

# Parallel Programming in C with MPI and OpenMP

Michael J. Quinn



# Chapter 3

## Parallel Algorithm Design

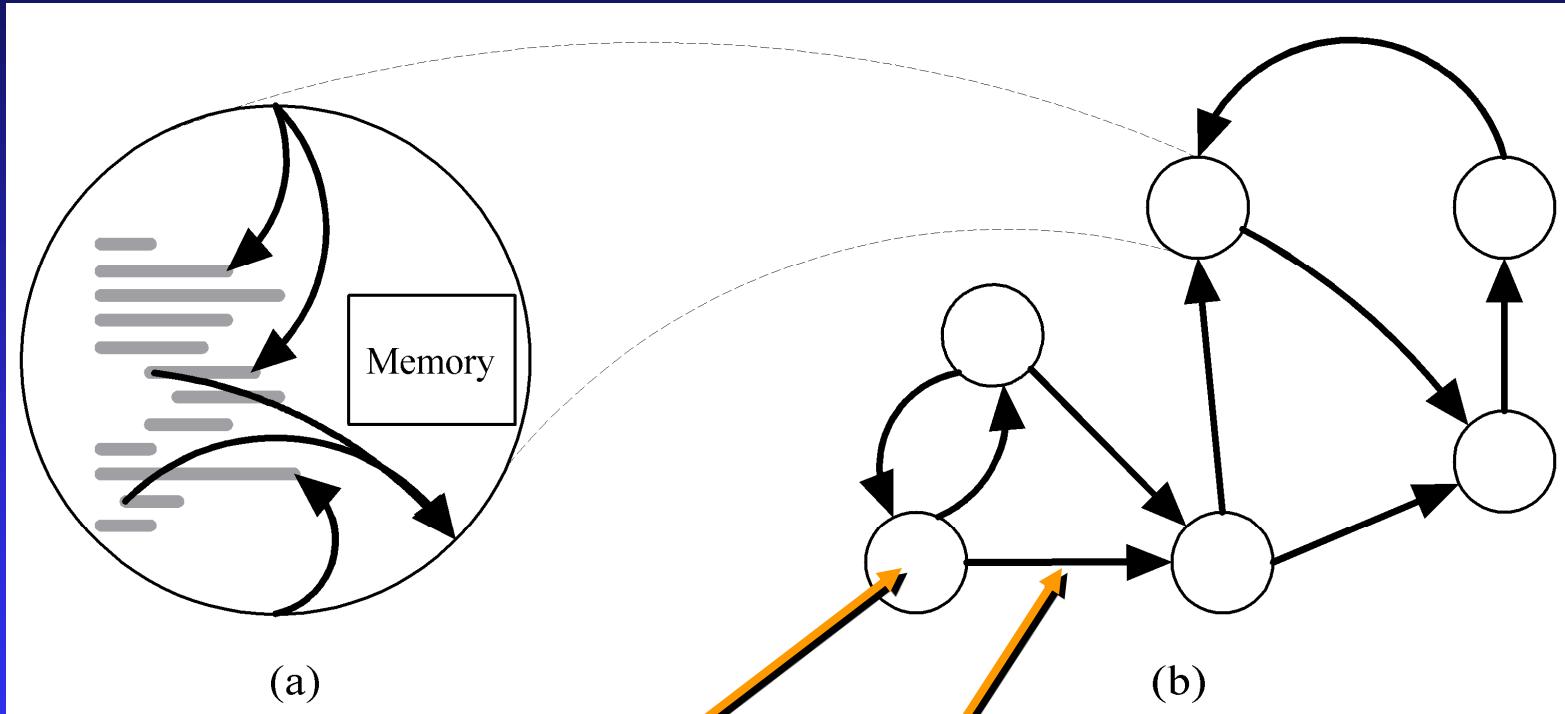
# Outline

- Task/channel model
- Algorithm design methodology
- Case studies

# Task/Channel Model

- Parallel computation = set of tasks
- Task
  - ◆ Program
  - ◆ Local memory
  - ◆ Collection of I/O ports
- Tasks interact by sending messages through channels

