Electrical Engineering and Computer Science

EECS 358 - INTRODUCTION TO PARALLEL COMPUTING

Lecture 13

Designing Efficient Parallel Algorithms

Outline

- A Methodology for Parallel Algorithm Design
- Partitioning
- Communication
- Agglomeration
- Mapping
- READING: I. Foster, "Designing and Building Efficient Parallel Programs", Chapter 2.

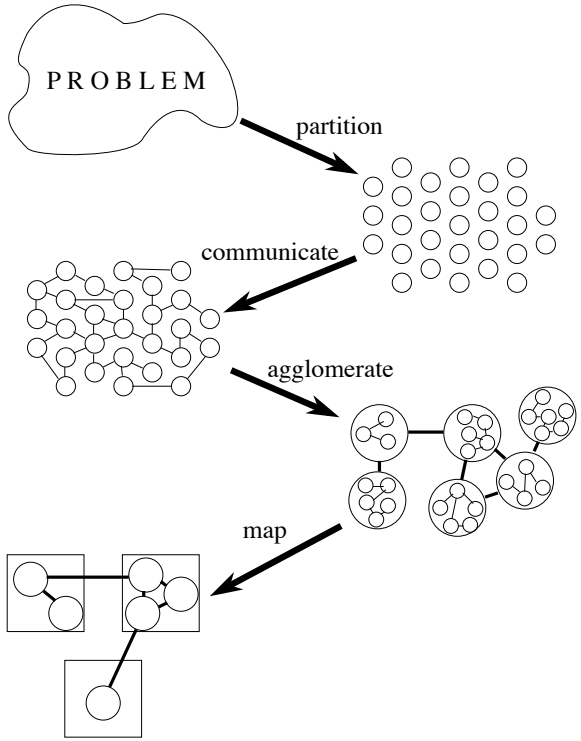
Desirable Attributes of Parallel Algorithms

- *Concurrency*: ability to perform many actions simultaneously
- *Scalability*: resilience to increasing processor counts
- *Locality:* a high ratio of local memory accesses to remote memory accesse.
- *Modularity:* the decomposition of complex entities into simpler components

Methodology of Parallel Algorithm Design

- Four stages of Parallel Algorithm Design
  - Partitioning
  - Communication
  - Agglomeration
  - Mapping
- Partitioning and Communication deal with machine independent issues and affect concurrency and scalability.
- Agglomeration and Mapping deal with machine dependent issues and affect locality and other performance issues.

Design Methodology



Partitioning

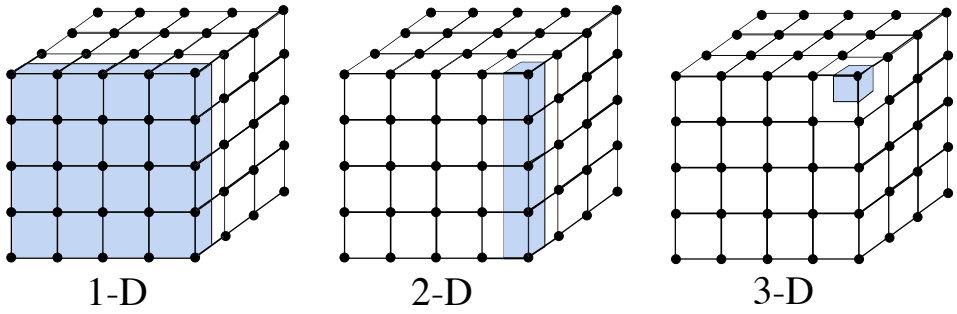
- Expose opportunities for parallel execution
- Fine-grained decomposition: define a large number of small tasks
- Later, may forgo some parallelism to reduce communication, or increase locality.
- Good partition divides into small pieces both *computation* and *data*.
- Two approaches
  - Domain decomposition
  - Functional decomposition

Domain Decomposition

- First decompose the data associated with a problem.
- If possible, divide data into small pieces of equal size.
- Next, partition computations that operates on each data.
- Generates a a number of tasks, each with some data and set of operations on data.
- Communication is needed to move data between tasks.

