Parallel Algorithms - Design, Performance & Analysis Issues

Dr. Seema Bawa
Professor & Head, Dept. of Computer Sc. & Engineering
Head, Centre for Information Super Highway
Thapar University, Patiala
Seema@thapar.edu
Parallel Processing

• **Goal:**
  – speedup computationally intensive tasks

• **Method:**
  – multiple processes execute sub-tasks

• **Requirement:**
  – sharing information among the processes
Why Parallel Processing?

Computation requirements are ever increasing—visualization, distributed databases, simulations, scientific prediction (earthquake), etc.

Sequential architectures reaching physical limitations (e.g., speed of light, thermodynamics)
Architecture

• **Single processor:**
  – single instruction stream
  – single data stream
  – von Neumann model

• **Multiple processors:**
  – Flynn’s taxonomy
Flynn’s Taxonomy

- MISD
- MIMD
- SISD
- SIMD

Instruction Streams vs. Data Streams
Single Instruction, Single Data (SISD)

• Typical **Sequential machine**

• Can execute one instruction at a time on one piece of data

• Disregards Pipelining and vector processing
SISD: A Conventional Computer

- Speed is limited by the rate at which computer can transfer information internally.

  Ex: PC, Macintosh, Workstations
More of an intellectual exercise than a practical configuration. Few built, but commercially not available.
SIMD Architecture

Ex: CRAY machine vector processing, Thinking machine cm*
Intel MMX (multimedia support)
Single Instruction, Multiple Data (SIMD)

- **Array Processor**
  - Multiple Memory Units
  - Interconnection Network
  - One control unit that broadcasts instructions to all processors which execute on their own memory

- **Vector Processor**
  - One Memory
  - One Processor
  - Specialized Functional Unit that executes one instruction on pipeline of data
Array Processor

Control Unit

Interconnection Network
Vector Processor

- M
- M
- M
- M

Interconnection Network

- Functional Unit
- P
Unlike SISD, MISD, MIMD computer works asynchronously.

Shared memory (tightly coupled) MIMD

Distributed memory (loosely coupled) MIMD
Multiple Instruction, Multiple Data (MIMD)

- Complete & Independent Processors
- Shared Memory
  - One memory accessible to all processors
  - Instructions can execute on any memory location
- Distributed Memory
  - Memory distributed with processors
  - Processors execute instructions on their data
  - Data passed among processors with messages
Shared Memory MIMD machine

Comm: Source PE writes data to GM & destination retrieves it

- Easy to build, conventional OSes of SISD can be easily be ported
- Limitation: reliability & expandability. A memory component or any processor failure affects the whole system.
- Increase of processors leads to memory contention.

Ex. : Silicon graphics supercomputers....
Distributed Memory MIMD

- Communication: IPC (Inter-Process Communication) via High Speed Network.
- Network can be configured to ... Tree, Mesh, Cube, etc.
- Unlike Shared MIMD
  - easily/ readily expandable
  - Highly reliable (any CPU failure does not affect the whole system)
Parallel vs. Distributed Computing

- **Parallel:**
  - several processing elements concurrently solving a single problem

- **Distributed:**
  - processing elements do not share memory or system clock
Parallelization

• Functional & Data parallelism:
  – **functional**: different operations on same data elements
  – **data**: same operations on different data elements
Principles of Parallel Algorithm Design

• A **sequential algorithm** is essentially a recipe or a sequence of basic steps for solving a given problem.

• A **parallel algorithm** is a recipe that tells us how to solve a given problem using multiple processors.