# Task/Channel Model

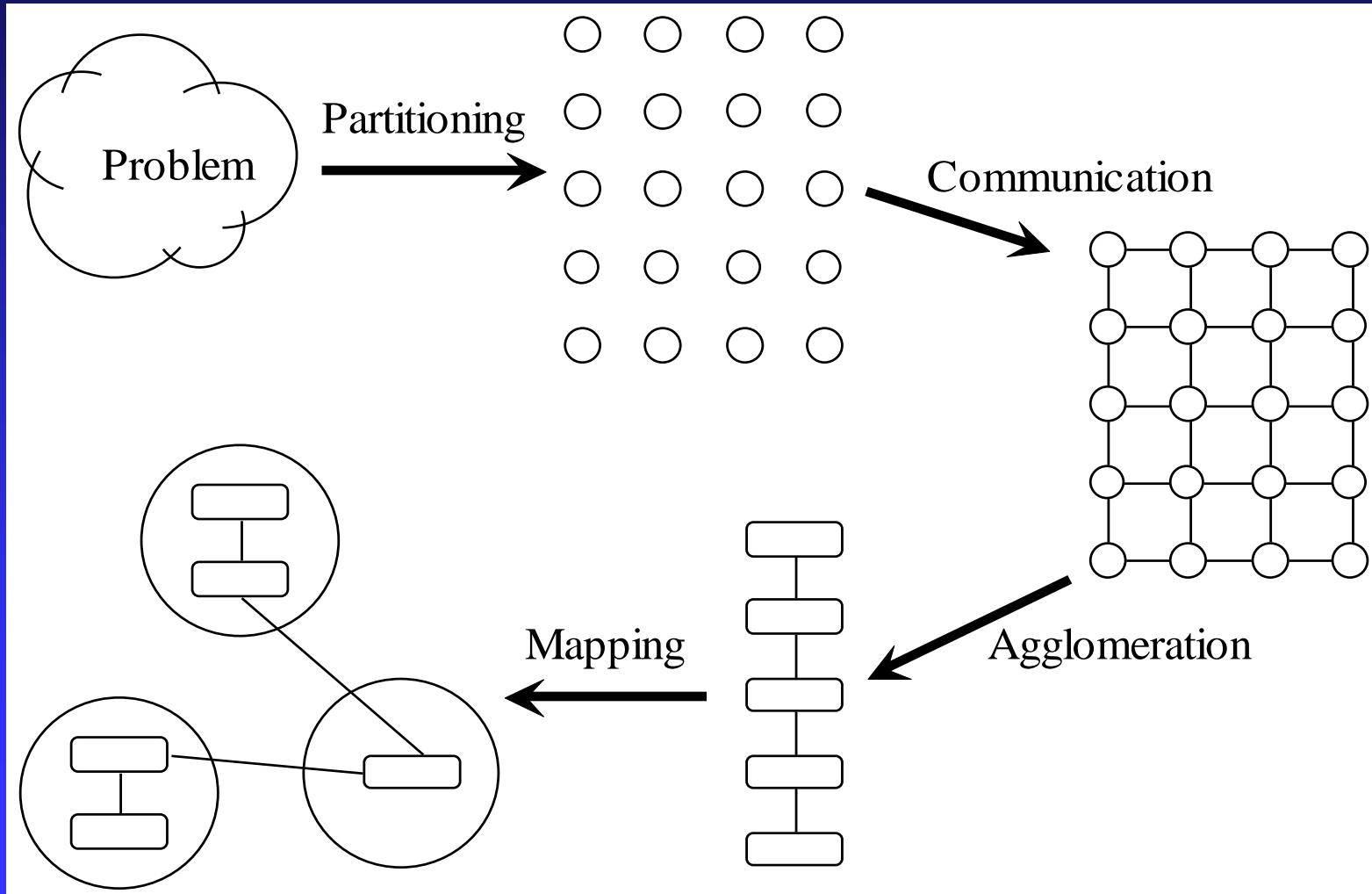


Task  
Channel

# Foster's Design Methodology

- Partitioning
- Communication
- Agglomeration
- Mapping

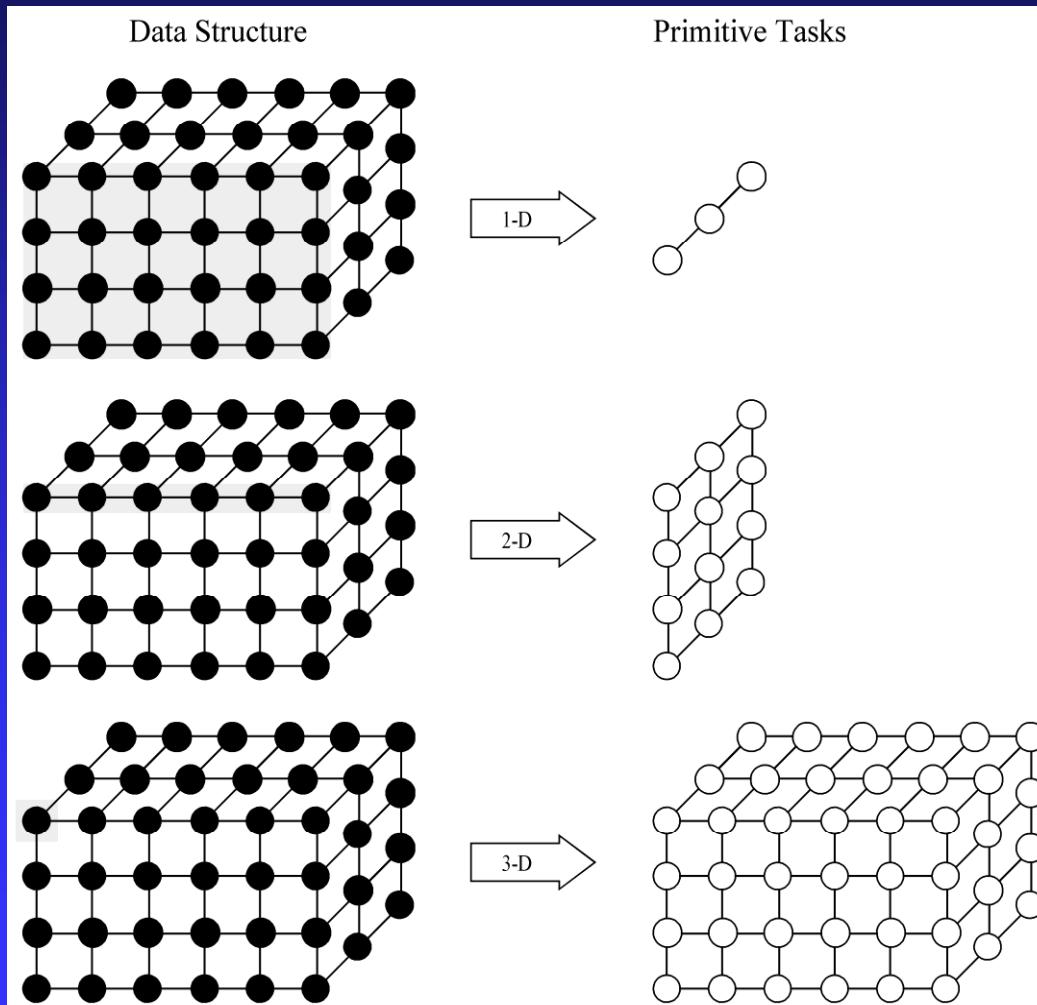
# Foster's Methodology



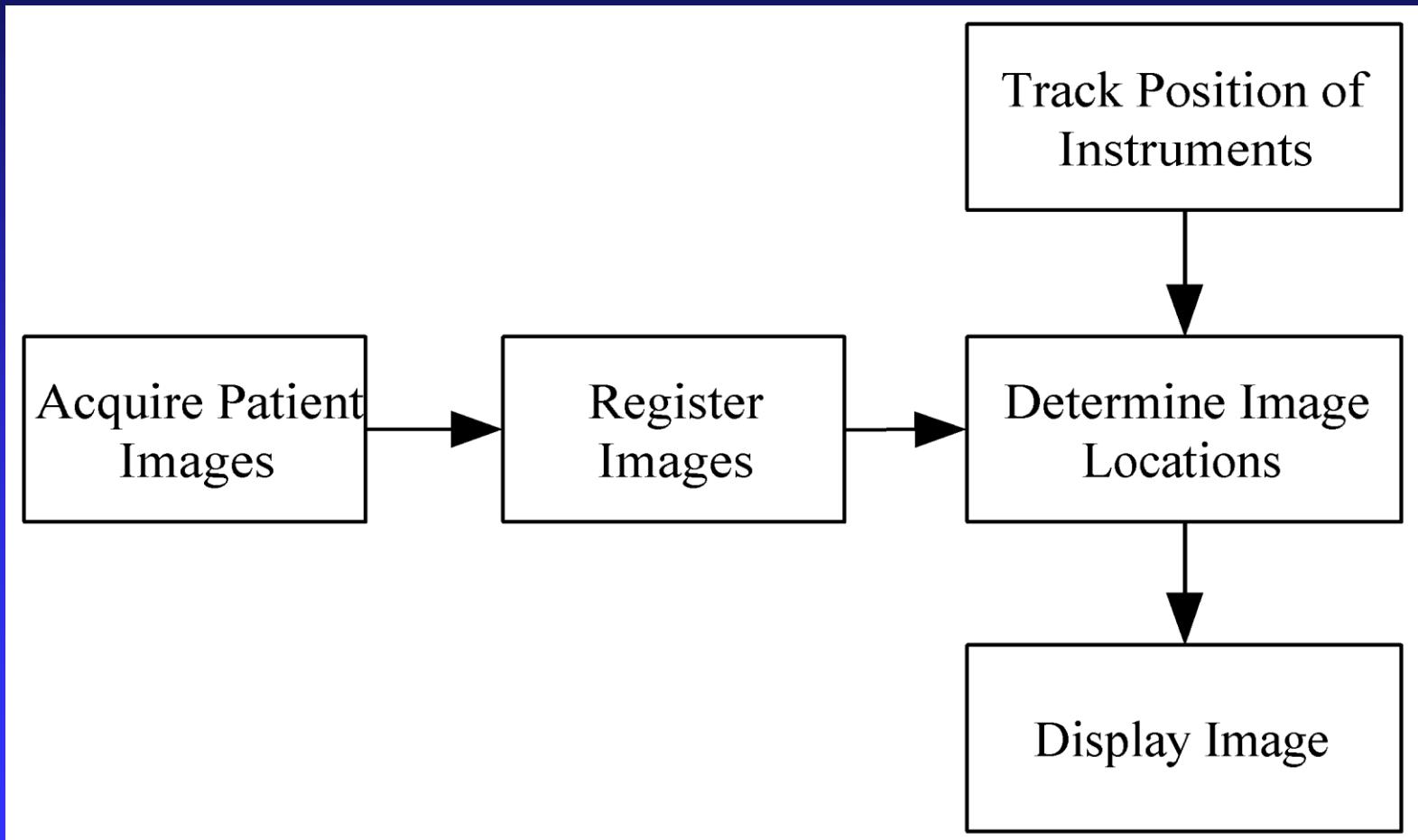
# Partitioning

- Dividing computation and data into pieces
- Domain decomposition
  - ◆ Divide data into pieces
  - ◆ Determine how to associate computations with the data
- Functional decomposition
  - ◆ Divide computation into pieces
  - ◆ Determine how to associate data with the computations

