Principles of Parallel Algorithm Design (2)

Jinkyu Jeong (jinkyu@skku.edu)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu
Topics

- **Introduction to Parallel Algorithms**
  - Tasks and Decomposition
  - Processes and Mapping
  - Processes vs. Processors

- **Decomposition Techniques**
  - Recursive Decomposition
  - Data Decomposition
  - Exploratory Decomposition
  - Hybrid Decomposition

- **Characteristics of Tasks and Interactions**
  - Task Generation, Granularity, and Context
  - Characteristics of Task Interactions.
Topics – con’t

- **Mapping Techniques for Load Balancing**
  - Static and Dynamic Mapping

- **Methods for Minimizing Interaction Overheads**
  - Maximizing Data Locality
  - Minimizing Contention and Hot-Spots
  - Overlapping Communication and Computations
  - Replication vs. Communication
  - Group Communications vs. Point-to-Point Communication

- **Parallel Algorithm Design Models**
  - Data-Parallel, Work-Pool, Task Graph, Master-Slave, Pipeline, and Hybrid Models
Characteristics of Tasks

- **Tasks**
  - Pieces of work decomposed

- **Key characteristics**
  - Task generation
  - Task sizes
  - Size of data associated with tasks

- **Critically impact choice and performance of parallel algorithms**
Task Generation

- **Static task generation**
  - Tasks can be identified a-priori
  - Typically decompose using data or recursive decomposition techniques
    - Matrix operations
    - Graph algorithms
    - Image processing applications
    - Other regularly structured problems

- **Dynamic task generation**
  - Tasks are generated as computation performed
  - Typically decompose using exploratory or speculative decompositions
    - Game playing
    - 15 puzzle
Task Sizes

- **Uniform**
  - Example: matrix-vector multiplication

- **Non-uniform**
  - Task sizes can be determined a-priori
  - Or not
  - Examples
    - In quicksort, size of each partition depends on pivot selected
    - In 15-puzzle, sizes of tasks are unknown
Data associated with a task may be small or large compared to computation

- Example
  - 15 puzzle: size(input) < size(computation)
  - Min: size(input) = size(computation) > size(output)
  - Qsort: size(input) = size(output) < size(computation)

Implications

- Small data: task can migrate to other processes easily and dynamically
- Large data: ties the task to a process
  - Avoids excessive communication of task contexts
Characteristics of Task Interactions

- **Different dimensions of task interactions**
  - Static vs. dynamic
  - Regular vs. irregular
  - Read-only vs. read-write
  - One-sided vs. two-sided
Characteristics of Task Interactions

- **Static interactions**
  - The tasks and their interactions are known a-priori
  - Simpler to code

- **Dynamic interactions**
  - Timing or interacting tasks cannot be determined a-priori
  - Harder to code
    - Especially, using message passing APIs.
Characteristics of Task Interactions

- **Regular interactions**
  - Having a definite pattern (in the graph sense) to the interactions
  - Regular interactions can be exploited for efficient implementation
    - Schedule tasks to avoid conflicts

- **Irregular interactions**
  - Lack well-defined topologies
  - Modeled by graphs
Static Regular Interaction Pattern

- **Example: image dithering**
  - Pixel values on edges should be passed to adjacent tasks
    - Location of data to be sent to other tasks is deterministic regardless of input

```
Static Regular Interaction Pattern

- **Example: image dithering**
  - Pixel values on edges should be passed to adjacent tasks
    - Location of data to be sent to other tasks is deterministic regardless of input

  ![Diagram](image)
```

SSE3054: Multicore Systems | Spring 2016 | Jinkyu Jeong (jinkyu@skku.edu)
Static Irregular Interaction Pattern

- Example: sparse matrix-vector multiplication
  - Interaction pattern varies depending on the input matrix

(a) 