• A **parallel algorithm** has the added dimension of concurrency and the algorithm designer must specify sets of steps that can be executed in parallel.
Specifying a Parallel Algorithm includes:

- Identifying portions of the work that can be performed concurrently.
  - Mapping the concurrent pieces of work onto multiple processes running in parallel.
- Distributing the input, output, and intermediate data associated with the program.
- Managing accesses to data shared by multiple processors.
Specifying a Parallel Algorithm includes:

- Synchronizing the processors at various stages of the parallel program execution.
Parallel Algorithm Design

Two key steps in the design of parallel algorithms:

- Dividing a computation into smaller computations
- Assigning them to different processors
Decomposition & Tasks

• **Decomposition**: The processes of dividing a computation into smaller parts, some or all of which may potentially be executed in parallel.

• **Tasks**: are programmer-defined units of computation into which the main computation is subdivided by means of decomposition.
Example :- Matrix multiplication

\[
\begin{pmatrix}
A_{1,1} & A_{1,2} \\
A_{2,1} & A_{2,2}
\end{pmatrix}
\cdot
\begin{pmatrix}
B_{1,1} & B_{1,2} \\
B_{2,1} & B_{2,2}
\end{pmatrix}
\rightarrow
\begin{pmatrix}
C_{1,1} & C_{1,2} \\
C_{2,1} & C_{2,2}
\end{pmatrix}
\]

(a)

Task 1: \( C_{1,1} = A_{1,1}B_{1,1} + A_{1,2}B_{2,1} \)
Task 2: \( C_{1,2} = A_{1,1}B_{1,2} + A_{1,2}B_{2,2} \)
Task 3: \( C_{2,1} = A_{2,1}B_{1,1} + A_{2,2}B_{2,1} \)
Task 4: \( C_{2,2} = A_{2,1}B_{1,2} + A_{2,2}B_{2,2} \)

(b)
Decomposition & Tasks (Contd..)

<table>
<thead>
<tr>
<th>Decomposition I</th>
<th>Decomposition II</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task 1: $C_{1,1} = A_{1,1}B_{1,1}$</td>
<td>Task 1: $C_{1,1} = A_{1,1}B_{1,1}$</td>
</tr>
<tr>
<td>Task 2: $C_{1,1} = C_{1,1} + A_{1,2}B_{2,1}$</td>
<td>Task 2: $C_{1,1} = C_{1,1} + A_{1,2}B_{2,1}$</td>
</tr>
<tr>
<td>Task 3: $C_{1,2} = A_{1,1}B_{1,2}$</td>
<td>Task 3: $C_{1,2} = A_{1,2}B_{2,2}$</td>
</tr>
<tr>
<td>Task 4: $C_{1,2} = C_{1,2} + A_{1,2}B_{2,2}$</td>
<td>Task 4: $C_{1,2} = C_{1,2} + A_{1,1}B_{1,2}$</td>
</tr>
<tr>
<td>Task 5: $C_{2,1} = A_{2,1}B_{1,1}$</td>
<td>Task 5: $C_{2,1} = A_{2,2}B_{2,1}$</td>
</tr>
<tr>
<td>Task 6: $C_{2,1} = C_{2,1} + A_{2,2}B_{2,1}$</td>
<td>Task 6: $C_{2,1} = C_{2,1} + A_{2,1}B_{1,1}$</td>
</tr>
<tr>
<td>Task 7: $C_{2,2} = A_{2,1}B_{1,2}$</td>
<td>Task 7: $C_{2,2} = A_{2,1}B_{1,2}$</td>
</tr>
<tr>
<td>Task 8: $C_{2,2} = C_{2,2} + A_{2,2}B_{2,2}$</td>
<td>Task 8: $C_{2,2} = C_{2,2} + A_{2,2}B_{2,2}$</td>
</tr>
</tbody>
</table>
Task Dependency Graphs
Example: Matrix Multiplication
Example: Database Query Processing

