (2,4) (2,5) (2,6) (2,7)</td>
</tr>
<tr>
<td>P_3</td>
<td>0 0 1 (3,4) (3,5) (3,6) (3,7)</td>
</tr>
<tr>
<td>P_4</td>
<td>0 0 (4,3) (4,4) (4,5) (4,6) (4,7)</td>
</tr>
<tr>
<td>P_5</td>
<td>0 0 (5,3) (5,4) (5,5) (5,6) (5,7)</td>
</tr>
<tr>
<td>P_6</td>
<td>0 0 (6,3) (6,4) (6,5) (6,6) (6,7)</td>
</tr>
<tr>
<td>P_7</td>
<td>0 0 (7,3) (7,4) (7,5) (7,6) (7,7)</td>
</tr>
</tbody>
</table>

(c) Computation:

(i) \( A[i,j] := A[i,j] - A[i,k] \times A[k,j] \) for \( k < i < n \) and \( k < j < n \)

(ii) \( A[i,k] := 0 \) for \( k < i < n \)
2D Mesh Partitioning on 64 Processes

(a) Rowwise broadcast of $A[i,k]$ for $(k - 1) < i < n$

(b) $A[k,j] := A[k,j]/A[k,k]$ for $k < j < n$

(c) Columnwise broadcast of $A[k,j]$ for $k < j < n$

Back Substitution to Find Solution

1. procedure BACK SUBSTITUTION (U, x, y)
2. begin
3. for k := n - 1 downto 0 do /* Main loop */
4. begin
5. x[k] := y[k];
6. for i := k - 1 downto 0 do
7. y[i] := y[i] - x[k] xU[i, k];
8. endfor;
9. end BACK SUBSTITUTION
Dense Linear Algebra (www.netlib.gov)

- Basic Linear Algebra Subroutines (BLAS)
  - Level 1 (vector-vector): vectorization
  - Level 2 (matrix-vector): vectorization, parallelization
  - Level 3 (matrix-matrix): parallelization

- LINPACK (Fortran)
  - Linear equations and linear least-squares

- EISPACK (Fortran)
  - Eigenvalues and eigenvectors for matrix classes

- LAPACK (Fortran, C) (LINPACK + EISPACK)
  - Use BLAS internally

- ScaLAPACK (Fortran, C, MPI) (scalable LAPACK)
Outline

- General Design Approach
- Gaussian Elimination
- Graph Algorithms
  - Minimum Spanning Tree
  - Shortest Path
- Map/Reduce
Graph Algorithms

- Graph theory important in computer science
- Many complex problems are graph problems

- $G = (V, E)$
  - $V$ finite set of points called vertices
  - $E$ finite set of edges
  - $e \in E$ is an pair $(u, v)$, where $u, v \in V$
  - Unordered and ordered graphs
Graph Terminology

- Vertex **adjacency** if \((u,v)\) is an edge
- Path from \(u\) to \(v\) if there is an edge sequence starting at \(u\) and ending at \(v\)
- If there exists a path, \(v\) is **reachable** from \(u\)
- A graph is **connected** if all pairs of vertices are connected by a path
- A **weighted graph** associates weights with each edge
- **Adjacency matrix** is an \(n \times n\) array \(A\) such that
  - \(A_{i,j} = 1\) if \((v_i,v_j) \in E\); 0 otherwise
  - Can be modified for weighted graphs (\(\infty\) is no edge)
  - Can represent as adjacency lists
Graph Representations

- Adjacency matrix

- Adjacency list
Minimum Spanning Tree

- A spanning tree of an undirected graph G is a subgraph of G that is a tree containing all the vertices of G.
- The minimum spanning tree (MST) for a weighted undirected graph is a spanning tree with minimum weight.
- Prim’s algorithm can be used
  - Greedy algorithm
  - Selects an arbitrary starting vertex
  - Chooses new vertex guaranteed to be in MST
  - $O(n^2)$
  - Prim’s algorithm is iterative
Prim’s Minimum Spanning Tree Algorithm

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

(a) Original graph

(b) After the first edge has been selected

\[
\begin{array}{ccccccc}
 & a & b & c & d & e & f \\
\hline
a & 0 & 1 & 3 & \infty & \infty & 3 \\
b & 1 & 0 & 5 & 1 & \infty & \infty \\
c & 3 & 5 & 0 & 2 & 1 & \infty \\
d & \infty & 1 & 2 & 0 & 4 & \infty \\
e & \infty & \infty & 1 & 4 & 0 & 5 \\
f & 2 & \infty & \infty & \infty & 5 & 0 \\
\end{array}
\]
Example: Prim’s MST Algorithm

(c) After the second edge has been selected

(d) Final minimum spanning tree

```
<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>∞</td>
<td>∞</td>
<td>3</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>1</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>c</td>
<td>3</td>
<td>5</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>∞</td>
</tr>
<tr>
<td>d</td>
<td>∞</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td>∞</td>
</tr>
<tr>
<td>e</td>
<td>∞</td>
<td>∞</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>f</td>
<td>2</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>5</td>
<td>0</td>
</tr>
</tbody>
</table>
```
Parallel Formulation of Prim’s Algorithm