(b) 

\[ \begin{array}{ccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\
\end{array} \]

\[ \begin{array}{c}
\text{Task 0} \\
\text{Task 11} \\
\end{array} \]

\[ \begin{array}{ccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\
\end{array} \]

\[ \begin{array}{c}
\text{Task 0} \\
\text{Task 11} \\
\end{array} \]
Characteristics of Task Interactions

- **Read-only interactions**
  - Tasks just read data associated with other tasks

- **Read-write interactions**
  - Tasks read, as well as modify data associated with other tasks
  - Harder to code
    - Requiring additional synchronization primitives
Characteristics of Task Interactions

- **One-way interaction**
  - Initiated and accomplished by one of the two interacting tasks
    - Read or write
    - Get or put
  - Somewhat harder to code in message passing APIs

- **Two-way interaction**
  - Requiring participation from both tasks involved in an interaction
    - Send and recv in message passing APIs
Mapping Techniques

- Decomposed concurrent tasks should be mapped to processes
  - Then, those can be executed on a parallel platform

- Mapping overheads
  - Communication
  - Idling (or serialization)

- Minimizing these overheads often represents contradicting objectives.
  - Assigning all work to one processor
    - No communication but significant idling
  - Minimizing serialization
    - Introducing communications trivially minimizes communication at the expense of significant idling
Mapping Techniques for Minimum Idleing

- Mapping must simultaneously minimize idling and load balance
- Merely balancing load does not minimize idling
  - Two cases are balanced in loads, but (b) shows much idling
Mapping Techniques for Minimum Idling

- **Static Mapping**
  - Tasks are mapped to processes a-priori
  - Requirements
    - A good estimate of the size of each task
    - Even in these cases, optimal mapping may be NP complete
      » E.g., multiple knapsack problem

- **Dynamic Mapping**
  - Tasks are mapped to processes at runtime
    - Tasks are generated at runtime
    - Their sizes are not known

- **Other factors influencing choice of mapping**
  - Size of data associated with a task
  - Nature of underlying domain
Schemes for Static Mapping

- Mappings based on data partitioning
  - Block distribution
  - (Block-)cyclic distribution
  - Randomized block distribution
  - Graph partitioning

- Mappings based on task partitioning

- Hybrid (or hierarchical) mappings
## Mappings Based on Data Partitioning

- Data partitioning + owner-computes rule
- Example: 1-D block distribution for dense matrices

### row-wise distribution

<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>
</table>

### column-wise distribution

|   | $P_0$ | $P_1$ | $P_2$ | $P_3$ | $P_4$ | $P_5$ | $P_6$ | $P_7$ |
## Block Array Distribution Schemes

- **Multi-dimensional block distributions**
  - More tasks and data partitions $\rightarrow$ higher degree of concurrency can be possible

<table>
<thead>
<tr>
<th>$P_0$</th>
<th>$P_1$</th>
<th>$P_2$</th>
<th>$P_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_4$</td>
<td>$P_5$</td>
<td>$P_6$</td>
<td>$P_7$</td>
</tr>
<tr>
<td>$P_8$</td>
<td>$P_9$</td>
<td>$P_{10}$</td>
<td>$P_{11}$</td>
</tr>
<tr>
<td>$P_{12}$</td>
<td>$P_{13}$</td>
<td>$P_{14}$</td>
<td>$P_{15}$</td>
</tr>
</tbody>
</table>
Block Array Distribution Schemes: Examples

- Multiplying two dense matrices $C = A \times B$
  - Partition the output matrix $C$ using a block decomposition.

