Lecture 10: Principles of Parallel Algorithm Design

CSCE 569 Parallel Computing

Department of Computer Science and Engineering
Yonghong Yan
yanyh@cse.sc.edu
http://cse.sc.edu/~yanyh
Last two lectures: Algorithms and Concurrency

- Introduction to Parallel Algorithms
  - Tasks and decomposition
  - Processes and mapping
- Decomposition Techniques
  - Recursive decomposition (divide-conquer)
  - Data decomposition (input, output, input+output, intermediate)
- Terms and concepts
  - Task dependency graph, task granularity, degree of concurrency
  - Task interaction graph, critical path
- Examples:
  - Dense vector addition, matrix vector and matrix matrix product
  - Database query, quicksort, MIN
  - Image convolution (filtering) and Jacobi
Today’s lecture

Decomposition Techniques - continued
  – Exploratory Decomposition
  – Hybrid Decomposition

Mapping tasks to processes/cores/CPU/PEs

• Characteristics of Tasks and Interactions
  – Task Generation, Granularity, and Context
  – Characteristics of Task Interactions

• Mapping Techniques for Load Balancing
  – Static and Dynamic Mapping

• Methods for Minimizing Interaction Overheads

• Parallel Algorithm Design Models
Exploratory Decomposition

• Decomposition is fixed/static from the design
  – Data and recursive

• Exploration (search) of a state space of solutions
  – Problem decomposition reflects shape of execution
  – Goes hand-in-hand with its execution

• Examples
  – discrete optimization, e.g. 0/1 integer programming
  – theorem proving
  – game playing
Exploratory Decomposition: Example

Solve a 15 puzzle

- Sequence of three moves from state (a) to final state (d)

(a)  

(b)  

(c)  

(d)  

- From an arbitrary state, must search for a solution
Exploratory Decomposition: Example

Solving a 15 puzzle

• Search
  – generate successor states of the current state
  – explore each as an independent task
Exploratory Decomposition Speedup

Solve a 15 puzzle

• The decomposition behaves according to the parallel formulation
  – May change the amount of work done

![Diagram of 15 puzzle decomposition](image)

Total serial work: 2m+1
Total parallel work: 1

(a)

Total serial work: m
Total parallel work: 4m

(b)

Execution terminate when a solution is found
Speculative Decomposition

• Dependencies between tasks are not known a-priori.
  – Impossible to identify independent tasks
• Two approaches
  – Conservative approaches, which identify independent tasks only when they are guaranteed to not have dependencies
    • May yield little concurrency
  – Optimistic approaches, which schedule tasks even when they may potentially be inter-dependent
    • Roll-back changes in case of an error
Speculative Decomposition: Example

Discrete event simulation

• Centralized time-ordered event list
  – you get up ➔ get ready ➔ drive to work ➔ work ➔ eat lunch ➔ work some more ➔ drive back ➔ eat dinner ➔ and sleep

• Simulation
  – extract next event in time order
  – process the event
  – if required, insert new events into the event list

• Optimistic event scheduling
  – assume outcomes of all prior events
  – speculatively process next event
  – if assumption is incorrect, roll back its effects and continue
Speculative Decomposition: Example

Simulation of a network of nodes

• Simulate network behavior for various input and node delays
  – The input are dynamically changing
  • Thus task dependency is unknown

• Speculate execution: tasks’ input
  – Correct: parallelism
  – Incorrect: rollback and redo
Speculative vs Exploratory

• Exploratory decomposition
  – The output of multiple tasks from a branch is unknown
  – Parallel program perform more, less or same amount of work as serial program

• Speculative
  – The input at a branch leading to multiple parallel tasks is unknown
  – Parallel program perform more or same amount of work as the serial algorithm
Hybrid Decompositions

Use multiple decomposition techniques together

• One decomposition may be not optimal for concurrency
  – Quicksort recursive decomposition limits concurrency (Why?)