- Difficult to perform different iterations of the while loop in parallel because $d[v]$ may change each time
- Can parallelize each iteration though
- Partition vertices into $p$ subsets $V_i$, $i=0,...,p-1$
- Each process $P_i$ computes
  - $d_i[u]=\min\{d_i[v] \mid v \in (V-VT) \cap V_i\}$
- Global minimum is obtained using all-to-one reduction
- New vertex is added to $VT$ and broadcast to all processes
- New values of $d[v]$ are computed for local vertex
- $O(n^2/p) + O(n \log p)$ (computation + communication)
Partitioning in Prim’s Algorithm

\[ d[1..n] \]

(a)

\[ A \]

(b)

Processors 0 1 \( i \) \( p-1 \)
Single-Source Shortest Paths

- Find shortest path from a vertex $v$ to all other vertices
- The shortest path in a weighted graph is the edge with the minimum weight
- Weights may represent time, cost, loss, or any other quantity that accumulates additively along a path
- Dijkstra’s algorithm finds shortest paths from vertex $s$
  - Similar to Prim’s MST algorithm
    - MST with vertex $v$ as starting vertex
  - Incrementally finds shortest paths in greedy manner
  - Keep track of minimum cost to reach a vertex from $s$
  - $O(n^2)$
Dijkstra’s Single-Source Shortest Path

1. procedure DIJKSTRA SINGLE SOURCE SP(V, E,w, s)
2. begin
3. VT := {s};
4. for all v ∈ (V - VT ) do
5. if (s, v) exists set l[v] := w(s, v);
6. else set l[v] :=∞;
7. while VT ≠ V do
8. begin
9. find a vertex u such that l[u] := min{l[v]|v ∈ (V - VT )};
10. VT := VT ∪ {u};
11. for all v ∈ (V - VT ) do
12. l[v] := min{l[v], l[u] + w(u, v)};
13. endwhile
14. end DIJKSTRA SINGLE SOURCE SP
Big-Data and Map-Reduce

- Big-data deals with processing large data sets
- Nature of data processing problem makes it amenable to parallelism
  - Looking for features in the data
  - Extracting certain characteristics
  - Analyzing properties with complex data mining algorithms
- Data size makes it opportunistic for partitioning into large # of sub-sets and processing these in parallel
- We need new algorithms, data structures, and programming models to deal with problems
A Simple Big-Data Problem

- Consider a large data collection of text documents
- Suppose we want to find how often a particular word occurs and determine a probability distribution for all word occurrences

Sequential algorithm

1. Data collection
2. Get next document
3. Find and count words
4. Count words and update statistics
5. Generate probability distributions
6. Check if more documents

- web: 2
- weed: 1
- green: 2
- sun: 1
- moon: 1
- land: 1
- part: 1
Parallelization Approach

- **Map**: partition the data collection into subsets of documents and process each subset in parallel
- **Reduce**: assemble the partial frequency tables to derive final probability distribution

**Parallel algorithm**

1. Get next document
2. Find and count words
3. Count words and update statistics
4. Check if more documents
5. Generate probability distributions
Parallelization Approach

- **Map**: partition the data collection into subsets of documents and process each subset in parallel
- **Reduce**: assemble the partial frequency tables to derive final probability distribution

**Parallel algorithm**

1. **Data collection**
2. **Get next document**
3. **Find and count words**
4. **Count words and update statistics**
5. **Generate probability distributions**
6. **Check if more documents**

<table>
<thead>
<tr>
<th>Word</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>web</td>
<td>2</td>
</tr>
<tr>
<td>weed</td>
<td>1</td>
</tr>
<tr>
<td>green</td>
<td>2</td>
</tr>
<tr>
<td>sun</td>
<td>1</td>
</tr>
<tr>
<td>moon</td>
<td>1</td>
</tr>
<tr>
<td>land</td>
<td>1</td>
</tr>
<tr>
<td>part</td>
<td>1</td>
</tr>
</tbody>
</table>
Map-Reduce Parallel Programming

- Become an important distributed parallel programming paradigm for large-scale applications
  - Also applies to shared-memory parallelism
  - Becomes one of the core technologies powering big IT companies, like Google, IBM, Yahoo and Facebook.

- Framework runs on a cluster of machines and automatically partitions jobs into number of small tasks and processes them in parallel

- Can capture in combining Map and Reduce parallel patterns
Map-Reduce Example

- **MAP**: Input data $\rightarrow$ <key, value> pair

Data Collection: split 1

Data Collection: split 2

Data Collection: split n

Split the data to
Supply multiple processors
MapReduce

- **MAP**: Input data ➔ <key, value> pair
- **REDUCE**: <key, value> pair ➔ <result>

Data Collection:
- split1
- split2
- split n

Split the data to supply multiple processors

Map → Reduce

Map → Reduce

Map → Reduce