- Give each task the same number of elements of $C$
  - Each element of $C$ corresponds to a single dot product

- Precise decomposition (1-D or 2-D) is determined by the associated communication overhead

- In general, higher dimension decomposition allows the use of larger number of processes
Data Sharing in Dense Matrix Multiplication

\[
\begin{align*}
\text{Required memory} & = \frac{2n^2}{p} + n^2 \\
& = \frac{2n}{\sqrt{p}} + \frac{n^2}{p}
\end{align*}
\]
Cyclic and Block Cyclic Distributions

- If the amount of computation associated with data items varies
  - A block decomposition may lead to significant load imbalances

- A simple example of this is in LU decomposition (or Gaussian Elimination) of dense matrices
LU Factorization of a Dense Matrix

- A decomposition of LU factorization into 14 tasks

\[
\begin{pmatrix}
A_{1,1} & A_{1,2} & A_{1,3} \\
A_{2,1} & A_{2,2} & A_{2,3} \\
A_{3,1} & A_{3,2} & A_{3,3}
\end{pmatrix}
\rightarrow
\begin{pmatrix}
L_{1,1} & 0 & 0 \\
L_{2,1} & L_{2,2} & 0 \\
L_{3,1} & L_{3,2} & L_{3,3}
\end{pmatrix}
\cdot
\begin{pmatrix}
U_{1,1} & U_{1,2} & U_{1,3} \\
0 & U_{2,2} & U_{2,3} \\
0 & 0 & U_{3,3}
\end{pmatrix}
\]

1: \( A_{1,1} \rightarrow L_{1,1}U_{1,1} \)
2: \( L_{2,1} = A_{2,1}U_{1,1}^{-1} \)
3: \( L_{3,1} = A_{3,1}U_{1,1}^{-1} \)
4: \( U_{1,2} = L_{1,1}^{-1}A_{1,2} \)
5: \( U_{1,3} = L_{1,1}^{-1}A_{1,3} \)
6: \( A_{2,2} = A_{2,2} - L_{2,1}U_{1,2} \)
7: \( A_{3,2} = A_{3,2} - L_{3,1}U_{1,2} \)
8: \( A_{2,3} = A_{2,3} - L_{2,1}U_{1,3} \)
9: \( A_{3,3} = A_{3,3} - L_{3,1}U_{1,3} \)
10: \( A_{2,2} \rightarrow L_{2,2}U_{2,2} \)
11: \( L_{3,2} = A_{3,2}U_{2,2}^{-1} \)
12: \( U_{2,3} = L_{2,2}^{-1}A_{2,3} \)
13: \( A_{3,3} = A_{3,3} - L_{3,2}U_{2,3} \)
14: \( A_{3,3} \rightarrow L_{3,3}U_{3,3} \)
LU Factorization of a Dense Matrix

- Block distribution causes load imbalance (or idling)

<table>
<thead>
<tr>
<th>P₀</th>
<th>P₃</th>
<th>P₆</th>
</tr>
</thead>
<tbody>
<tr>
<td>T₁</td>
<td>T₄</td>
<td>T₅</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P₁</th>
<th>P₄</th>
<th>P₇</th>
</tr>
</thead>
<tbody>
<tr>
<td>T₂</td>
<td>T₆</td>
<td>T₈</td>
</tr>
<tr>
<td></td>
<td>T₁₀</td>
<td>T₁₂</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P₂</th>
<th>P₅</th>
<th>P₈</th>
</tr>
</thead>
<tbody>
<tr>
<td>T₃</td>
<td>T₇</td>
<td>T₉</td>
</tr>
<tr>
<td></td>
<td>T₁₁</td>
<td>T₁₃</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T₁₄</td>
</tr>
</tbody>
</table>
Block-Cyclic Distributions

• **Variation of the block distribution**
  • Alleviate the load-imbalance and idling problems.