• Combined recursive and data decomposition for MIN
Today’s lecture

• Decomposition Techniques - continued
  – Exploratory Decomposition
  – Hybrid Decomposition

**Mapping tasks to processes/cores/CPU/PEs**

- Characteristics of Tasks and Interactions
  – Task Generation, Granularity, and Context
  – Characteristics of Task Interactions

• Mapping Techniques for Load Balancing
  – Static and Dynamic Mapping

• Methods for Minimizing Interaction Overheads

• Parallel Algorithm Design Models
Characteristics of Tasks

• Theory
  – Decomposition: to parallelize theoretically
    • Concurrency available in a problem

• Practice
  – Task creations, interactions and mapping to PEs.
    • Realizing concurrency practically
  – Characteristics of tasks and task interactions
    • Impact choice and performance of parallelism

• Characteristics of tasks
  – Task generation strategies
  – Task sizes (the amount of work, e.g. FLOPs)
  – Size of data associated with tasks
Task Generation

• Static task generation
  – Concurrent tasks and task graph known a-priori (before execution)
  – Typically using recursive or data decomposition
  – Examples
    • Matrix operations
    • Graph algorithms
    • Image processing applications
    • Other regularly structured problems

• Dynamic task generation
  – Computations formulate concurrent tasks and task graph on the fly
    • Not explicit a priori, though high-level rules or guidelines known
  – Typically by exploratory or speculative decompositions.
    • Also possible by recursive decomposition, e.g. quicksort
  – A classic example: game playing
    • 15 puzzle board
Task Sizes/Granularity

• The amount of work \( \rightarrow \) amount of time to complete
  – E.g. FLOPs, #memory access

• Uniform:
  – Often by \textit{even} data decomposition, i.e. regular

• Non-uniform
  – Quicksort, the choice of pivot
Size of Data Associated with Tasks

• May be small or large compared to the task sizes
  – How relevant to the input and/or output data sizes
  – Example:
    • size(input) < size(computation), e.g., 15 puzzle
    • size(input) = size(computation) > size(output), e.g., min
    • size(input) = size(output) < size(computation), e.g., sort

• Considering the efforts to reconstruct the same task context
  – small data: small efforts: task can easily migrate to another process
  – large data: large efforts: ties the task to a process

• Context reconstructing vs communicating
  – It depends
Characteristics of Task Interactions

• Aspects of interactions
  – What: shared data or synchronizations, and sizes of the media
  – When: the timing
  – Who: with which task(s), and overall topology/patterns
  – Do we know details of the above three before execution
  – How: involve one or both?
    • The implementation concern, implicit or explicit

Orthogonal classification
• Static vs. dynamic
• Regular vs. irregular
• Read-only vs. read-write
• One-sided vs. two-sided
Characteristics of Task Interactions

• Aspects of interactions
  – What: shared data or synchronizations, and sizes of the media
  – When: the timing
  – Who: with which task(s), and overall topology/patterns
  – Do we know details of the above three before execution
  – How: involve one or both?

• Static interactions
  – Partners and timing (and else) are known a-priori
  – Relatively simpler to code into programs.

• Dynamic interactions
  – The timing or interacting tasks cannot be determined a-priori.
  – Harder to code, especially using explicit interaction.
Characteristics of Task Interactions

• Aspects of interactions
  – What: shared data or synchronizations, and sizes of the media
  – When: the timing
  – Who: with which task(s), and overall topology/patterns
  – Do we know details of the above three before execution
  – How: involve one or both?

• Regular interactions
  – Definite pattern of the interactions
    • E.g. a mesh or ring
    – Can be exploited for efficient implementation.

• Irregular interactions
  – Lack well-defined topologies
  – Modeled as a graph
Example of *Regular* Static Interaction

Image processing algorithms: dithering, edge detection

- Nearest neighbor interactions on a 2D mesh

\[
\begin{align*}
G_x &= \begin{bmatrix}
-1 & 0 & +1 \\
-2 & 0 & +2 \\
-1 & 0 & +1 \\
\end{bmatrix} \\
G_y &= \begin{bmatrix}
-1 & -2 & -1 \\
0 & 0 & 0 \\
+1 & +2 & +1 \\
\end{bmatrix}
\end{align*}
\]
Example of *Irregular* Static Interaction

Sparse matrix vector multiplication
Characteristics of Task Interactions

• Aspects of interactions
  – What: shared data or synchronizations, and sizes of the media

