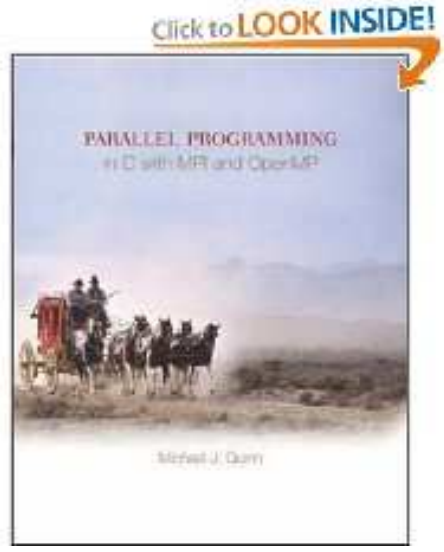
Parallel Algorithm Design

Overview

- Chapter 3 from *Michael J. Quinn*, **Parallel Programming in C with MPI and OpenMP**

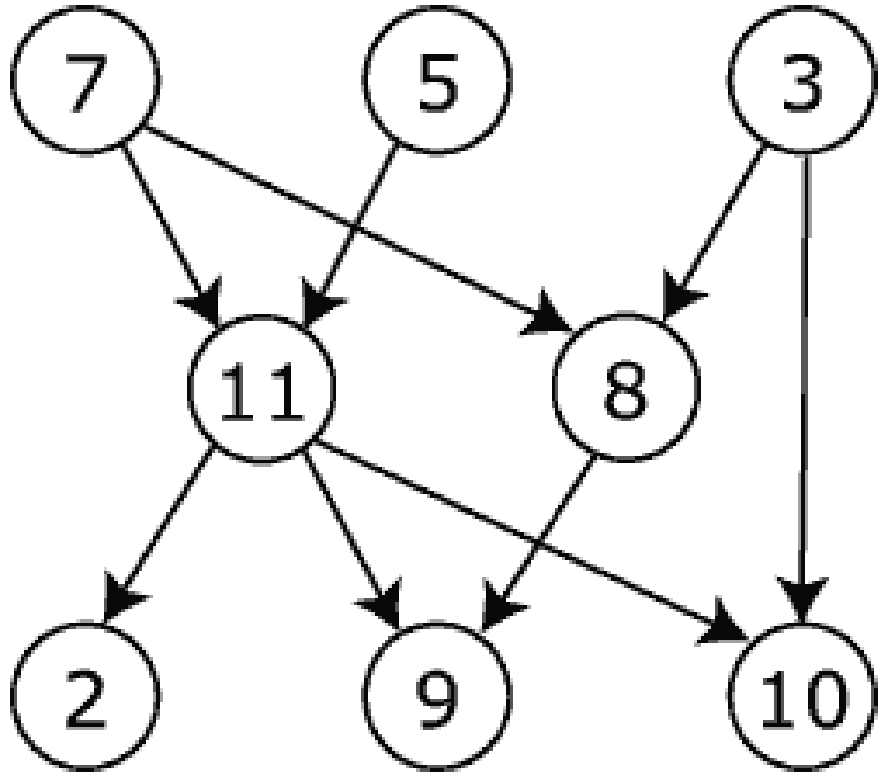


- Another resource: `http://www.mcs.anl.gov/~itf/dbpp/text/node14.html`
- How to develop a parallel algorithm?
  - Partitioning
  - Communication
  - Agglomeration
  - Mapping

General remarks

- Warning: Parallel algorithm design is not always easily reduced to simple recipes
- May still benefit from a methodical approach
- Intuition for good designs need to be developed gradually

The task/channel model



Directed-graph representation of a parallel computation:
A set of tasks may interact with each other by sending messages through channels

Foster's design methodology

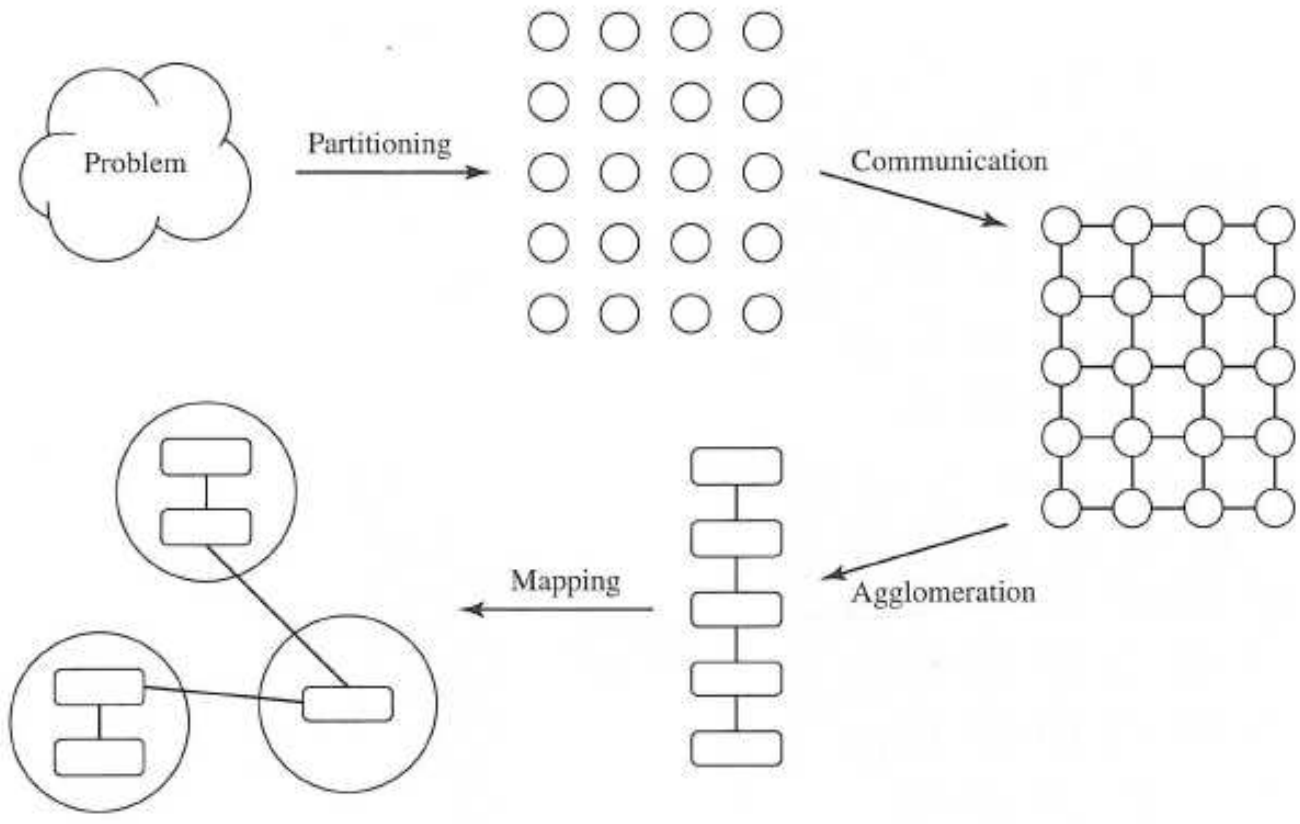


Figure 3.2 Foster's parallel algorithm design methodology.