Domain Decomposition Example

- 1-, 2-, or 3- dimensional decomposition of a 3-d model of atmospheric model

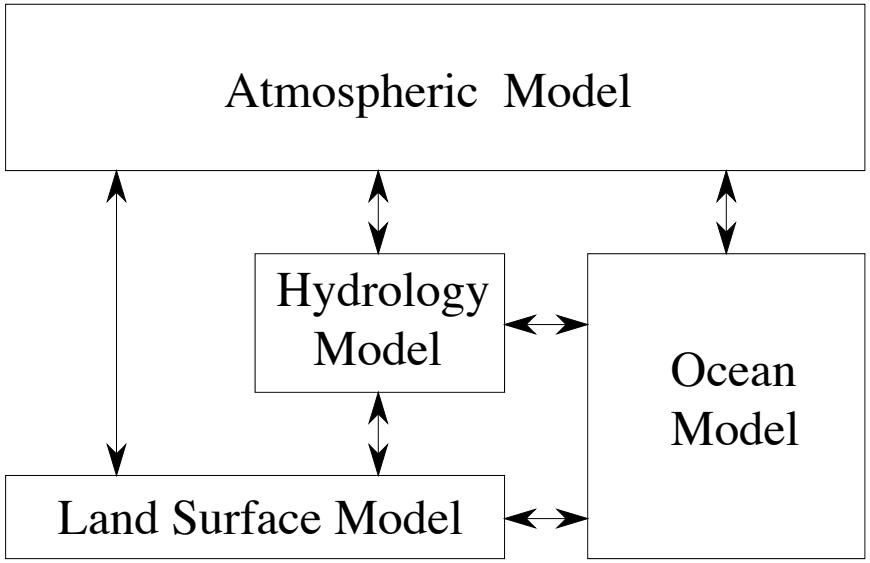


Functional Decomposition

- Focus is computation that is to be performed, rather than data.
- Divide computation into separate tasks.
- Data requirements of these tasks may be disjoint or they may overlap significantly
- An example of problem for which functional decomposition is appropriate is a search tree problem.

Example of Functional Decomposition

- A simulation of the earth's climate comprises atmosphere, ocean, hydrology, etc.



Partitioning Checklist

- Does your partition define at least an order of magnitude more tasks that there are processors?
- Does your partition avoid redundant computation and storage requirements?
- Are tasks of equal size?
- Does number of tasks scale with problem size?
- Have you identified alternate partitions?

Communication

- Tasks generated by above partitioning intended to execute concurrently, but not independently
- The computation to be performed in one task requires data from another.
- Information flow is specified in communication phase.
- We can conceptualize a need for communication between two tasks as a channel linking tasks (message passing)
- Above is easy for functional decomposition, not easy for domain decomposition

Categories of Communication

- Local versus global
  - Local communication: each task communicates with small set of other tasks.
  - Global communication: each task communicates with many tasks.
- Structured versus unstructured
  - Structured communication: a task and its neighbors form a regular structure: tree, grid
  - Unstructured communication: arbitrary graphs

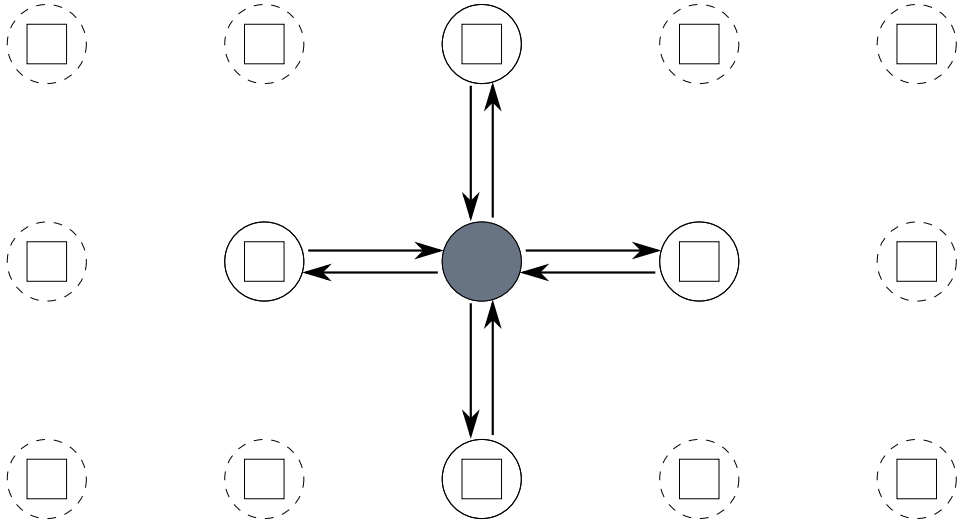
Categories of Communication (Contd)

- Static versus dynamic
  - Static: identity of communcation partners does not change
  - Dynamic: communication partners determined at runtime, data dependent
- Synchronous versus asynchronous
  - Synchronous: Producers and consumers execute in coordinated fashion.
  - Asynchronous: may require consumer obtain data without cooperation of producer.