# Example Domain Decompositions



# Example Functional Decomposition



# Partitioning Checklist

- At least 10x more primitive tasks than processors in target computer
- Minimize redundant computations and redundant data storage
- Primitive tasks roughly the same size
- Number of tasks an increasing function of problem size

# Communication

- Determine values passed among tasks
- Local communication
  - ◆ Task needs values from a small number of other tasks
  - ◆ Create channels illustrating data flow
- Global communication
  - ◆ Significant number of tasks contribute data to perform a computation
  - ◆ Don't create channels for them early in design

# Communication Checklist

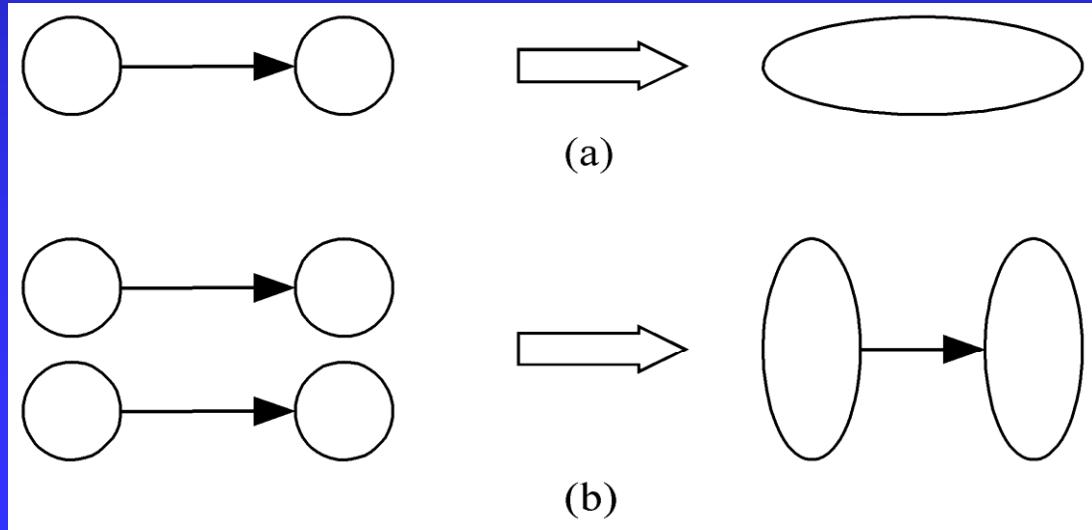
- Communication operations balanced among tasks
- Each task communicates with only small group of neighbors
- Tasks can perform communications concurrently
- Task can perform computations concurrently

# Agglomeration

- Grouping tasks into larger tasks
- Goals
  - ◆ Improve performance
  - ◆ Maintain scalability of program
  - ◆ Simplify programming
- In MPI programming, goal often to create one agglomerated task per processor

# Agglomeration Can Improve Performance

- Eliminate communication between primitive tasks agglomerated into consolidated task
- Combine groups of sending and receiving tasks



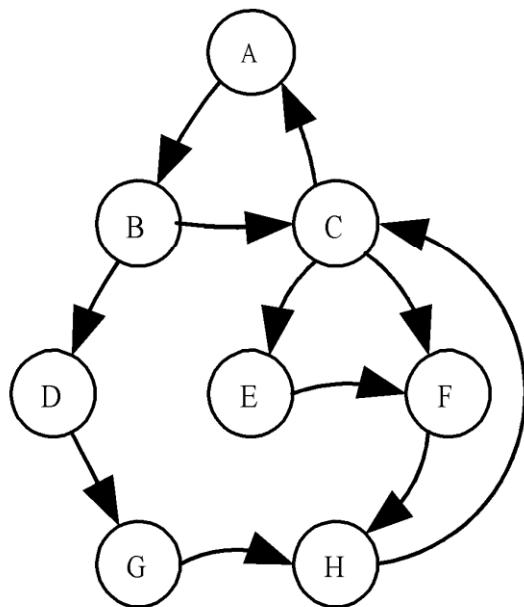
# Agglomeration Checklist

- Locality of parallel algorithm has increased
- Replicated computations take less time than communications they replace
- Data replication doesn't affect scalability
- Agglomerated tasks have similar computational and communications costs
- Number of tasks increases with problem size
- Number of tasks suitable for likely target systems
- Tradeoff between agglomeration and code modifications costs is reasonable

