Principles of Parallel Algorithm Design:
Concurrency and Mapping

John Mellor-Crummey

Department of Computer Science
Rice University

johnmc@rice.edu
Last Thursday

• Introduction to parallel algorithms
  — tasks and decomposition
  — threads and mapping
  — threads versus cores

• Decomposition techniques - part 1
  — recursive decomposition
  — data decomposition
Topics for Today

• Decomposition techniques - part 2
  — exploratory decomposition
  — hybrid decomposition

• Characteristics of tasks and interactions

• Mapping techniques for load balancing
  — static mappings
  — dynamic mappings

• Methods for minimizing interaction overheads

• Parallel algorithm design templates
Exploratory Decomposition

- Exploration (search) of a state space of solutions
  - problem decomposition reflects shape of execution
- Examples
  - discrete optimization
    - 0/1 integer programming
  - theorem proving
  - game playing
Exploratory Decomposition Example

Solving a 15 puzzle

• Sequence of three moves from state (a) to final state (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**

- Parallel formulation may perform a different amount of work

![Diagram](image)

Total serial work = $2m + 1$

Total parallel work = 4

- Can cause super- or sub-linear speedup
Speculative Decomposition

• Dependencies between tasks are not always known *a-priori*
  —makes it impossible to identify independent tasks

• Conservative approach
  —identify independent tasks only when no dependencies left

• Optimistic (speculative) approach
  —schedule tasks even when they may potentially be erroneous

• Drawbacks for each
  —conservative approaches
    – may yield little concurrency
  —optimistic approaches
    – may require a roll-back mechanism if a dependence is encountered
Speculative Decomposition In Practice

Discrete event simulation

- Data structure: centralized time-ordered event list
- 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

Time Warp
David Jefferson. “Virtual Time,”
ACM TOPLAS, 7(3):404-425, July 1985
Hybrid Decomposition

Use multiple decomposition strategies together

Often necessary for adequate concurrency

- **Quicksort**
  - recursive decomposition alone limits concurrency (why?)

- **Climate simulation**
  - data parallelism can be applied within atmosphere, ocean, land, and sea-ice simulations
CESM Simulations on a Cray XT

Performance Limiters: Left is CAM; Right is POP.

Figure courtesy of Pat Worley (ORNL)
Topics for Today

• Decomposition techniques - part 2
  —exploratory decomposition
  —hybrid decomposition

❖ Characteristics of tasks and interactions

• Mapping techniques for load balancing
  —static mappings
  —dynamic mappings

• Methods for minimizing interaction overheads

• Parallel algorithm design templates
Characteristics of Tasks

• Key characteristics
  — generation strategy
  — associated work
  — associated data size

• Impact choice and performance of parallel algorithms
Task Generation

- **Static task generation**
  - identify concurrent tasks a-priori
  - typically decompose using data or recursive decomposition
  - examples
    - matrix operations
    - graph algorithms
    - image processing applications
    - other regularly structured problems

- **Dynamic task generation**
  - identify concurrent tasks as a computation unfolds
  - typically decompose using exploratory or speculative decompositions
  - examples
    - puzzle solving
    - game playing
Task Size

- Uniform: all the same size
- Non-uniform
  - sometimes sizes known or can be estimated *a-priori*
  - sometimes not
    - example: tasks in quicksort
      size of each partition depends upon pivot selected
Size of Data Associated with Tasks

- Data may be small or large compared to the computation
  - 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

- Implications
  - small data: task can easily migrate to another thread
  - large data: ties the task to a thread
    - possibly can avoid communicating the task context
      reconstruct/recompute the context elsewhere
Characteristics of Task Interactions

Orthogonal classification criteria

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

• **Static interactions**
  — tasks and interactions are known a-priori
  — simpler to code

• **Dynamic interactions**
  — timing or interacting tasks cannot be determined a-priori
  — harder to code
    – especially using two-sided message passing APIs
Characteristics of Task Interactions

• Regular interactions
  —interactions have a pattern that can be described with a function
    − e.g. mesh, ring
  —regular patterns can be exploited for efficient implementation
    − e.g. schedule communication to avoid conflicts on network links