Local Communication

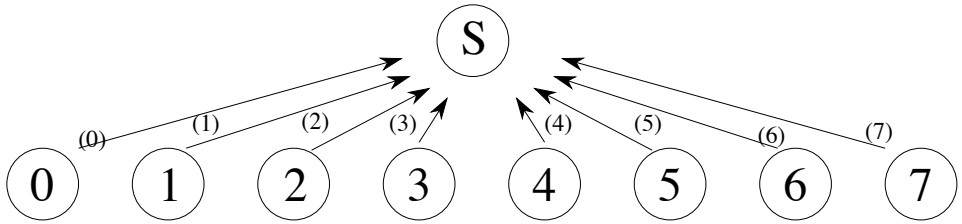
- Consider communication requirements of Jacobi finite difference schems

$$X_{i,j}^{(t+1)} = \frac{4X_{i,j}^{(t)} + X_{i-1,j}^{(t)} + X_{i+1,j}^{(t)} + X_{i,j-1}^{(t)} + X_{i,j+1}^{(t)}}{8}$$



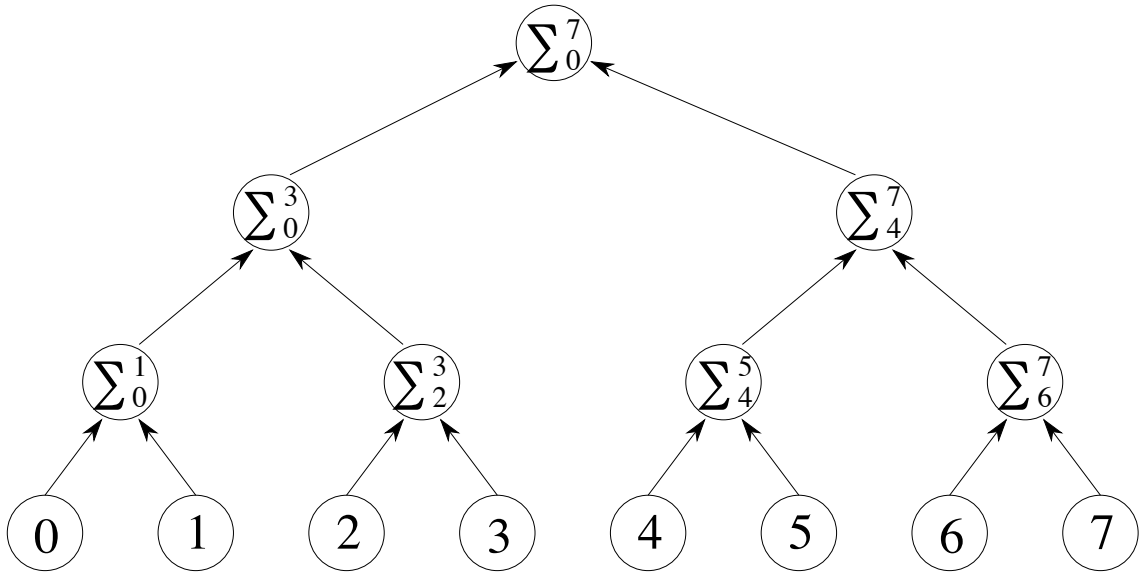
Global Communication

- Consider problem of performing a parallel reduction of N numbers



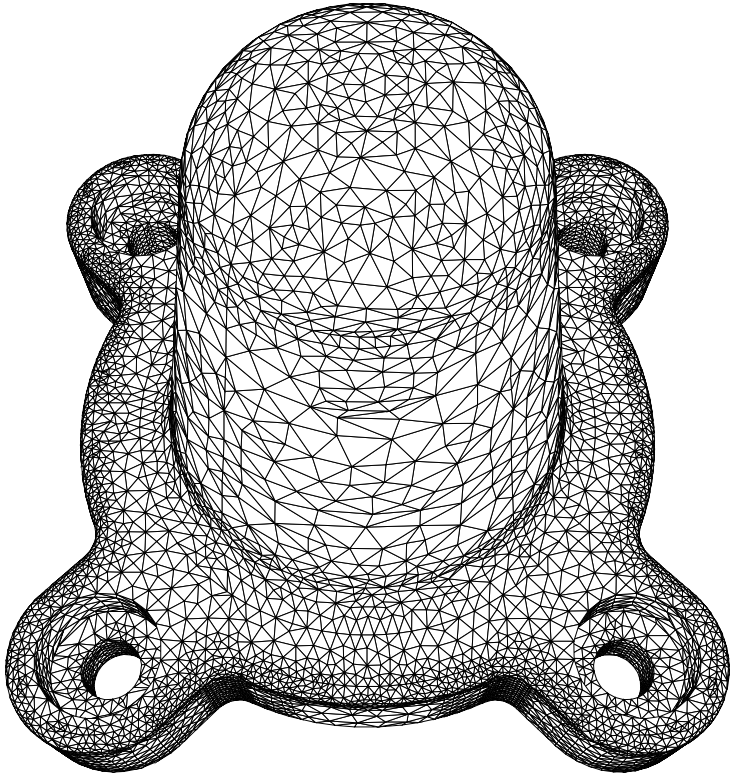
Divide and Conquer

- Previous approach was completely serial
- Can exploit concurrency



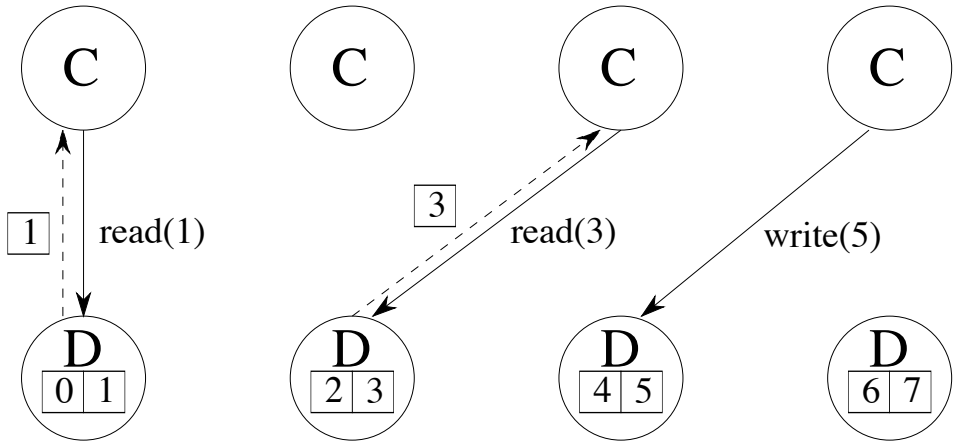