QUERY

MODEL="Accord" AND YEAR="1999"
(COLOR="Green" OR COLOR="Black")
Example: Database Query Processing

```
<table>
<thead>
<tr>
<th>ID#</th>
<th>Model</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>4523</td>
<td>Accord</td>
<td></td>
</tr>
<tr>
<td>6734</td>
<td>Accord</td>
<td></td>
</tr>
<tr>
<td>4395</td>
<td>Accord</td>
<td></td>
</tr>
<tr>
<td>7352</td>
<td>Accord</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID#</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>7623</td>
<td>1999</td>
</tr>
<tr>
<td>6734</td>
<td>1999</td>
</tr>
<tr>
<td>5342</td>
<td>1999</td>
</tr>
<tr>
<td>3845</td>
<td>1999</td>
</tr>
<tr>
<td>4395</td>
<td>1999</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID#</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>7623</td>
<td>Black</td>
</tr>
<tr>
<td>9834</td>
<td>Black</td>
</tr>
<tr>
<td>5342</td>
<td>Black</td>
</tr>
<tr>
<td>8354</td>
<td>Black</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID#</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>3476</td>
<td>Green</td>
</tr>
<tr>
<td>6734</td>
<td>Green</td>
</tr>
<tr>
<td>8354</td>
<td>Black</td>
</tr>
</tbody>
</table>

Accord

1999

Green

Black

Accord AND 1999

Green OR Black

Accord AND 1999 AND (Green OR Black)

<table>
<thead>
<tr>
<th>ID#</th>
<th>Model</th>
<th>Year</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>6734</td>
<td>Accord</td>
<td>1999</td>
<td>Green</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID#</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>3476</td>
<td>Green</td>
</tr>
<tr>
<td>7623</td>
<td>Black</td>
</tr>
<tr>
<td>9834</td>
<td>Black</td>
</tr>
<tr>
<td>6734</td>
<td>Green</td>
</tr>
<tr>
<td>5342</td>
<td>Black</td>
</tr>
<tr>
<td>8354</td>
<td>Black</td>
</tr>
</tbody>
</table>
```
Granularity & Concurrency

• The number and size of tasks into which a problem is decomposed determines the **granularity** of the decomposition.

  **TYPES:**
  
  o **FINE GRAINED**
  o **COARSE GRAINED**
Processes and Mapping

- **Process** is an abstract entity that uses the code and data corresponding to a task to produce the output of that task within a finite amount of time after the task is activated by the parallel program.

- The mechanism by which tasks are assigned to processes for execution is called *mapping*.
Decomposition Techniques

- Recursive decomposition
- Data decomposition
- Exploratory decomposition
- Speculative decomposition
Characteristics of Tasks

- Task Generation
- Task Sizes
- Size of Data Associated with Tasks
Characteristics of Inter-task Interactions

- STATIC
- DYNAMIC
Mapping Techniques for Load Balancing

- STATIC
- DYNAMIC
Performance Metrics for Parallel Systems

- **Execution Time**
  The parallel run time is the time that elapses from the moment that a parallel computation starts to the moment that the last processor finishes execution.

\[ T_p = \frac{T_s}{p}. \]
Total Parallel Overhead

**Total overhead** or *overhead function* of a parallel system as the total time collectively spent by all the processors over and above that required by the fastest known sequential algorithm for solving the same problem on a single processor.

\[ T_o = pT_p - W \]
Speedup

It is defined as the ratio of the time taken to solve a problem on a single processor to the time required to solve the same problem on a parallel computer with $p$ identical processors.

$$S_p = \frac{T_s}{T_p} \quad (T_s \leq T_1)$$
Efficiency

It is defined as the ratio of speedup to the number of processors. In an ideal parallel system, speedup is equal to $p$ and efficiency is equal to one.

\[
E = \frac{S}{p}.
\]
Efficiency

• **A parallel algorithm is efficient iff**
  – it is fast (e.g. polynomial time) and
  – the product of the parallel time and number of processors is close to the time of at the best known sequential algorithm

\[
T_{\text{sequential}} \approx T_{\text{parallel}} \cdot N \text{ processors}
\]

• **A parallel algorithms** is optimal iff this product is of the same order as the best known sequential time
The **COST** of solving a problem on a parallel system as the product of parallel run time and the number of processors used.