• Read-only interactions
  – Tasks only read data items associated with other tasks

• Read-write interactions
  – Read, as well as modify data items associated with other tasks.
  – Harder to code
    • Require additional synchronization primitives
      – to avoid read-write and write-write ordering races
Characteristics of Task Interactions

• Aspects of interactions
  – What: shared data or synchronizations, and sizes of the media
  – When: the timing
  – Who: with which task(s), and overall topology/patterns
  – Do we know details of the above three before execution
  – How: involve one or both?
    • The implementation concern, implicit or explicit

• One-sided
  – initiated & completed independently by 1 of 2 interacting tasks
    • GET and PUT

• Two-sided
  – both tasks coordinate in an interaction
    • SEND + RECV
Today’s lecture

• Decomposition Techniques - continued
  – Exploratory Decomposition
  – Hybrid Decomposition

Mapping tasks to processes/cores/CPU/PEs

• Characteristics of Tasks and Interactions
  – Task Generation, Granularity, and Context
  – Characteristics of Task Interactions

Mapping Techniques for Load Balancing
  – Static and Dynamic Mapping

• Methods for Minimizing Interaction Overheads
• Parallel Algorithm Design Models
Mapping Techniques

- Parallel algorithm design
  - Program decomposed
  - Characteristics of task and interactions identified

Assign large amount of concurrent tasks to equal or relatively small amount of processes for execution

- Though often we do 1:1 mapping
Mapping Techniques

- Goal of mapping: minimize overheads
  - There is cost to do parallelism
    - Interactions and idling (serialization)

- Contradicting objectives: interactions vs idling
  - Idling (serialization) ↑: insufficient parallelism
  - Interactions ↑: excessive concurrency

- E.g. Assigning all work to one processor trivially minimizes interaction at the expense of significant idling.
Mapping Techniques for Minimum Idling

- Execution: alternating stages of computation and interaction

- Mapping must simultaneously minimize idling and load balance
  - Idling means not doing useful work
  - Load balance: doing the same amount of work

- Merely balancing load does not minimize idling

A poor mapping, 50% waste
Mapping Techniques for Minimum Idling

Static or dynamic mapping

• Static Mapping
  – Tasks are mapped to processes a-priori
  – Need a good estimate of task sizes
  – Optimal mapping may be NP complete

• Dynamic Mapping
  – Tasks are mapped to processes at runtime
  – Because:
    • Tasks are generated at runtime
    • Their sizes are not known.

• Other factors determining the choice of mapping techniques
  – the size of data associated with a task
  – the characteristics of inter-task interactions
  – even the programming models and target architectures
Schemes for Static Mapping

- Mappings based on data decomposition
  - Mostly 1-1 mapping

- Mappings based on task graph partitioning
- Hybrid mappings
# Mappings Based on Data Partitioning

- Partition the computation using a combination of
  - Data decomposition
  - The "owner-computes" rule

**Example: 1-D block distribution of 2-D dense matrix**

1-1 mapping of task/data and process

<table>
<thead>
<tr>
<th>row-wise distribution</th>
<th>column-wise distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P_0 )</td>
<td>( P_0 )</td>
</tr>
<tr>
<td>( P_1 )</td>
<td>( P_1 )</td>
</tr>
<tr>
<td>( P_2 )</td>
<td>( P_2 )</td>
</tr>
<tr>
<td>( P_3 )</td>
<td>( P_3 )</td>
</tr>
<tr>
<td>( P_4 )</td>
<td>( P_4 )</td>
</tr>
<tr>
<td>( P_5 )</td>
<td>( P_5 )</td>
</tr>
<tr>
<td>( P_6 )</td>
<td>( P_6 )</td>
</tr>
<tr>
<td>( P_7 )</td>
<td>( P_7 )</td>
</tr>
</tbody>
</table>
In general, higher dimension decomposition allows the use of larger # of processes.
Block Array Distribution Schemes: Examples