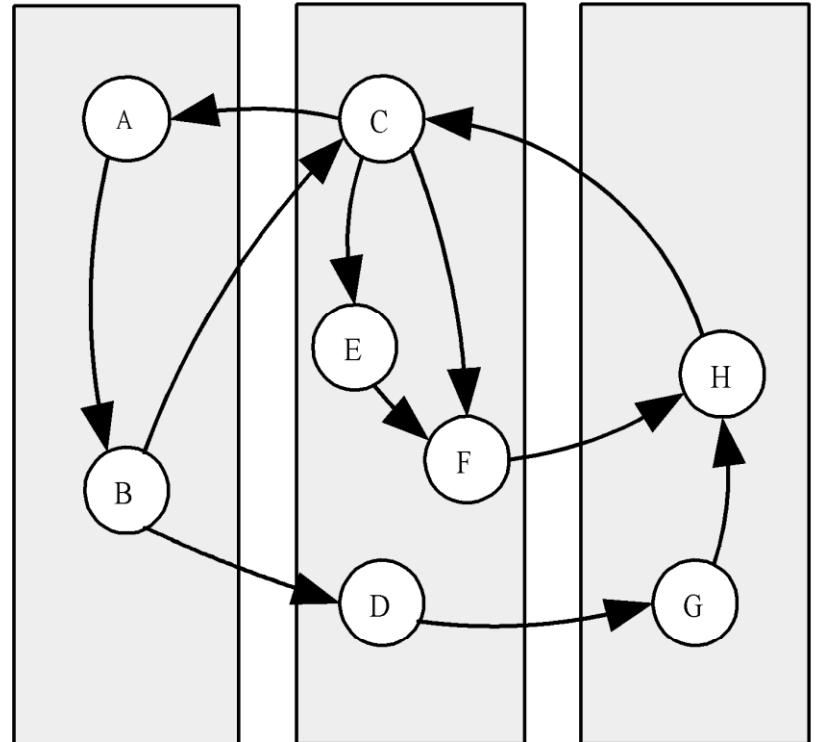
# Mapping

- Process of assigning tasks to processors
- Centralized multiprocessor: mapping done by operating system
- Distributed memory system: mapping done by user
- Conflicting goals of mapping
  - ◆ Maximize processor utilization
  - ◆ Minimize interprocessor communication

# Mapping Example



(a)



(b)

# Optimal Mapping

- Finding optimal mapping is NP-hard
- Must rely on heuristics

# Mapping Decision Tree

- Static number of tasks
  - ◆ Structured communication
    - ◆ Constant computation time per task
      - Agglomerate tasks to minimize comm
      - Create one task per processor
    - ◆ Variable computation time per task
      - Cyclically map tasks to processors
  - ◆ Unstructured communication
    - Use a static load balancing algorithm
- Dynamic number of tasks

# Mapping Strategy

- Static number of tasks
- Dynamic number of tasks
  - ◆ Frequent communications between tasks
    - ◆ Use a dynamic load balancing algorithm
  - ◆ Many short-lived tasks
    - ◆ Use a run-time task-scheduling algorithm

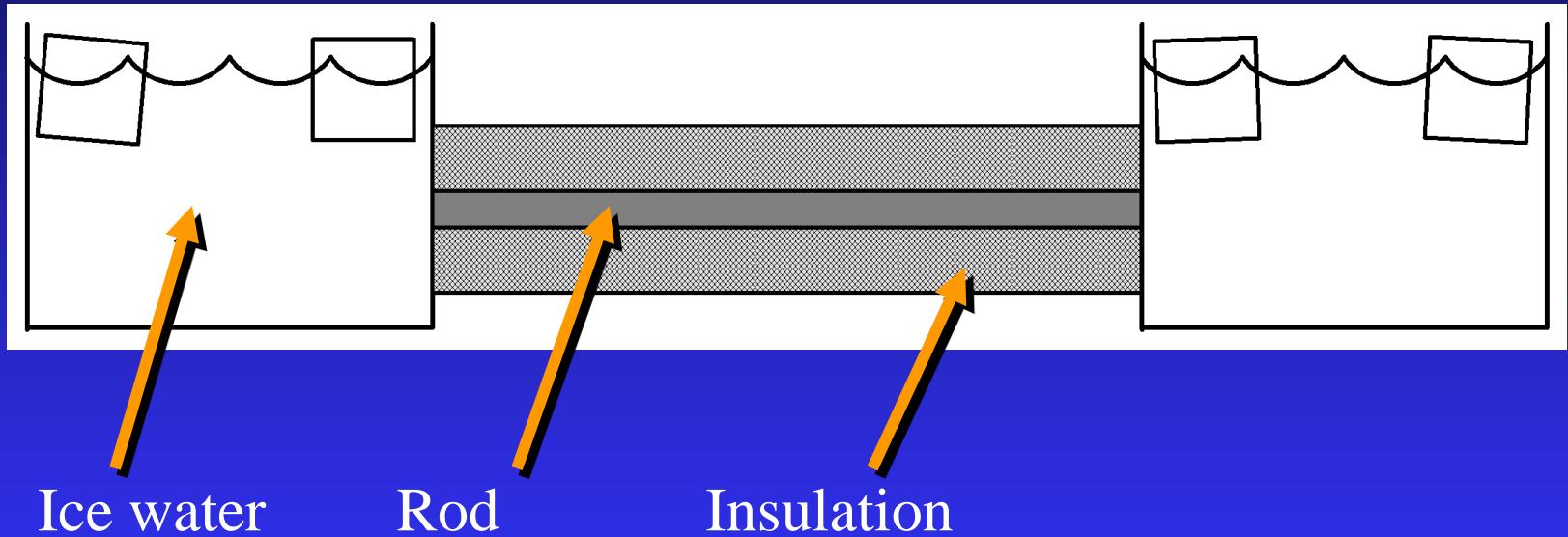
# Mapping Checklist

- Considered designs based on one task per processor and multiple tasks per processor
- Evaluated static and dynamic task allocation
- If dynamic task allocation chosen, task allocator is not a bottleneck to performance
- If static task allocation chosen, ratio of tasks to processors is at least 10:1