Unstructured and Dynamic Communication

- Finite element mesh problem



Asynchronous Communication

- In asynchronous communication, tasks that possess data (producers) are not able to determine when other tasks (consumers) will require data.



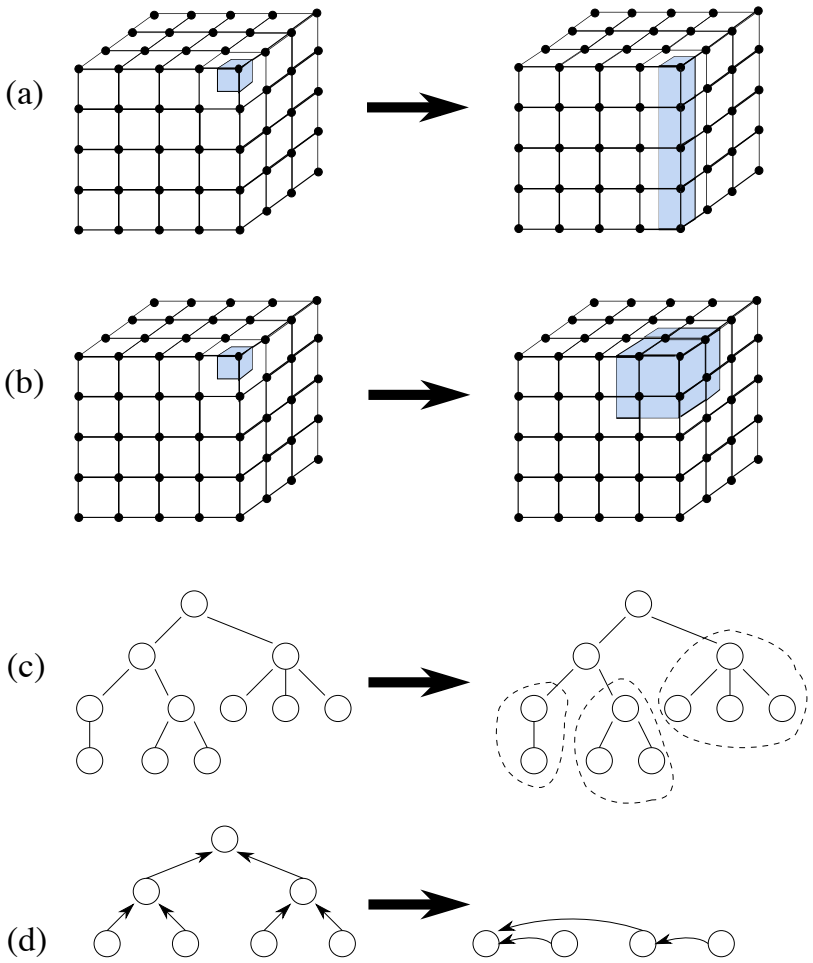
Communication Design Checklist

- Do all tasks perform same amount of communication operations?
- Does each task communicate with small number of neighbors?
- Are communication operations able to proceed concurrently?
- Is computation for different tasks able to proceed concurrently?