**Multiplying two dense matrices:** \( A \times B = C \)

- Partition the output matrix \( C \) using a block decomposition
  - Load balance: Each task compute the same number of elements of \( C \)
  - Note: each element of \( C \) corresponds to a single dot product
- The choice of precise decomposition: 1-D (row/col) or 2-D
  - Determined by the associated communication overhead

\[
\begin{align*}
\begin{bmatrix}
A_{11} & A_{12} & A_{13} \\
A_{21} & A_{22} & A_{23} \\
A_{31} & A_{32} & A_{33}
\end{bmatrix} \times 
\begin{bmatrix}
B_{11} & B_{12} & B_{13} \\
B_{21} & B_{22} & B_{23} \\
B_{31} & B_{32} & B_{33}
\end{bmatrix} &= 
\begin{bmatrix}
C_{11} & C_{12} & C_{13} \\
C_{21} & C_{22} & C_{23} \\
C_{31} & C_{32} & C_{33}
\end{bmatrix}
\end{align*}
\]

\[
\begin{align*}
C_{11} &= A_{11}B_{11} + A_{12}B_{21} + A_{13}B_{13} \\
C_{21} &= A_{21}B_{11} + A_{22}B_{21} + A_{23}B_{13} \\
C_{31} &= A_{31}B_{11} + A_{32}B_{21} + A_{33}B_{13} \\
C_{12} &= A_{11}B_{12} + A_{12}B_{22} + A_{13}B_{13} \\
C_{22} &= A_{21}B_{12} + A_{22}B_{22} + A_{23}B_{13} \\
C_{32} &= A_{31}B_{12} + A_{32}B_{22} + A_{33}B_{13} \\
C_{13} &= A_{11}B_{13} + A_{12}B_{23} + A_{13}B_{13} \\
C_{23} &= A_{21}B_{13} + A_{22}B_{23} + A_{23}B_{13} \\
C_{33} &= A_{31}B_{13} + A_{32}B_{23} + A_{33}B_{13}
\end{align*}
\]
Block Distribution and Data Sharing for Dense Matrix Multiplication

- **Row-based 1-D**
  
  ![Row-based 1-D Diagram]

- **Column-based 1-D**
  
  ![Column-based 1-D Diagram]

- **Row/Col-based 2-D**
  
  ![Row/Col-based 2-D Diagram]
Cyclic and Block Cyclic Distributions

- Consider a block distribution for LU decomposition (Gaussian Elimination)
  - The amount of computation per data item varies
  - Block decomposition would lead to significant load imbalance

```
1. procedure COL_LU (A)
2. begin
3.   for k := 1 to n do
4.     for j := k to n do
6.     endfor;
7.   for j := k + 1 to n do
8.     for i := k + 1 to n do
10.    endfor;
11.   endfor;
/*
After this iteration, column A[k + 1 : n, k] is logically the kth column of L and row A[k, k : n] is logically the kth row of U.
*/
```
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: \( A_{3,3} \rightarrow L_{3,3}U_{3,3} \)
Block Distribution for LU

Notice the significant load imbalance

<table>
<thead>
<tr>
<th></th>
<th>P₀</th>
<th></th>
<th>P₆</th>
</tr>
</thead>
<tbody>
<tr>
<td>T₁</td>
<td></td>
<td>T₄</td>
<td></td>
</tr>
<tr>
<td>P₁</td>
<td>T₂</td>
<td>P₄</td>
<td></td>
</tr>
<tr>
<td>T₃</td>
<td></td>
<td>T₇</td>
<td>T₁₁</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>P₃</th>
<th></th>
<th>P₇</th>
</tr>
</thead>
<tbody>
<tr>
<td>T₄</td>
<td></td>
<td>T₅</td>
<td></td>
</tr>
<tr>
<td>P₄</td>
<td>P₆</td>
<td>T₈</td>
<td>T₁₂</td>
</tr>
<tr>
<td>T₁₀</td>
<td></td>
<td>T₉</td>
<td>T₁₃ T₁₄</td>
</tr>
</tbody>
</table>


Block Cyclic Distributions

• Variation of the block distribution scheme
  – Partition an array into many more blocks (i.e. tasks) than the number of available processes.
  – Blocks are assigned to processes in a round-robin manner so that each process gets several non-adjacent blocks.
  – N-1 mapping of tasks to processes