• Irregular interactions
  —lack a well-defined topology
  —modeled by a graph
Static Regular Task Interaction Pattern

Image operations, e.g., edge detection

Nearest neighbor interactions on a 2D mesh

Sobel Edge Detection Stencils

\[
G_x = \begin{bmatrix}
-1 & 0 & +1 \\
-2 & 0 & +2 \\
-1 & 0 & +1 \\
\end{bmatrix}
\]

\[
G_y = \begin{bmatrix}
-1 & -2 & -1 \\
0 & 0 & 0 \\
+1 & +2 & +1 \\
\end{bmatrix}
\]
Static Irregular Task Interaction Pattern

Sparse matrix-vector multiply

(a) Task 0
(b) Task 11
Characteristics of Task Interactions

- Read-only interactions
  - tasks only read data associated with other tasks
- Read-write interactions
  - read and modify data associated with other tasks
  - harder to code: requires synchronization
    - need to avoid read-write and write-write ordering races
Characteristics of Task Interactions

- **One-sided**
  - initiated & completed independently by 1 of 2 interacting tasks
    - READ or WRITE
    - GET or PUT

- **Two-sided**
  - both tasks coordinate in an interaction
    - SEND and RECV
Topics for Today

- Decomposition techniques - part 2
  - exploratory decomposition
  - hybrid decomposition

- Characteristics of tasks and interactions

- Mapping techniques for load balancing
  - static mappings
  - dynamic mappings

- Methods for minimizing interaction overheads

- Parallel algorithm design templates
Mapping Techniques

Map concurrent tasks to processes for execution

- Overheads of mappings
  - serialization (idling)
  - communication

- Select mapping to minimize overheads

- Conflicting objectives: minimizing one increases the other
  - assigning all work to one processor
    - minimizes communication
    - significant idling
  - minimizing serialization introduces communication
Mapping Techniques for Minimum Idling

- Must simultaneously minimize idling and load balance
- Balancing load alone does not minimize idling
Mapping Techniques for Minimum Idling

Static vs. dynamic mappings

• Static mapping
  — *a-priori* mapping of tasks to processes
  — requirements
    - a good estimate of task size
    - even so, optimal mapping may be NP complete
      e.g., multiple knapsack problem

• Dynamic mapping
  — map tasks to processes at runtime
  — why?
    - tasks are generated at runtime, or
    - their sizes are unknown

Factors that influence choice of mapping
  • size of data associated with a task
  • nature of underlying domain
Schemes for Static Mapping

- Data partitionings
- Task graph partitionings
- Hybrid strategies
Mappings Based on Data Partitioning

Partition computation using a combination of

—data partitioning
—owner-computes rule

Example: 1-D block distribution for dense matrices

row-wise distribution

column-wise distribution
Block Array Distribution Schemes

Multi-dimensional block distributions

Multi-dimensional partitioning enables larger # of processes
Block Array Distribution Example

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 dot product
  — even load balance
- Obvious choices: 1D or 2D decomposition
- Select to minimize associated communication overhead
Data Usage in Dense Matrix Multiplication

\[ \begin{align*}
\text{Input} & \quad \times \quad \text{Input} & \quad = \quad \text{Output} \\
\text{Input} & \quad \times \quad \text{Input} & \quad = \quad \text{Output}
\end{align*} \]
Consider: Gaussian Elimination

Active submatrix shrinks as elimination progresses


Imbalance and Block Array Distributions

- Consider a block distribution for Gaussian Elimination
  — amount of computation per data item varies
  — a block decomposition would lead to significant load imbalance
Block Cyclic Distribution

Variant of the block distribution scheme that can be used to alleviate the load-imbalance and idling

Steps

1. partition an array into many more blocks than the number of available processes
2. assign blocks to processes in a round-robin manner
   - each process gets several non-adjacent blocks
Block-Cyclic Distribution

- Cyclic distribution: special case with block size = 1
- Block distribution: special case with block size is $n/p$
  —$n$ is the dimension of the matrix; $p$ is the # of processes
Decomposition by Graph Partitioning

Sparse-matrix vector multiply

- Graph of the matrix is useful for decomposition
  - work \( \sim \) number of edges
  - communication for a node \( \sim \) node degree