• **Steps**
  • Partition an array into many more blocks than the number of available processes
  • Assign blocks to processes in a round-robin manner
    – Each process gets several non-adjacent blocks
Block-Cyclic Distributions

- A cyclic distribution is a special case in which block size is one.
- A block distribution is a special case in which block size is $n/p$.
  - $n$ is the dimension of the matrix and $p$ is the number of processes.
Randomized Block Distribution

- Sometimes, block-cyclic distribution causes load imbalance
  - Example: sparse matrix-vector multiplication
    - Diagonal processes are overloaded

![Sparse matrix](image1.png)

<table>
<thead>
<tr>
<th></th>
<th>P_0</th>
<th>P_1</th>
<th>P_2</th>
<th>P_3</th>
<th>P_0</th>
<th>P_1</th>
<th>P_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>P_4</td>
<td>P_5</td>
<td>P_6</td>
<td>P_7</td>
<td>P_4</td>
<td>P_5</td>
<td>P_6</td>
<td>P_7</td>
</tr>
<tr>
<td>P_8</td>
<td>P_9</td>
<td>P_{10}</td>
<td>P_{11}</td>
<td>P_8</td>
<td>P_9</td>
<td>P_{10}</td>
<td>P_{11}</td>
</tr>
<tr>
<td>P_{12}</td>
<td>P_{13}</td>
<td>P_{14}</td>
<td>P_{15}</td>
<td>P_{12}</td>
<td>P_{13}</td>
<td>P_{14}</td>
<td>P_{15}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>P_0</th>
<th>P_1</th>
<th>P_2</th>
<th>P_3</th>
<th>P_0</th>
<th>P_1</th>
<th>P_2</th>
<th>P_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>P_4</td>
<td>P_5</td>
<td>P_6</td>
<td>P_7</td>
<td>P_4</td>
<td>P_5</td>
<td>P_6</td>
<td>P_7</td>
<td></td>
</tr>
<tr>
<td>P_8</td>
<td>P_9</td>
<td>P_{10}</td>
<td>P_{11}</td>
<td>P_8</td>
<td>P_9</td>
<td>P_{10}</td>
<td>P_{11}</td>
<td></td>
</tr>
<tr>
<td>P_{12}</td>
<td>P_{13}</td>
<td>P_{14}</td>
<td>P_{15}</td>
<td>P_{12}</td>
<td>P_{13}</td>
<td>P_{14}</td>
<td>P_{15}</td>
<td></td>
</tr>
</tbody>
</table>

Block-cyclic distribution
Randomized Block Distribution

- **Solution**
  - Randomize tasks
  - Then, block-cyclic or block distribution

Sparse matrix  
Randomized  
2D block distribution
Decomposition by Graph Partitioning

- **In case of sparse data structures, block decompositions are more complex**
  - Ex) physical phenomena simulation
  - Interaction is data dependent and irregular
  - Data is represented using graph

- **(Task interaction) graph of data structures is a useful indicator**
  - Work is number of nodes
  - Communication is the degree of each node

- **Partition the graph**
  - Assign equal number of nodes to each process
    - Balance work
  - Minimize edge count of the graph partition
    - Minimize communication
Partitioning the Graph of Lake Superior

Random Partitioning

Partitioning for minimum edge-cut.
Mappings Based on Task Partitioning

- Partitioning a given task-dependency graph
  - When task-dependency graph is static
  - When task sizes are known
- Optimal mapping for a general task-dependency graph
  - NP-complete problem
- Excellent heuristics exist for structured graphs
Mapping a Sparse Graph

- Task partitioning and mapping based on 1D block distribution

17 communications

C0 = (4,5,6,7,8)

C1 = (0,1,2,3,8,9,10,11)

C2 = (0,4,5,6)
Mapping a Sparse Graph

- Task partitioning and mapping based on the task interaction graph

13 communications

C0 = (1, 2, 6, 9)
C1 = (0, 5, 6)
C2 = (1, 2, 4, 5, 7, 8)
Hierarchical Mappings

- Sometimes a single mapping technique is inadequate
  - For example, the task mapping of the binary tree (quicksort) cannot use a large number of processors

- Hierarchical mappings
  - Use a task mapping at the top level
  - Data partitioning within each task

[Diagram showing basic and hierarchical task mappings]
Schemes for Dynamic Mapping

- **Dynamic mapping (or dynamic load balancing)**
  - Load balancing is the primary motivation for dynamic mapping.