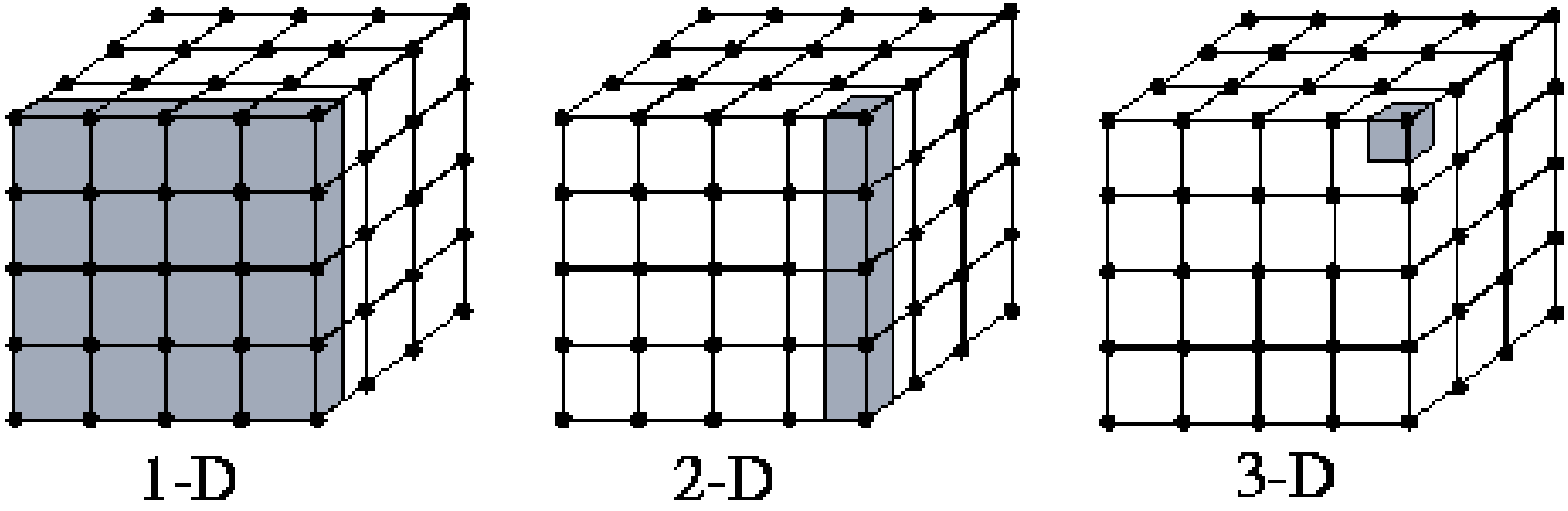
A four-step process for designing parallel algorithms