Agglomeration

- In first two phases, we partitioned the computation to be performed into a set of tasks and introduced communication to provide data required by these tasks.
- In agglomerate stage, we consider if it is useful to combine tasks to provide smaller number of tasks
- Determine if worthwhile to replicate data or computation.

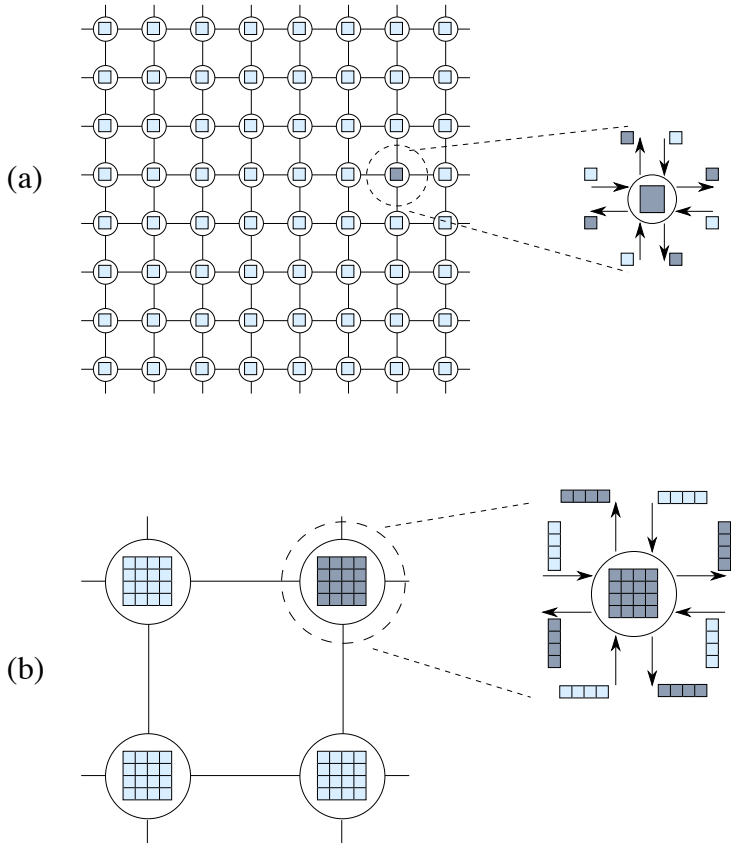
Examples of Agglomeration



Increasing Granularity

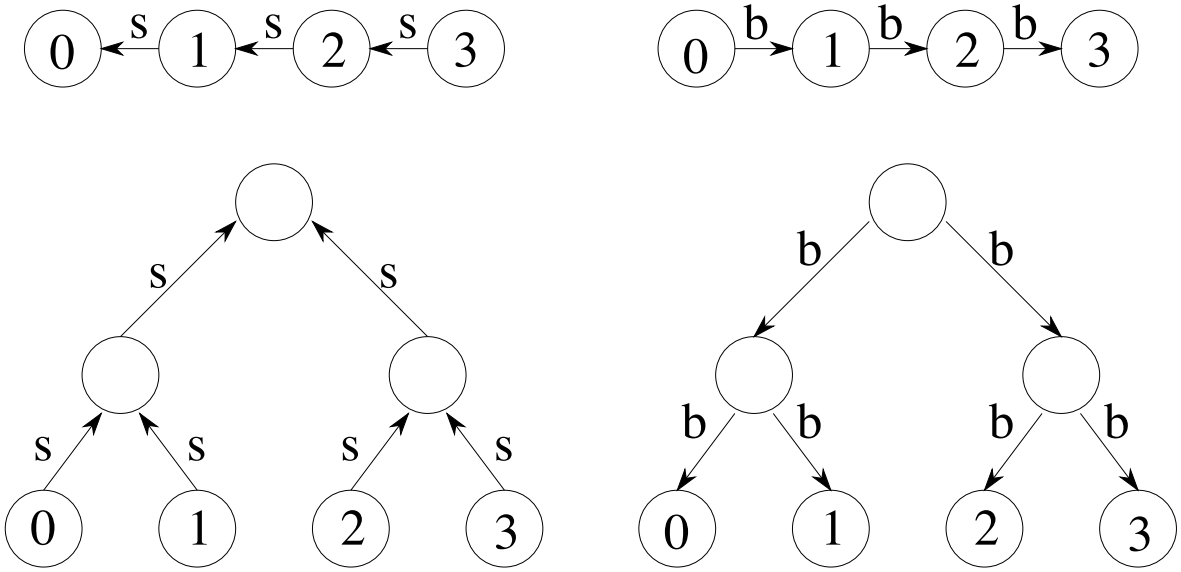
- In partitioning phase, efforts focused on exposing parallelism.
- Large number of parallel tasks does not result in efficient parallel algorithm
- For communication, send same data in less number of messages
- Also, less task creation costs and task scheduling costs