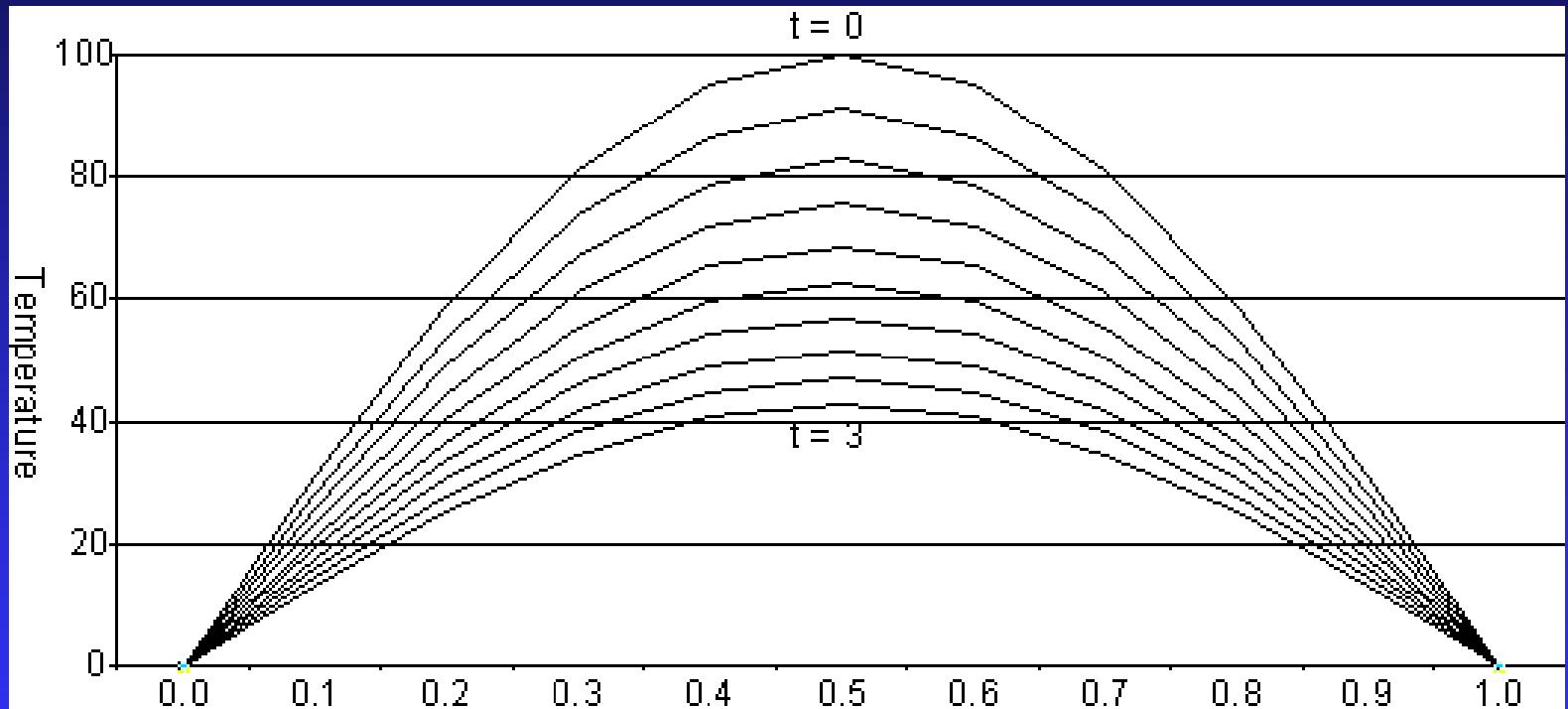
# Case Studies

- Boundary value problem
- Finding the maximum
- The n-body problem
- Adding data input