I. Foster, **Designing and Building Parallel Programs**, Addison-Wesley, 1995

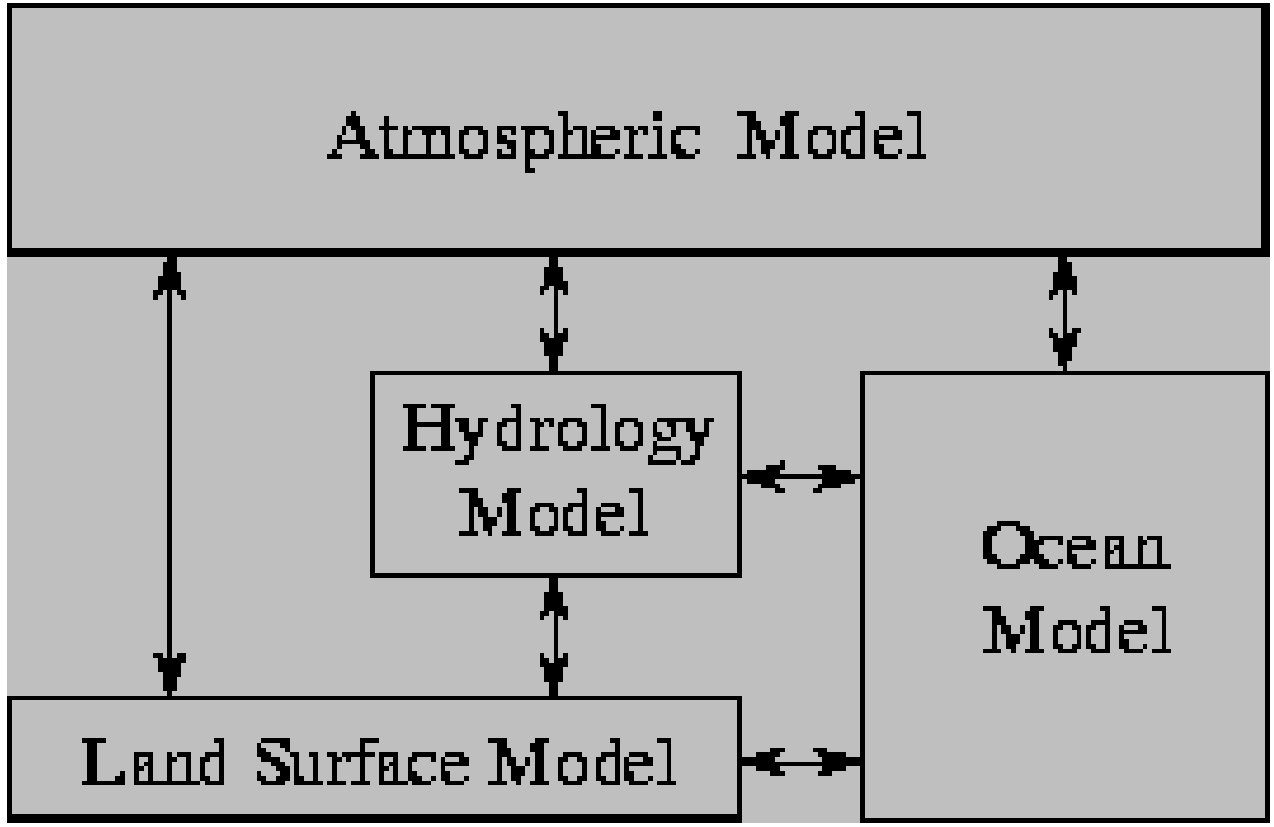
Partitioning

- At the beginning of design, discover as much parallelism as possible
- **Partitioning: divide the computation and data into pieces**
- Data-centric partitioning (domain decomposition)
  - Data is first divided into pieces, then computation is assigned to data
  - Result: a number of tasks, each having some data and a set of operations on the data
  - If an operation requires data from several tasks $\rightarrow$ communication needed
- Computation-centric partitioning (functional decomposition)
  - Computation is divided into disjoint tasks, then data is associated with the individual tasks
  - If data is shared between tasks $\rightarrow$ communication needed
  - Often a natural division of the code following the division of tasks

3 examples of domain decomposition



Example of functional decomposition



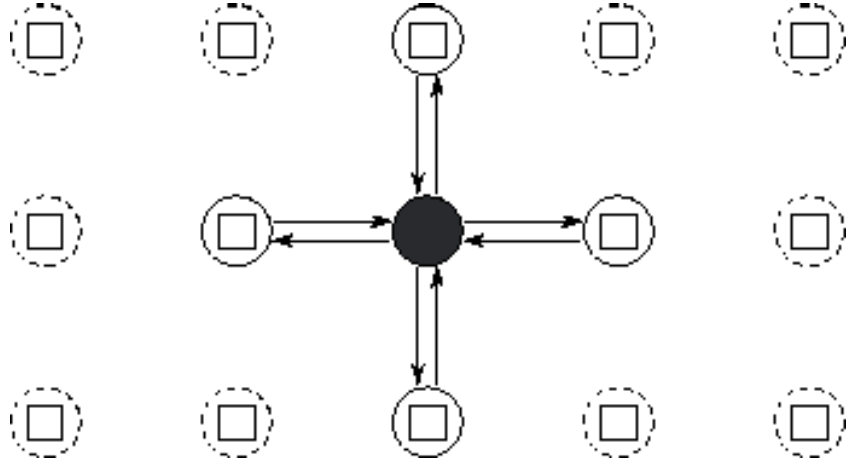
Partitioning design checklist

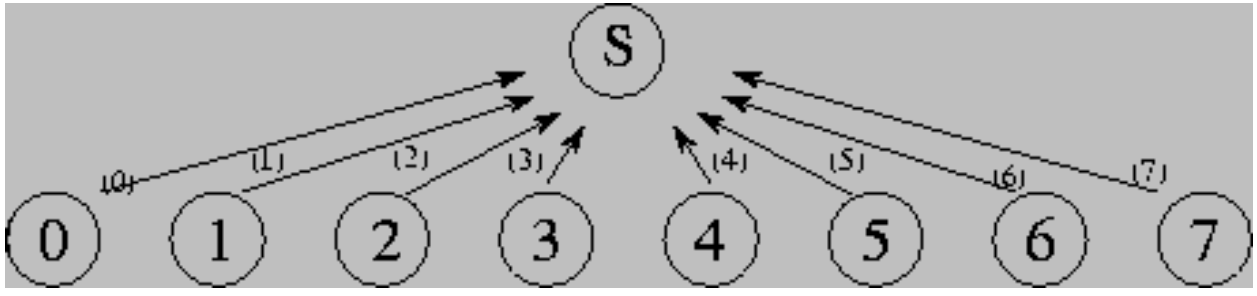
- There are many more tasks than processors
- Redundant computation and data storage are minimized
- Primitive tasks are roughly the same size
- The number of tasks is an increasing function of the problem size

Communication

- When primitive tasks are divided, determine the communication pattern
- Two kinds of communication
- **Local communication**: A task needs values from a small number of other tasks
- **Global communication**: A significant number of tasks must contribute data to perform a computation
- Communication among tasks is part of parallelization overhead