A parallel system is said to be *cost-optimal* if the cost of solving a problem on a parallel computer is proportional to the execution time of the fastest-known sequential algorithm on a single processor.
Example - Adding on a Hypercube (cont.)

\[ T_p = \Theta(\log n) \quad S = \Theta\left(\frac{n}{\log n}\right) \quad E = \frac{S}{n} = \Theta\left(\frac{1}{\log n}\right) \]
EXAMPLE ILLUSTRATING THE PERFORMANCE METRICS

4 processors simulating 16

\[ T_p = \Theta\left(\frac{n}{p} + \left(\frac{n}{p}\right) \log p\right) = \Theta\left(\left(\frac{n}{p}\right) \log p\right) \]

\[ \Rightarrow p^* T_p = \Theta(n \log p) > \Theta(n) \]
A cost optimal alternative

Figure 4.4 A cost-optimal way of computing the sum of 16 numbers on a four-processor hypercube.

Copyright © 1994 Benjamin/Cummings Publishing
Scalability of Parallel Systems

The **scalability** of a parallel system is a measure of its capacity to increase speedup in proportion to the number of processors. It reflects a parallel system’s ability to utilize increasing processing resources effectively.
Parallel Algorithm Analysis

Computation complexity $T_{comp}(n,p)$

Communication complexity $T_{comm}(n,p)$
ALGORITHM ANALYSIS
TECNIQUES

The essence of parallel programming lies in being able to partition the task gracefully.

DIVIDE AND CONQUER
Steps
1. Divide an instance of a problem into one or more smaller instances.
2. Conquer each of the smaller instances. Unless a smaller instance is sufficiently small, use recursion to do this.
3. If necessary, combine the solutions to the smaller instances to obtain the solution to the original instance.
DYNAMIC PROGRAMMING:

STEPS:-
1. Establish a *recursive property* that gives the Solution to an instance of the problem.

2. Solve an instance of the problem in a *bottom-up* fashion by solving smaller instances first.
The binomial coefficient is given by:

\[ \binom{n}{k} = \left\{ \begin{array}{ll} \binom{n-1}{k-1} + \binom{n-1}{k} & 0 < k < n \end{array} \right. \]
Algorithm ---- Binomial Coefficient using Divide and Conquer

• function bin(n, k : integer):integer;
• begin
•    if k=0 or n=k then
•        bin:=1
•    else
•        bin:=bin(n-1,k-1)+bin(n-1,k)
•    end
• end;
function bin2(n,k :integer):integer;
  var
  i,j:index;
  B:array[0..n,0..k] of integer;
  Begin
  For i:= 0 to n do
    For j:=0 to minimum(i,k) do
      If j=0 or j=1 then
        B[i,j]:=i
      Else
        B[i,j]:= b[i-1,j-1] + b[i-1,j]
  End
  End;
THE GREEDY APPROACH

STEPS:

A selection procedure chooses the next item to add to the set. The selection is performed according to the greedy criterion that satisfies some locally optimal consideration at the time.

A feasibility check determines if the new set is feasible by checking whether it is possible to complete this set in such a way as to give a solution to the instance.
THE GREEDY APPROACH

- A solution check determines whether the new set constitutes a solution to the instance.
- A greedy algorithm works in phases: At each phase:
  - You take the best you can get right now, without regard for future consequences
  - You hope that by choosing a local optimum at each step, you will end up at a global optimum
BACKTRACKING

• Backtracking is used to solve problems in which a sequence of objects is chosen from a specified set so that the sequence satisfies some criteria.

• EXAMPLE: n-Queens Problem
BACKTRACKING

STEPS:-
– Tests to see if a solution has been found, and if so, returns it; otherwise
– For each choice that can be made at this point,
  • Make that choice
  • Recur
  • If the recursion returns a solution, return it
– If no choices remain, return failure