Increased Granularity of Communication



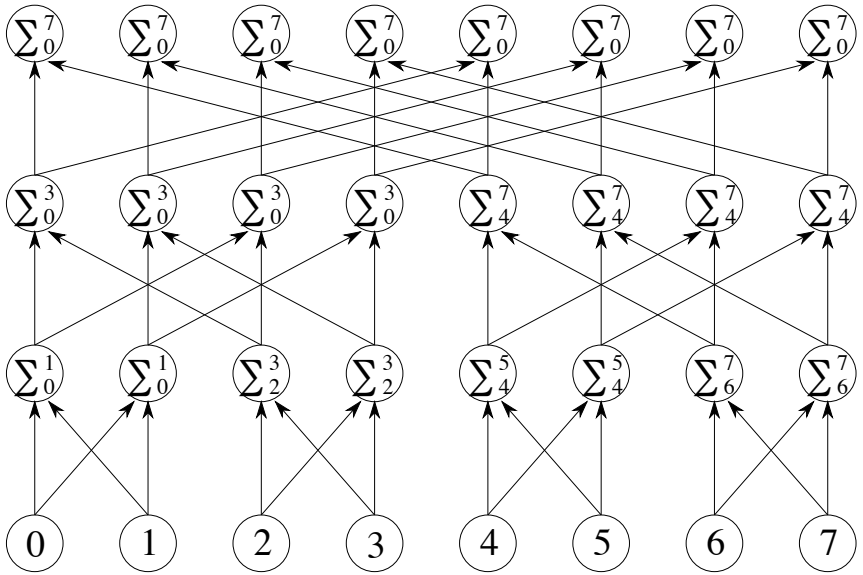
Replicating Computations

- Consider problem of replicating the sum of N numbers in N processors
- Use a sum reduction followed by a broadcast takes $2(N-1)$ in ring and $2 \log N$ for a tree.



Replicating Computations

- Can accomplish in log (N) steps in butterfly algorithm



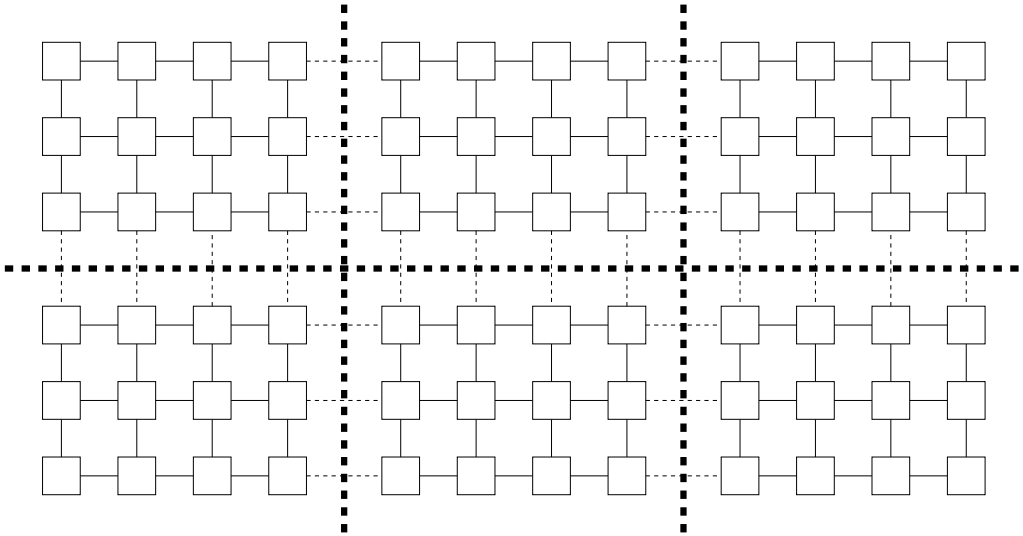
Agglomeration Checklist

- Has agglomeration reduced communication costs by increasing locality?
- If agglomeration uses replicated computations, do benefits outweigh costs?
- If agglomeration replicates data, is scalability affected?
- Has agglomeration yielded tasks of similar computation and communication costs?
- Does number of tasks still scale with problem size?