2 examples of communication





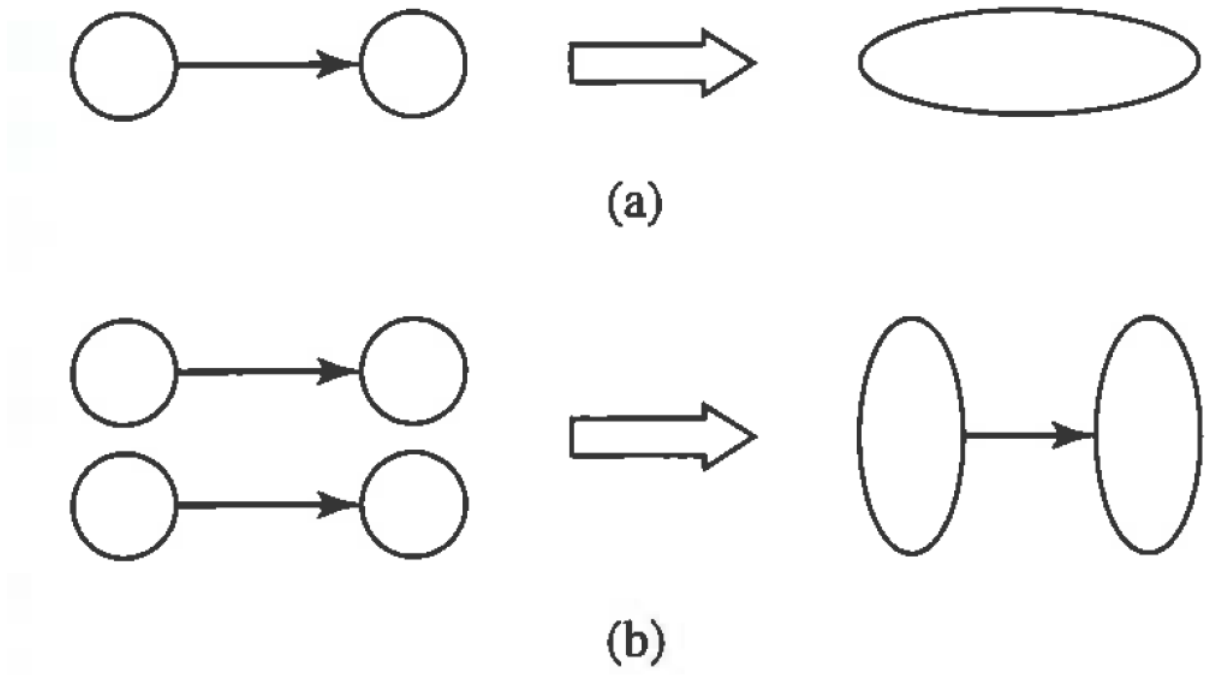
Communication design checklist

- The communication operations are balanced among the tasks
- Each task communicates with only a small number of neighbors
- Tasks can perform their communications concurrently
- Tasks can perform their computations concurrently

Agglomeration

- Motivation: If the number of tasks exceeds the processors by several orders of magnitude, creating these tasks will be a source of significant overhead. Also non-trivial: “map which tasks to which processors?”
- **Agglomeration is the process of grouping tasks into larger tasks**
- The purpose is to improve performance and simplify programming
- Typically in MPI programs, one consolidated task per processor
- Sometimes, more consolidated tasks than processors

2 examples of agglomeration



(a) Elimination of communication (increase the locality)
(b) Decrease of message transmissions

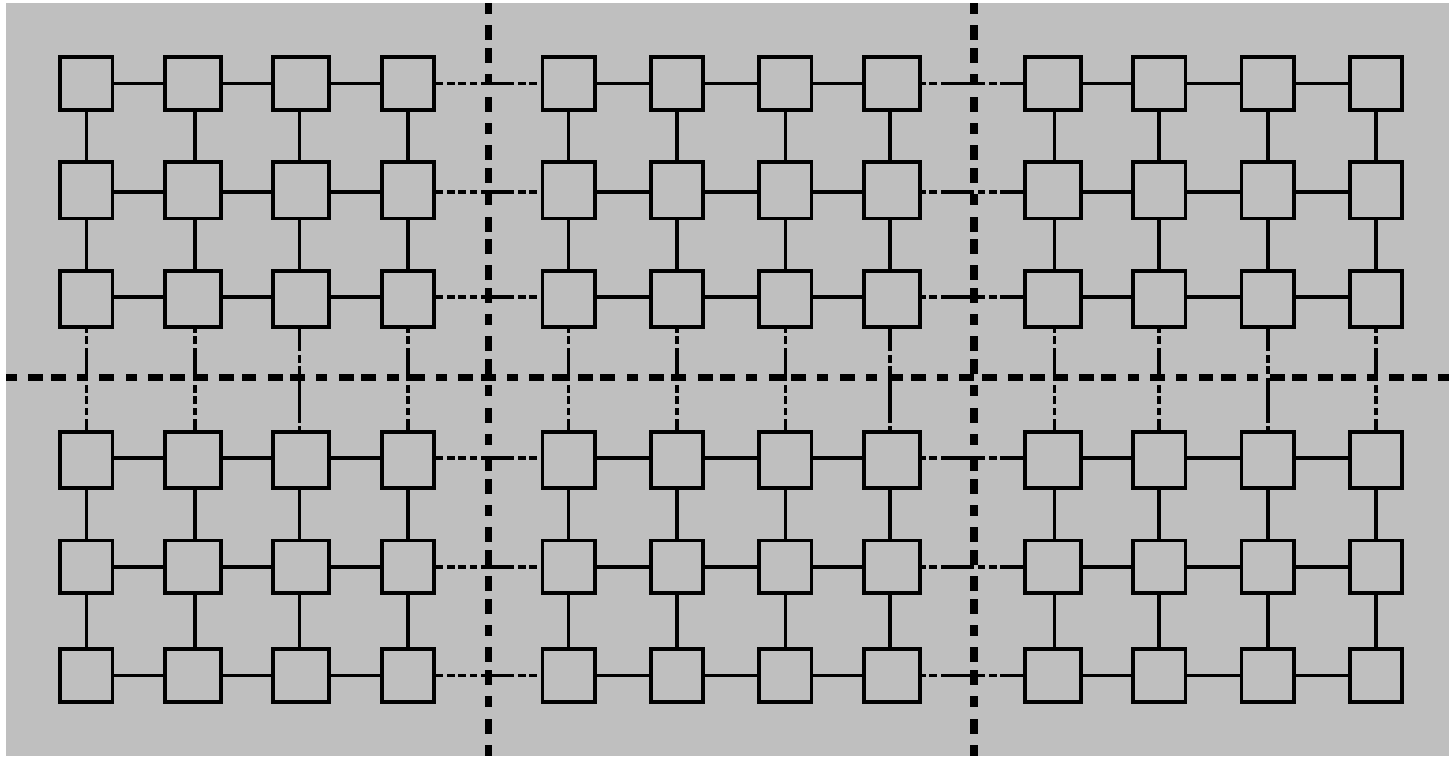
Agglomeration design checklist

- Locality is increased
- Replicated computations take less time than the communication they replace
- The amount of replicated data is small enough to allow the algorithm to scale
- Agglomerated tasks have similar computational and communication costs
- The number of tasks is an increasing function of the problem size
- The number of tasks is as small as possible, yet at least as great as the number of processors
- The trade-off between the chosen agglomeration and the cost of modification to existing sequential code is reasonable