- Goal: balance work & minimize communication

- Partition the graph
  - assign equal number of nodes to each process
  - minimize edge count of the graph partition
Partitioning a Graph of Lake Superior

Random Partitioning

Partitioning for minimum edge-cut
Partitioning a task-dependency graph

- Optimal partitioning for general task-dependency graph — NP-complete problem
- Excellent heuristics exist for structured graphs
Mapping a Sparse Matrix

Sparse matrix-vector product

sparse matrix structure

mapping
partitioning

17 items to communicate

C0 = (4,5,6,7,8)
C1 = (0,1,2,3,8,9,10,11)
C2 = (0,4,5,6)
Mapping a Sparse Matrix

Sparse matrix-vector product

sparse matrix structure

mapping partitioning
Hierarchical Mappings

• Sometimes a single mapping is inadequate
  —e.g., task mapping of quicksort binary tree cannot readily use a large number of processors.

• Hierarchical approach
  —use a task mapping at the top level
  —data partitioning within each task
Topics for Today

• Decomposition techniques - part 2
  —exploratory decomposition
  —hybrid decomposition

• Characteristics of tasks and interactions

• Mapping techniques for load balancing
  —static mappings
  —dynamic mappings

• Methods for minimizing interaction overheads

• Parallel algorithm design templates
Schemes for Dynamic Mapping

- Dynamic mapping AKA dynamic load balancing
  - load balancing is the primary motivation for dynamic mapping
- Styles
  - centralized
  - distributed
Centralized Dynamic Mapping

- Processes = masters or slaves
- General strategy
  - when a slave runs out of work → request more from master
- Challenge
  - master may become bottleneck for large # of processes
- Approach
  - chunk scheduling: process picks up several of tasks at once
  - however
    - large chunk sizes may cause significant load imbalances
    - gradually decrease chunk size as the computation progresses
Distributed Dynamic Mapping

- All processes as peers
- Each process can send or receive work from other processes
  - avoids centralized bottleneck
- 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?
- Ideal answers can be application specific
- Cilk uses a distributed dynamic mapping: “work stealing”
Topics for Today

• Decomposition techniques - part 2
  —exploratory decomposition
  —hybrid decomposition

• Characteristics of tasks and interactions

• Mapping techniques for load balancing
  —static mappings
  —dynamic mappings

Methods for minimizing interaction overheads

• Parallel algorithm design templates
Minimizing Interaction Overheads (1)

“Rules of thumb”

• Maximize data locality
  — don’t fetch data you already have
  — restructure computation to reuse data promptly

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

• Minimize frequency of communication
  — try to aggregate messages where possible

• Minimize contention and hot-spots
  — use decentralized techniques (avoidance)
Minimizing Interaction Overheads (2)

Techniques

• Overlap communication with computation
  — use non-blocking communication primitives
    – overlap communication with your own computation
    – one-sided: prefetch remote data to hide latency
  — multithread code on a processor
    – overlap communication with another thread’s computation

• Replicate data or computation to reduce communication

• Use group communication instead of point-to-point primitives

• Issue multiple communications and overlap their latency
  (reduces exposed latency)
Topics for Today

• Decomposition techniques - part 2
  —exploratory decomposition
  —hybrid decomposition

• Characteristics of tasks and interactions

• Mapping techniques for load balancing
  —static mappings
  —dynamic mappings

• Methods for minimizing interaction overheads

Parallel algorithm design templates
Parallel Algorithm Model

• Definition: ways of structuring a parallel algorithm

• Aspects of a model
  — decomposition
  — mapping technique
  — strategy to minimize interactions
Common Parallel Algorithm Templates

• Data parallel
  — each task performs similar operations on different data
  — typically statically map tasks to processes

• Task graph
  — use task dependency graph relationships to
    – promote locality, or reduce interaction costs

• Master-slave
  — one or more master processes generate work
  — allocate it to worker processes
  — allocation may be static or dynamic

• Pipeline / producer-consumer
  — pass a stream of data through a sequence of processes
  — each performs some operation on it

• Hybrid
  — apply multiple models hierarchically, or
  — apply multiple models in sequence to different phases
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