Mapping

- Final stage of parallel algorithm design
- Specify which processor each task will execute
- Place tasks that are able to execute concurrently on *different* processors
- Place tasks that communicate frequently to *same* processor
- These are conflicting goals

Example of Mapping



Mapping Problem

- Mapping problem to minimize execution time is NP-complete, i.e. no polynomial time algorithm exists for optimal solution.
- Hence resort to heuristics
- Many algorithms developed using domain decomposition techniques feature a fixed number of equal sized tasks and structured and regular communication.
- Then mapping is straightforward
- In more complex domain decomposition based algorithms with variable amount of work per task and unstructured communication, difficult to do mapping

Load Balancing Strategies

- Time required to execute these algorithms weighed against benefits of reduced execution costs
- Dynamic load balancing: where a load balancing algorithm is periodically executed to determine a new mapping.
- Local versus global algorithms
- Probabilistic versus deterministic

Recursive Bisection Load Balancing

- Recursive bisection techniques are used to partition a domain (e.g. finite element grid) into subdomain of approximately equal computational cost
- Attempt to minimize communication costs
- A divide and conquer approach is taken
- Most straightforward approach is recursive coordinate bisection.
- Makes cuts based on physical coordinates of domain
- At each step, subdivide along longer dimension (say x) so that at that step, points in one subdomain will have all x-coordinates greater than grid points in the other.
- Good job of partitioning computations equally, but does not take care of communication.

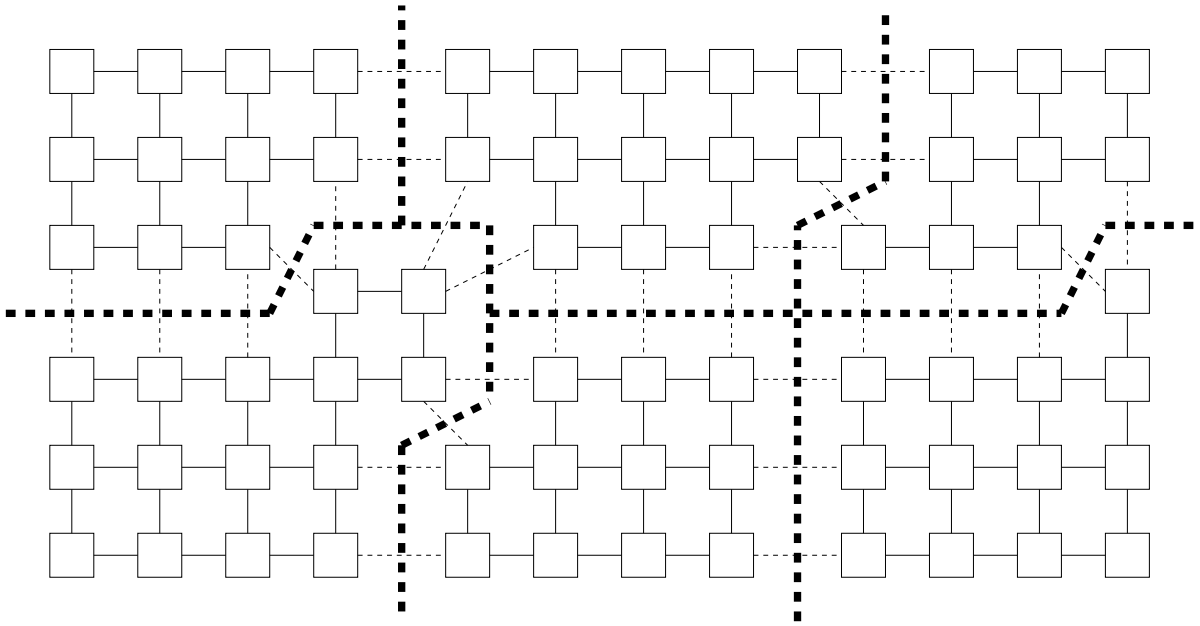
Other Recursive Bisection Techniques

- Unbalanced Recursive Bisection
- Recursive Graph Bisection
- Recursive Spectral Bisection

Local Load Balancing Algorithms

- Previous techniques are expensive since they require global knowledge of computation state
- Local algorithms compensate for changes in computational load using information from small number of neighboring processors.
- For example, if processors arranged as a logical mesh, each processor compares load with that of its neighbors and transfers computations if the difference in load exceeds some threshold.