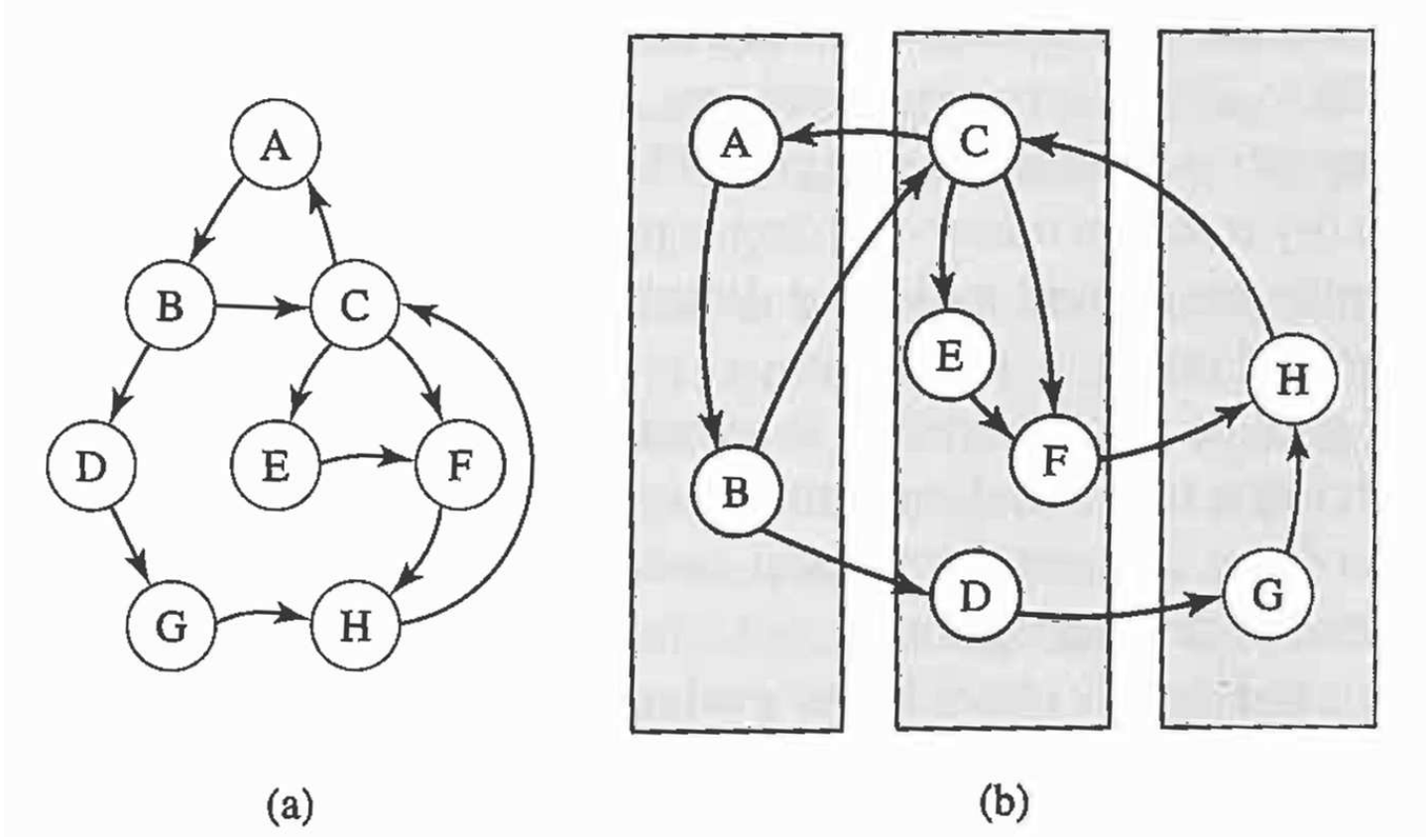
Mapping

- **Mapping: assigning tasks to processors**
- We focus on distributed-memory architecture
- Goals of mapping: maximize processor utilization and minimize interprocessor communication
- Processor utilization is maximized when the computation is balanced evenly
- Interprocessor communication decreases when two tasks connected by a channel are mapped to the same processor
- However, finding an ideal mapping is often NP-hard
- We must rely on heuristics that can do a reasonably good job of mapping

An example of good mapping



An example of bad mapping



Mapping design checklist

- Designs based on one task per processor and multiple tasks per processor have been considered
- Both static and dynamic allocations of tasks to processors have been evaluated
- If a dynamic allocation has been chosen, the manager is not a performance bottleneck

Example: boundary value problem

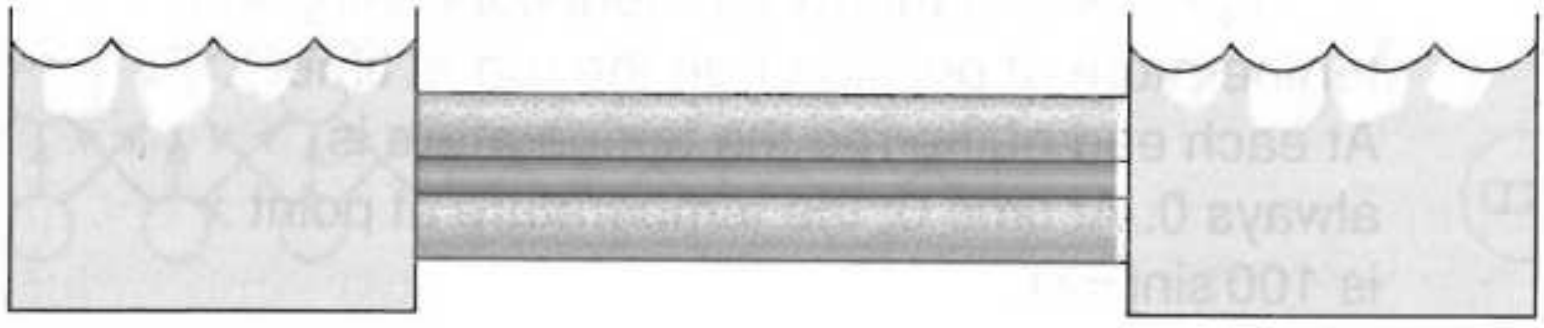


Figure 3.8 A thin rod (dark gray) is suspended between two ice baths. The ends of the rod are in contact with the icewater. The rod is surrounded by a thick blanket of insulation. We can use a partial differential equation to model the temperature at any point on the rod as a function of time.