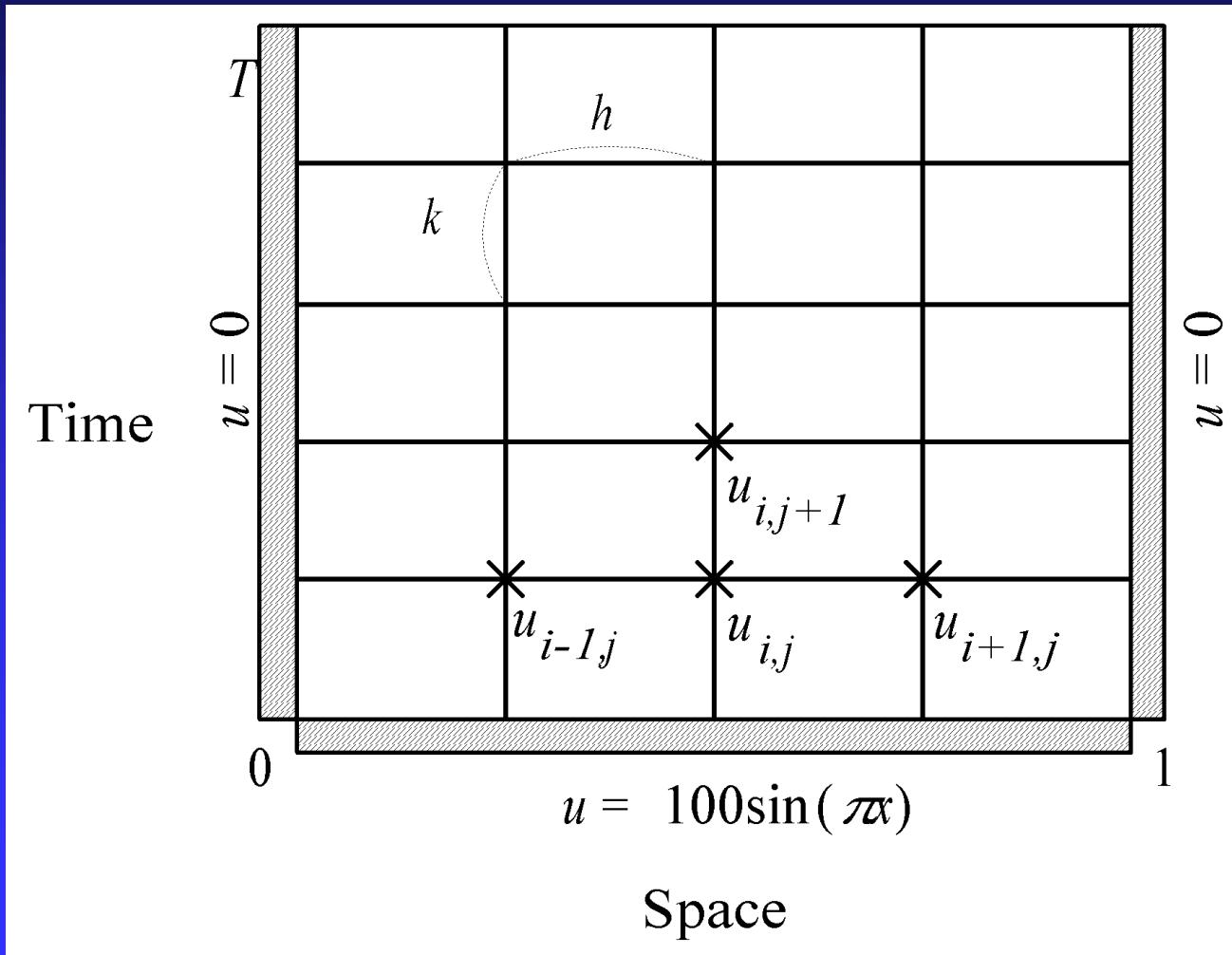
# Boundary Value Problem



# Rod Cools as Time Progresses



# Finite Difference Approximation



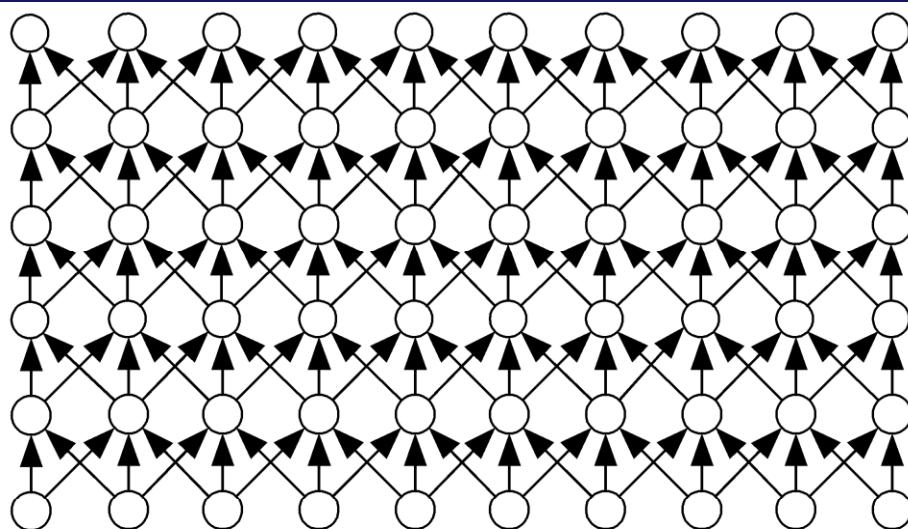
# Partitioning

- One data item per grid point
- Associate one primitive task with each grid point
- Two-dimensional domain decomposition

# Communication

- Identify communication pattern between primitive tasks
- Each interior primitive task has three incoming and three outgoing channels