• Used to alleviate the load-imbalance and idling problems.
Block-Cyclic Distribution for Gaussian Elimination

• Active submatrix shrinks as elimination progresses

• Assigning blocks in a block-cyclic fashion
  – Each PEs receives blocks from different parts of the matrix
  – In one batch of mapping, the PE doing the most will most likely receive the least in the next batch

\[
\]

\[
\]
Block-Cyclic Distribution

- A cyclic distribution: a special case with block size = 1
- A block distribution: a special case with block size = $n/p$
  - $n$ is the dimension of the matrix and $p$ is the # of processes.
Block Partitioning and Random Mapping

Sparse matrix computations

• Load imbalance using block-cyclic partitioning/mapping
  – more non-zero blocks to diagonal processes $P_0$, $P_5$, $P_{10}$, and $P_{15}$ than others
  – $P_{12}$ gets nothing
Block Partitioning and Random Mapping

\[ V = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11] \]

\[ \text{random}(V) = [8, 2, 6, 0, 3, 7, 11, 1, 9, 5, 4, 10] \]

\[ \text{mapping} = 8 \ 2 \ 6 \ 0 \ 3 \ 7 \ 11 \ 1 \ 9 \ 5 \ 4 \ 10 \]

\[ P_0 \quad P_1 \quad P_2 \quad P_3 \]

\[ \begin{array}{cccc}
0 & 1 & 2 & 3 \\
4 & 5 & 6 & 7 \\
8 & 9 & 10 & 11 \\
\end{array} \quad \begin{array}{cccc}
10 & 11 & 12 & 13 \\
14 & 15 & & \\
& & & \\
\end{array} \quad \begin{array}{cccc}
\begin{array}{cccc}
P_0 & P_1 & P_2 & P_3 \\
& P_4 & P_5 & P_6 & P_7 \\
& P_8 & P_9 & P_{10} & P_{11} \\
& P_{12} & P_{13} & P_{14} & P_{15} \\
\end{array}
\end{array} \]
Graph Partitioning Based Data Decomposition

• Array-based partitioning and static mapping
  – Regular domain, i.e. rectangular, mostly dense matrix
  – Structured and regular interaction patterns
  – Quite effective in balancing the computations and minimizing the interactions

• Irregular domain
  – Spars matrix-related
  – Numerical simulations of physical phenomena
    • Car, water/blood flow, geographic

• Partition the irregular domain so as to
  – Assign equal number of nodes to each process
  – Minimizing edge count of the partition.
Partitioning the Graph of Lake Superior

- Each mesh point has the same amount of computation
  - Easy for load balancing
- Minimize edges
- Optimal partition is an NP-complete
  - Use heuristics

Random Partitioning

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

- Schemes for Static Mapping
  - Mappings based on data partitioning
    - Mostly 1-1 mapping
  - Mappings based on task graph partitioning
  - Hybrid mappings

- Data partitioning
  - Data decomposition and then 1-1 mapping of tasks to PEs

**Partitioning a given task-dependency graph across processes**
- An optimal mapping for a general task-dependency graph
  - NP-complete problem.
- Excellent heuristics exist for structured graphs.
Mapping a Binary Tree Dependency Graph

Mapping dependency graph of quicksort to processes in a hypercube

- Hypercube: n-dimensional analogue of a square and a cube
  - node numbers that differ in 1 bit are adjacent
Mapping a Sparse Graph

Sparse matrix vector multiplication

Using data partitioning
Mapping a Sparse Graph

Sparse matrix vector multiplication

Using task graph partitioning

<table>
<thead>
<tr>
<th>Process</th>
<th>Elements to Communicate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process 0</td>
<td>0,4,5,8</td>
</tr>
<tr>
<td>Process 1</td>
<td>1,2,3,7</td>
</tr>
<tr>
<td>Process 2</td>
<td>6,9,10,11</td>
</tr>
</tbody>
</table>

C0 = (1,2,6,9)

C1 = (0,5,6)

C2 = (1,2,4,5,7,8)
Hierarchical/Hybrid Mappings

• A single mapping is inadequate.
  – E.g. task graph mapping of the binary tree (quicksort) cannot use a large number of processors.