Boundary value problem (2)

- 1D problem in the spatial direction
- Time-dependent problem
- We can use a finite difference mesh: uniform in both spatial and temporal directions
- Let $u_{i,j}$ denote the solution on spatial point i and time level j
- Computation formula

$$u_{i,j+1} = r\,u_{i-1,j} + (1-2r)u_{i,j} + r\,u_{i+1,j}$$

Boundary value problem (3)

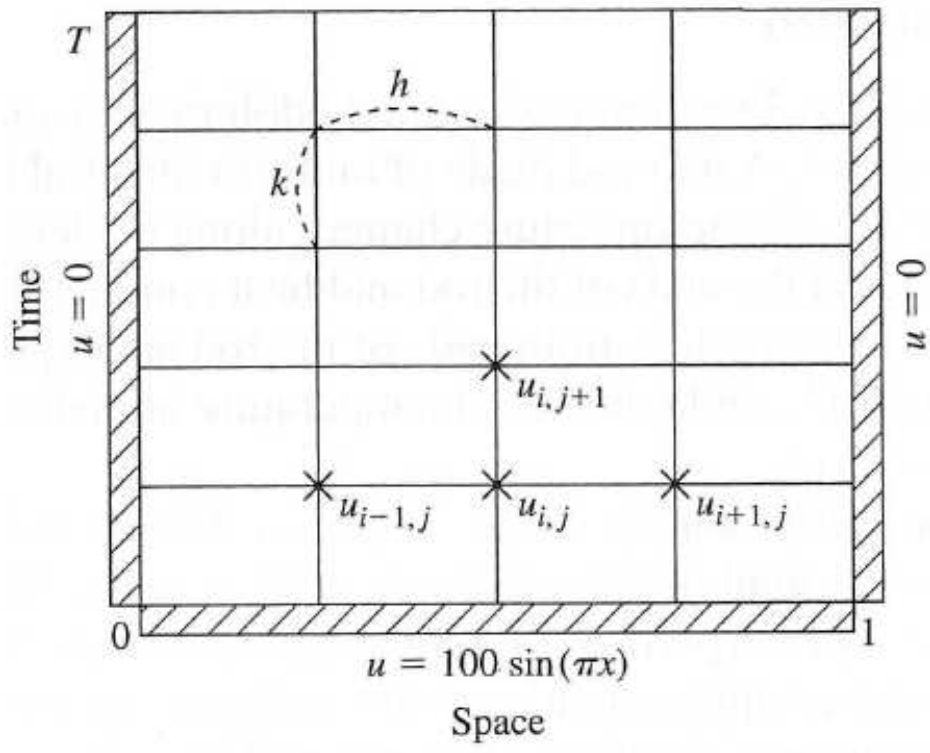
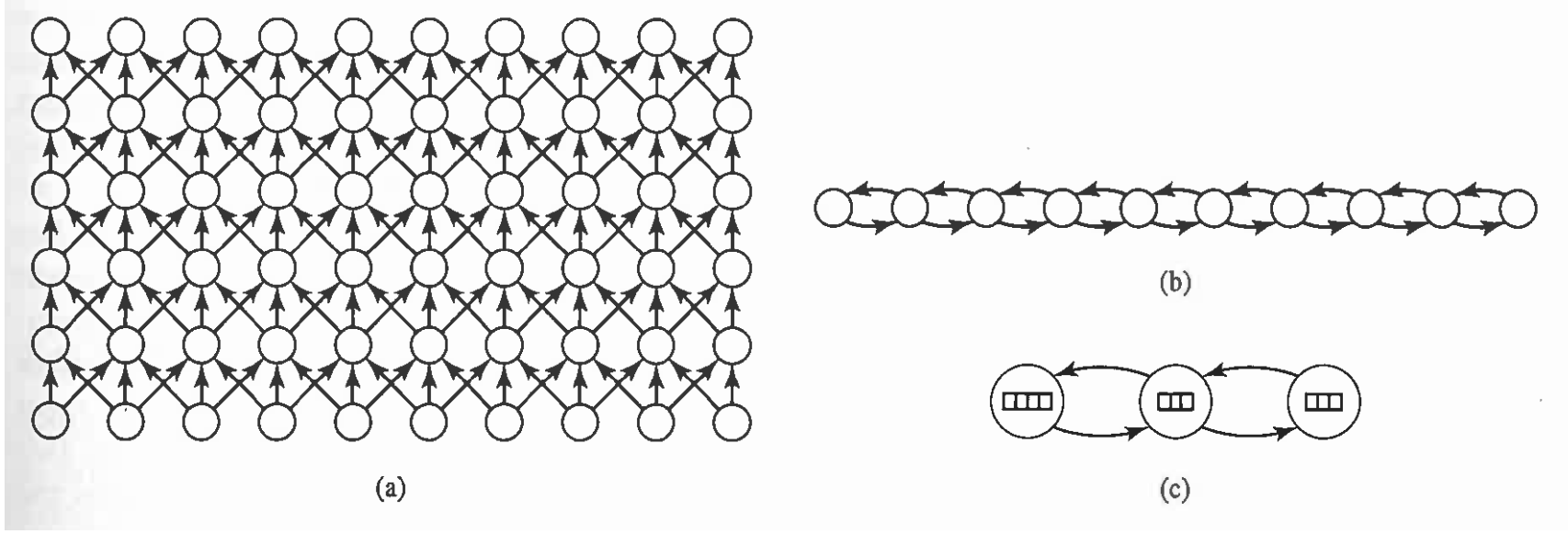


Figure 3.10 Data structure used in a finite difference approximation to the rod-cooling problem presented in Figure 3.8. Every point $u_{i,j}$ represents a matrix element containing the temperature at position i on the rod at time j.