- **Styles**
  - Centralized vs. distributed
Centralized Dynamic Mapping

- **Processes = masters or slaves**
- **General strategy**
  - When a slave runs out of work, it requests the master for more work
- **Challenge**
  - Master may become bottleneck for large number of processes
- **Approach**
  - Chunk scheduling
    - a process picks up a number of tasks (a chunk) at one time
    - Selecting large chunk sizes may lead to significant load imbalances as well
    - gradually decrease chunk size as the computation progresses
Distributed Dynamic Mapping

- Each process can send or receive work from other processes.
  - Avoids bottleneck in centralized schemes

- Four critical questions
  - How are sending and receiving processes paired together?
  - Who initiates work transfer?
  - How much work is transferred?
  - When is a transfer triggered?

- Answers are generally application-specific
Topics – con’t

- Mapping Techniques for Load Balancing
  - Static and Dynamic Mapping
- Methods for Minimizing Interaction Overheads
- Parallel Algorithm Design Models
Minimizing Interaction Overheads (1)

- **Maximize data locality**
  - Where possible, reuse intermediate data
  - Restructure computation to reuse data promptly

- **Minimize volume of data exchange**
  - Minimize the volume of data communicated by carefully partitioning task interaction graph

- **Minimize frequency of interactions**
  - Try to merge multiple interactions to one, where possible

- **Minimize contention and hot-spots**
  - Use decentralized techniques
  - Replicate data where necessary
Example: Minimizing Interaction Overheads (1)

- Minimize contention and hot-spots
  - Example: dense matrix multiplication
    - Matrices are split into 16 pieces \((A_{i,j}, B_{i,j}, C_{i,j})\)
    - 2D block distribution
      » P0, P1, P2 and P3 will contend on \(A_{0,*}\) at the same time

- Solution: rotate the access sequence of \(A_{0,*}\) in each process
  » P0 begins on \(A_{0,0}\) while P1 begins on \(A_{0,1}, \ldots\)
Minimizing Interaction Overheads (2)

- Overlapping computations with interactions
  - Non-blocking communications
    - E.g., non-blocking send in MPI
  - Multithreading
  - Prefetching

- Replicating data or computations

- Using group communications instead of point-to-point primitives

- Overlap interactions with other interactions
Minimizing Interaction Overheads (3)

- Overlap interactions with other interactions

  • Example: broadcast interaction

  ![Diagram showing efficient and naive broadcasts](image)

  (a) Efficient broadcast
  (b) A naïve broadcast
  (c) Overlapping four naïve broadcasts
Topics – con’t

- Mapping Techniques for Load Balancing
  - Static and Dynamic Mapping
- Methods for Minimizing Interaction Overheads
- Parallel Algorithm Design Models
Parallel Algorithm Models (1)

- A way of structuring a parallel algorithm
  - Decomposition
  - Mapping technique
  - Applying strategies to minimize interactions
Parallel Algorithm Models (2)

- **Data parallel model**
  - Tasks are statically (or semi-statically) mapped to processes
  - Each task performs similar operations on different data

- **Task graph model**
  - Use a task dependency graph and its interrelationships
    - to promote locality or to reduce interaction costs

- **Work pool model**
  - Dynamic mapping of tasks onto processes
  - Centralized/decentralized pool of tasks
  - Static or dynamic tasks
Parallel Algorithm Models (3)

- **Master-slave model**
  - One or more processes generate work
  - Allocate it to worker processes
  - Allocation may be static or dynamic.

- **Pipeline / producer-consumer model**
  - Pass a stream of data through a succession of processes
  - Each performs some task on it

- **Hybrid model**
  - Multiple models applied hierarchically
  - Or, multiple models applied sequentially to different phases
References

- “COMP422: Parallel Computing” by Prof. John Mellor-Crummey at Rice Univ.