• Hierarchical mapping
  – Task graph mapping at the top level
  – Data partitioning within each level.
Today’s lecture

- Decomposition Techniques - continued
  - Exploratory Decomposition
  - Hybrid Decomposition

**Mapping tasks to processes/cores/CPU/PEs**

- Characteristics of Tasks and Interactions
  - Task Generation, Granularity, and Context
  - Characteristics of Task Interactions

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

- Methods for Minimizing Interaction Overheads
- Parallel Algorithm Design Models
Schemes for Dynamic Mapping

• Also referred to as dynamic load balancing
  – Load balancing is the primary motivation for dynamic mapping.

• Dynamic mapping schemes can be
  – Centralized
  – Distributed
Centralized Dynamic Mapping

• Processes are designated as masters or slaves
  – Workers (slave is politically incorrect)
• General strategies
  – Master has pool of tasks and as central dispatcher
  – When one runs out of work, it requests from master for more work.
• Challenge
  – When process # increases, master may become the bottleneck.
• Approach
  – Chunk scheduling: a process picks up multiple tasks at once
  – Chunk size:
    • Large chunk sizes may lead to significant load imbalances as well
    • Schemes to gradually decrease chunk size as the computation progresses.
Distributed Dynamic Mapping

• All processes are created equal
  – Each can send or receive work from others
    • Alleviates the bottleneck in centralized schemes.
• Four critical design 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.
• Workstealing
Today’s lecture

• Decomposition Techniques - continued
  – Exploratory Decomposition
  – Hybrid Decomposition

**Mapping tasks to processes/cores/CPU/PEs**

• Characteristics of Tasks and Interactions
  – Task Generation, Granularity, and Context
  – Characteristics of Task Interactions

• Mapping Techniques for Load Balancing
  – Static and Dynamic Mapping

☞ Methods for Minimizing Interaction Overheads

• Parallel Algorithm Design Models
Minimizing Interaction Overheads

Rules of thumb

• Maximize data locality
  – Where possible, reuse intermediate data
  – Restructure computation so that data can be reused in smaller time windows.

• Minimize volume of data exchange
  – partition interaction graph to minimize edge crossings

• Minimize frequency of interactions
  – Merge multiple interactions to one, e.g. aggregate small msgs.

• Minimize contention and hot-spots
  – Use decentralized techniques
  – Replicate data where necessary
Minimizing Interaction Overheads (continued)

Techniques

• Overlapping computations with interactions
  – Use non-blocking communications
  – Multithreading
  – Prefetching to hide latencies.

• Replicating data or computations to reduce communication

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

• Overlap interactions with other interactions.
Today’s lecture

• Decomposition Techniques - continued
  – Exploratory Decomposition
  – Hybrid Decomposition

  Mapping tasks to processes/cores/CPU/PEs

• Characteristics of Tasks and Interactions
  – Task Generation, Granularity, and Context
  – Characteristics of Task Interactions

• Mapping Techniques for Load Balancing
  – Static and Dynamic Mapping

• Methods for Minimizing Interaction Overheads
  ➜ Parallel Algorithm Design Models
Parallel Algorithm Models

• Ways of structuring parallel algorithm
  – Decomposition techniques
  – Mapping technique
  – Strategy to minimize interactions.

• Data Parallel Model
  – Each task performs similar operations on different data
  – Tasks are statically (or semi-statically) mapped to processes

• Task Graph Model
  – Use task dependency graph to guide the model for better locality or low interaction costs.
Parallel Algorithm Models (continued)

• Master-Slave Model
  – Master (one or more) generate work
  – Dispatch work to workers.
  – Dispatching may be static or dynamic.

• Pipeline / Producer-Consumer Model
  – Stream of data is passed through a succession of processes, each of which perform some task on it
  – Multiple stream concurrently

• Hybrid Models
  – Applying multiple models hierarchically
  – Applying multiple models sequentially to different phases of a parallel algorithm.
References

• Adapted from slides “Principles of Parallel Algorithm Design” by Ananth Grama

• Based on Chapter 3 of “Introduction to Parallel Computing” by Ananth Grama, Anshul Gupta, George Karypis, and Vipin Kumar. Addison Wesley, 2003