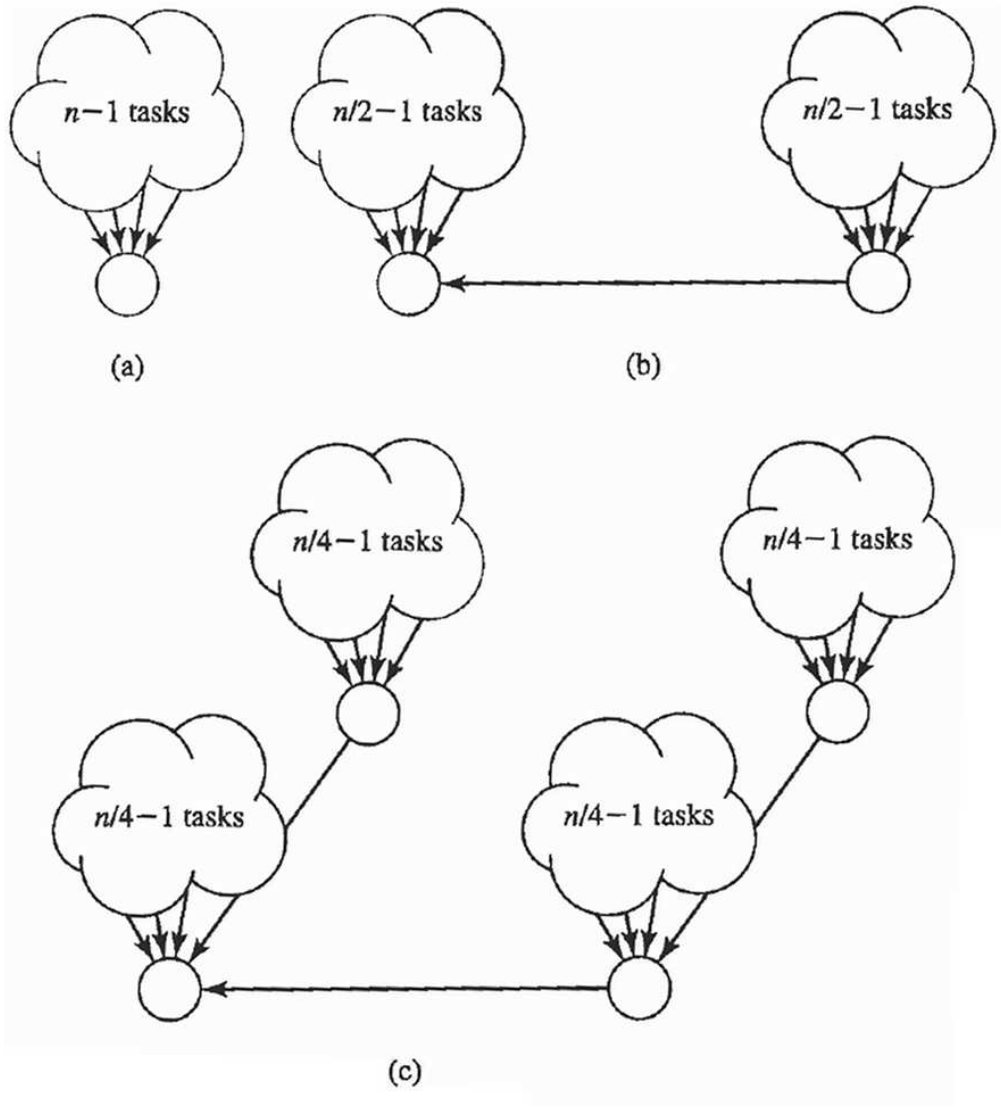
Boundary value problem (4)



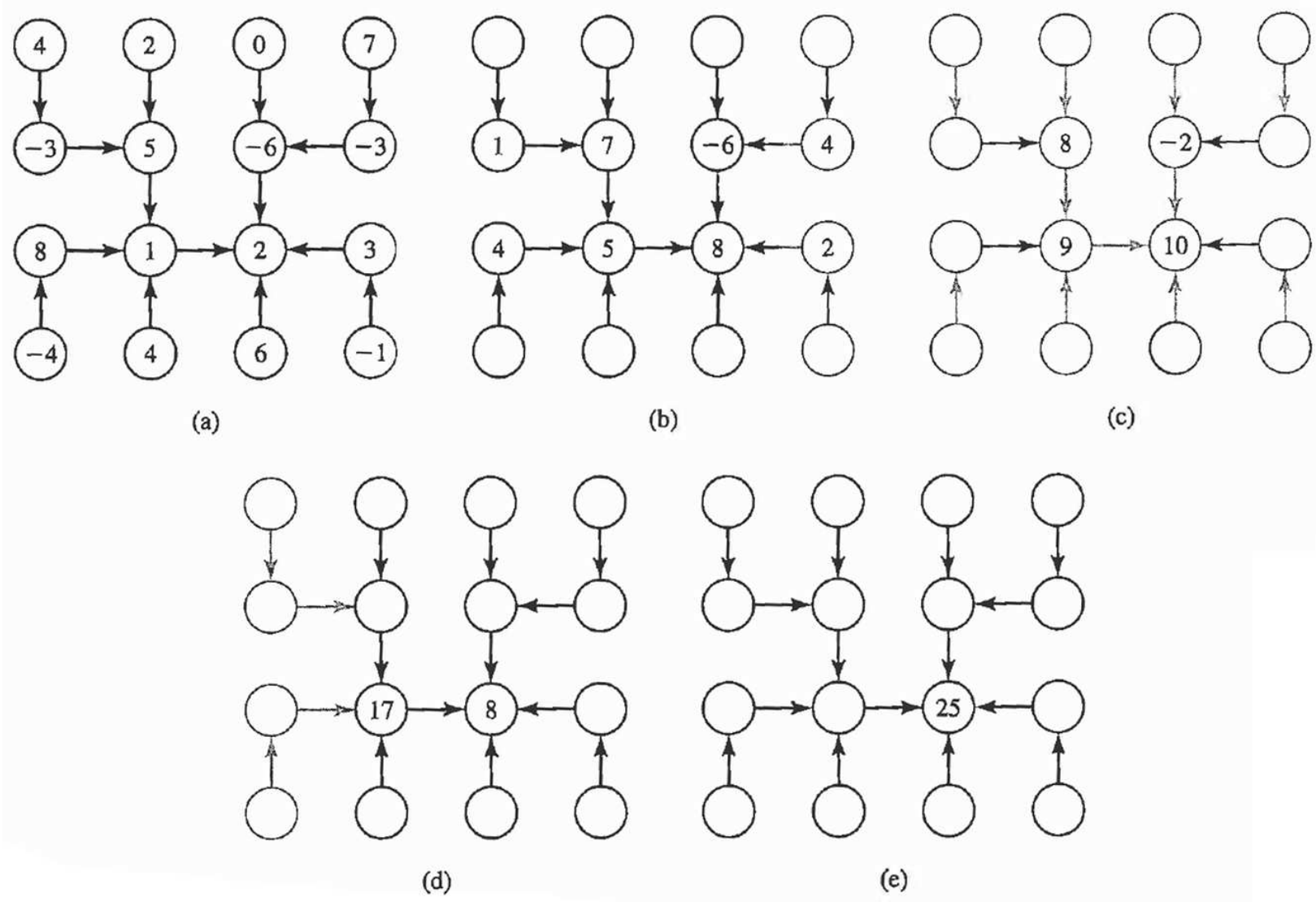
Example: parallel reduction

- Given a set of n values: a_0, a_1, a_2, ..., a_{n-1}
- Given an associative binary operator $\oplus$
- **Reduction**: compute $a_0 \oplus a_1 \oplus a_2 \cdots \oplus a_{n-1}$
- On a sequential computer: $n - 1$ operations are needed
- How to implement a parallel reduction?