Example of Local Load Balancing

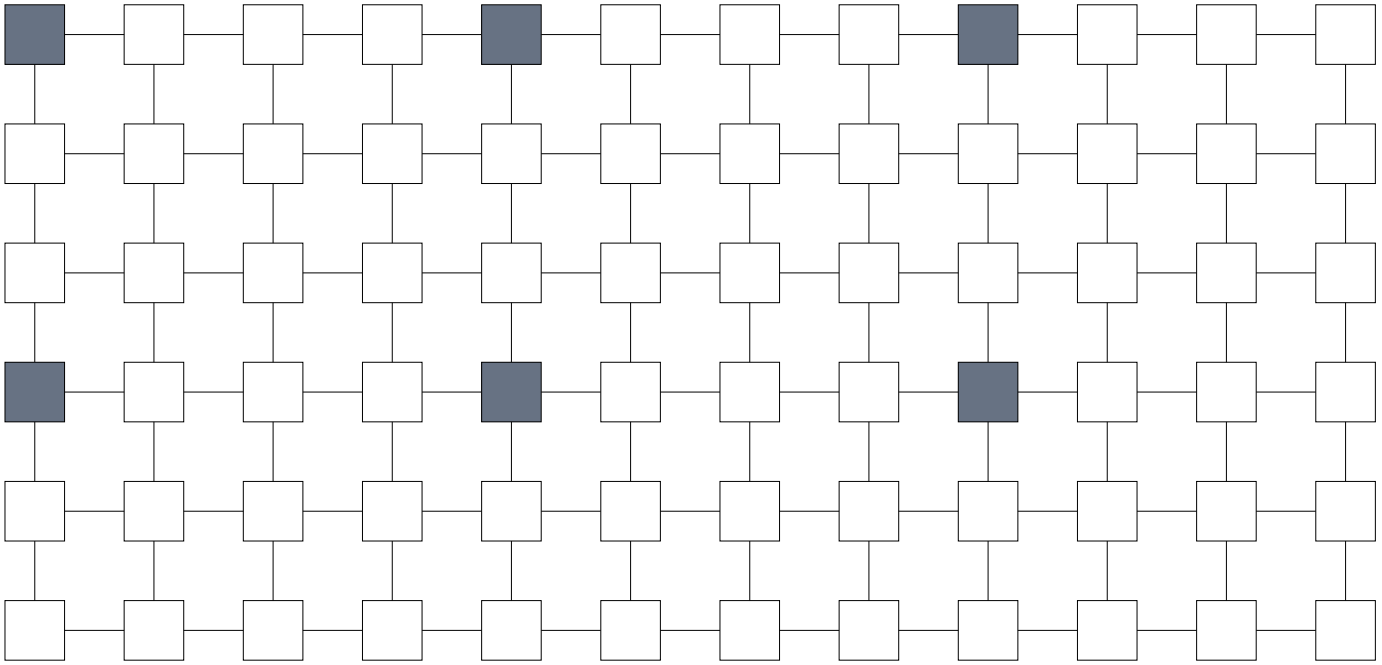


Probabilistic Load Balancing

- Simple load balancing method
- Allocate tasks randomly
- If number of tasks is large, scheme works well.
- Advantage: low cost, scalable
- Disadvantage: off-processor communication required for almost every tasks.

Cyclic Mappings

- Cyclic of scattered mapping
- Each of P processors is assigned every P^{th} task

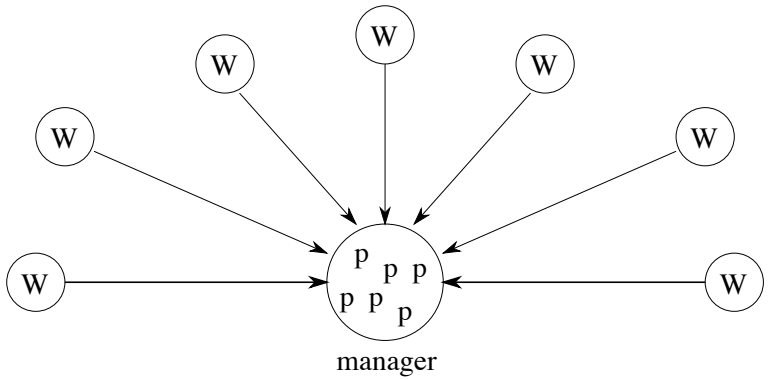


Task Scheduling Algorithms

- When a functional decomposition is used, need task scheduling
- Conflicting requirements of:
  - Independent operation to reduce communication costs
  - Global knowledge of computation state to improve load balance.
- Manager worker
- Decentralized Schemes

Manager-Worker Scheme

- Workers repeatedly request and process problem descriptions
- Manager maintains a pool of problem descriptions



Decentralized Schemes

- In completely decentralized schemes, there is no central manager.
- A separate task pool is maintained on each processor.
- Idle workers request problems from other processors.
- Task queue becomes distributed data structure accessed by different processors in asynchronous fashion.
- Variations: a worker may request work from a small number of predefined neighbors or may select other processors at random.

Mapping Design Checklist

- If considering an SPMD design for a complex problem, have you considered an algorithm based on dynamic task creation, and vice versa?
- If using a centralized load-balancing algorithm, have you verified that manager is not bottleneck?
- Within dynamic load balancing algorithms, have you evaluated costs of different strategies?

Summary

- A Methodology for Parallel Algorithm Design
- Partitioning
- Communication
- Agglomeration
- Mapping
- NEXT LECTURE: Search Algorithms
- READING: V. Kumar, "Introduction to Parallel Computing", Chapter 5.