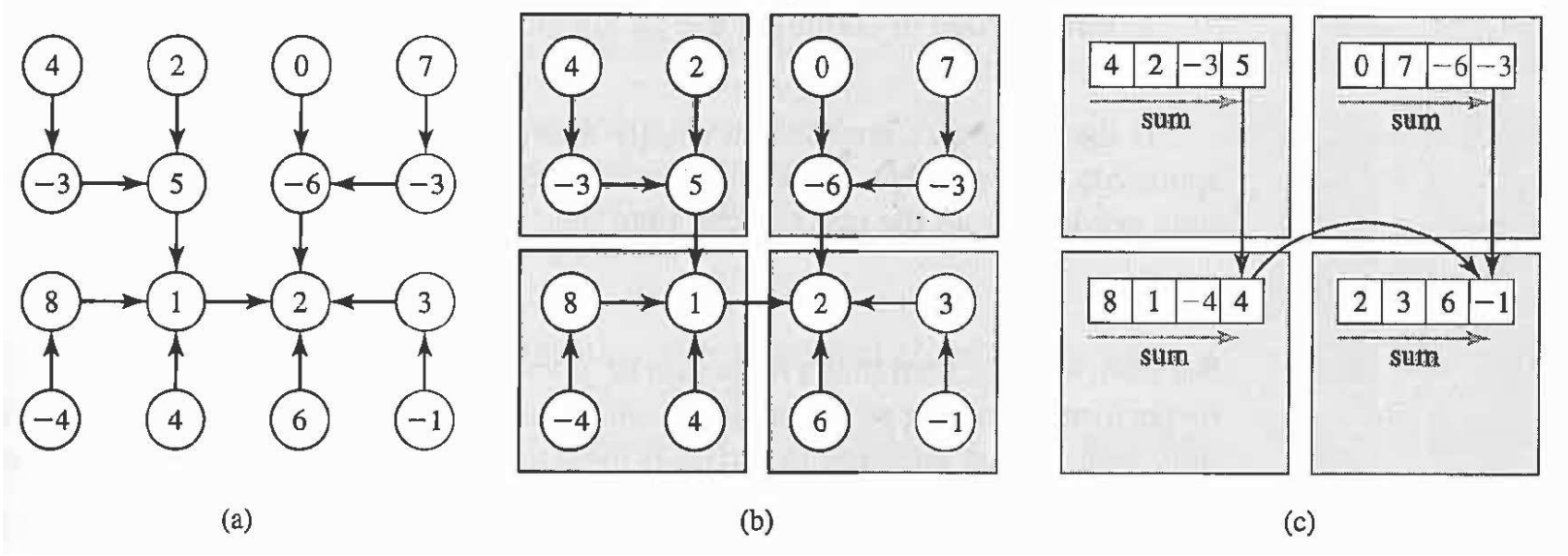
Parallel reduction (2)



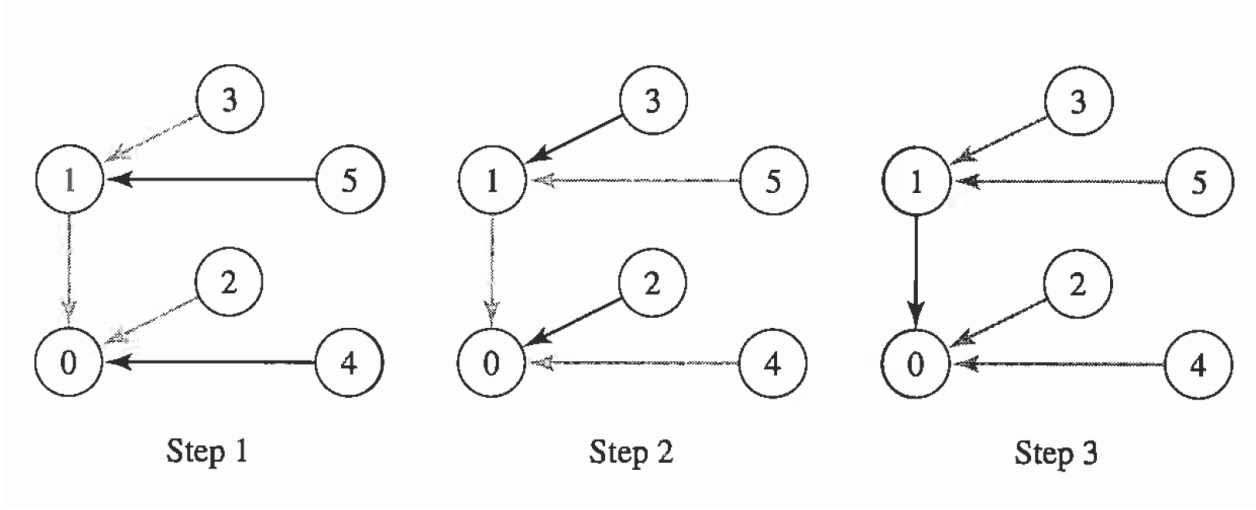
Finding global sum



Finding global sum (2)



Another example of parallel reduction



When the number of tasks is not a power of 2

Cost analysis of parallel reduction

- χ: time needed to perform the binary operation $\oplus$
- λ: time needed to communication a value from one task to another
- n: number of values
- p: number of processors
- If n is divided evenly among p processors
  - Time needed by each processor to treat its assigned values

$$(\lceil n/p \rceil - 1)\,\chi$$

  - $\lceil \log p \rceil$ communication steps are needed
  - Each communication step requires time $\lambda + \chi$
  - Total parallel computing time

$$(\lceil n/p \rceil - 1)\,\chi + \lceil \log p \rceil (\lambda + \chi)$$

Concluding remarks

- The task/channel model encourages parallel algorithm designs that maximize local computations and minimize communications
- The algorithm designer typically partitions the computation, identifies communications among primitive tasks, agglomerates primitive tasks into larger tasks, and decides how to map tasks to processors
- The goals are to maximize processor utilization and minimize interprocessor communications
- Good designs must often strike a balance between the above two goals

Real-life example of parallel processing