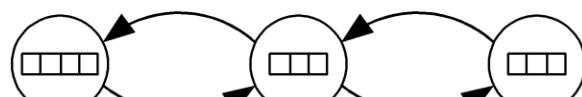
# Agglomeration and Mapping



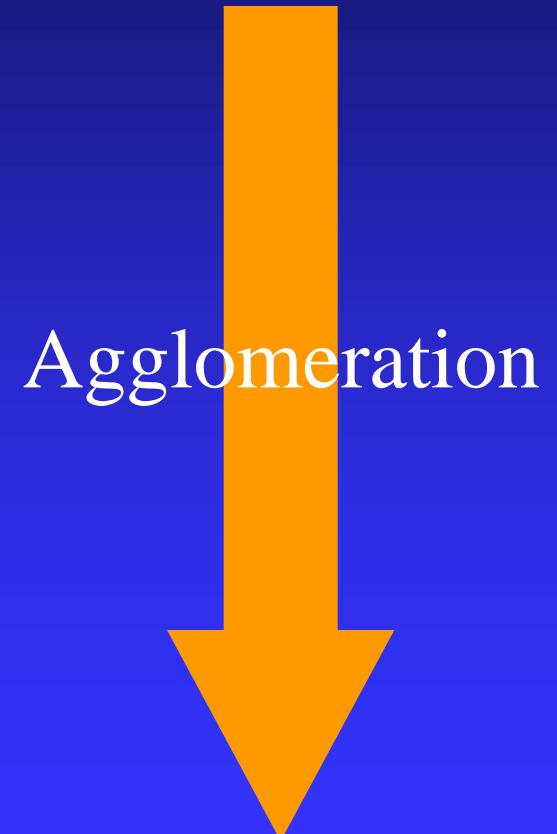
(a)



(b)



(c)



# Sequential execution time

- $\chi$  – time to update element
- $n$  – number of elements
- $m$  – number of iterations
- Sequential execution time:  $m(n-1)\chi$

# Parallel Execution Time

- $p$  – number of processors
- $\lambda$  – message latency
- Parallel execution time  $m(\chi \lceil (n-1)/p \rceil + 2\lambda)$

# Finding the Maximum Error

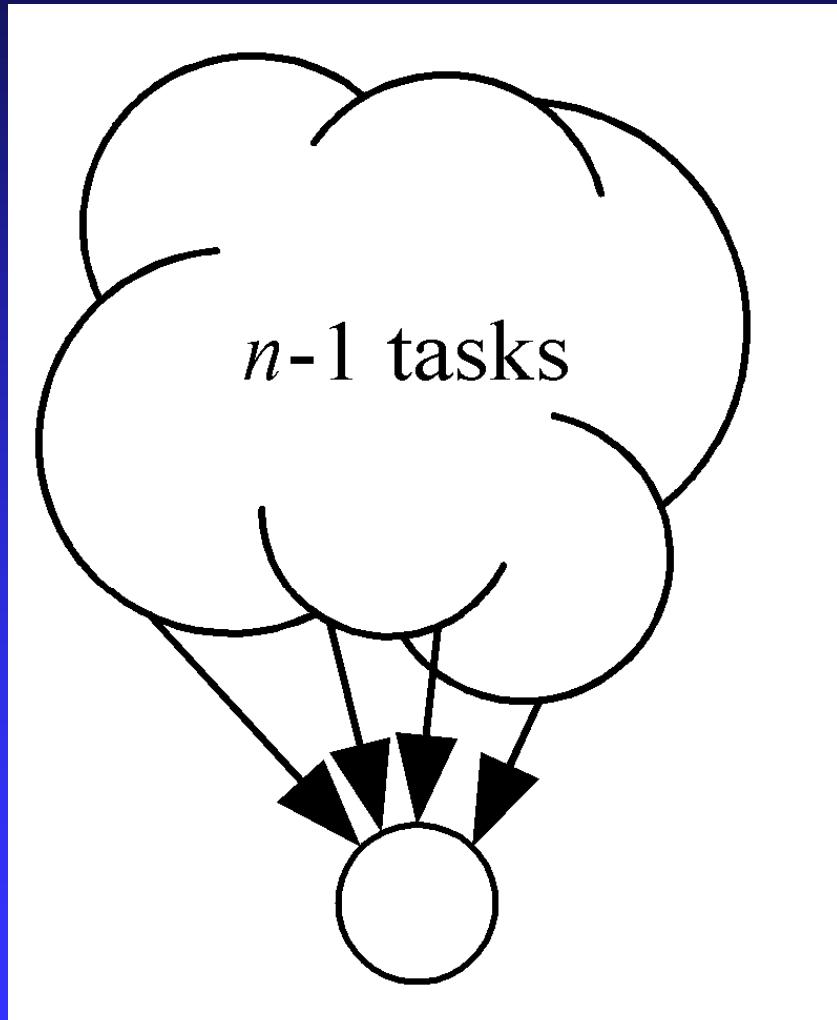
Computed	0.15	0.16	0.16	0.19
Correct	0.15	0.16	0.17	0.18
Error (%)	0.00%	0.00%	6.25%	5.26%

6.25%

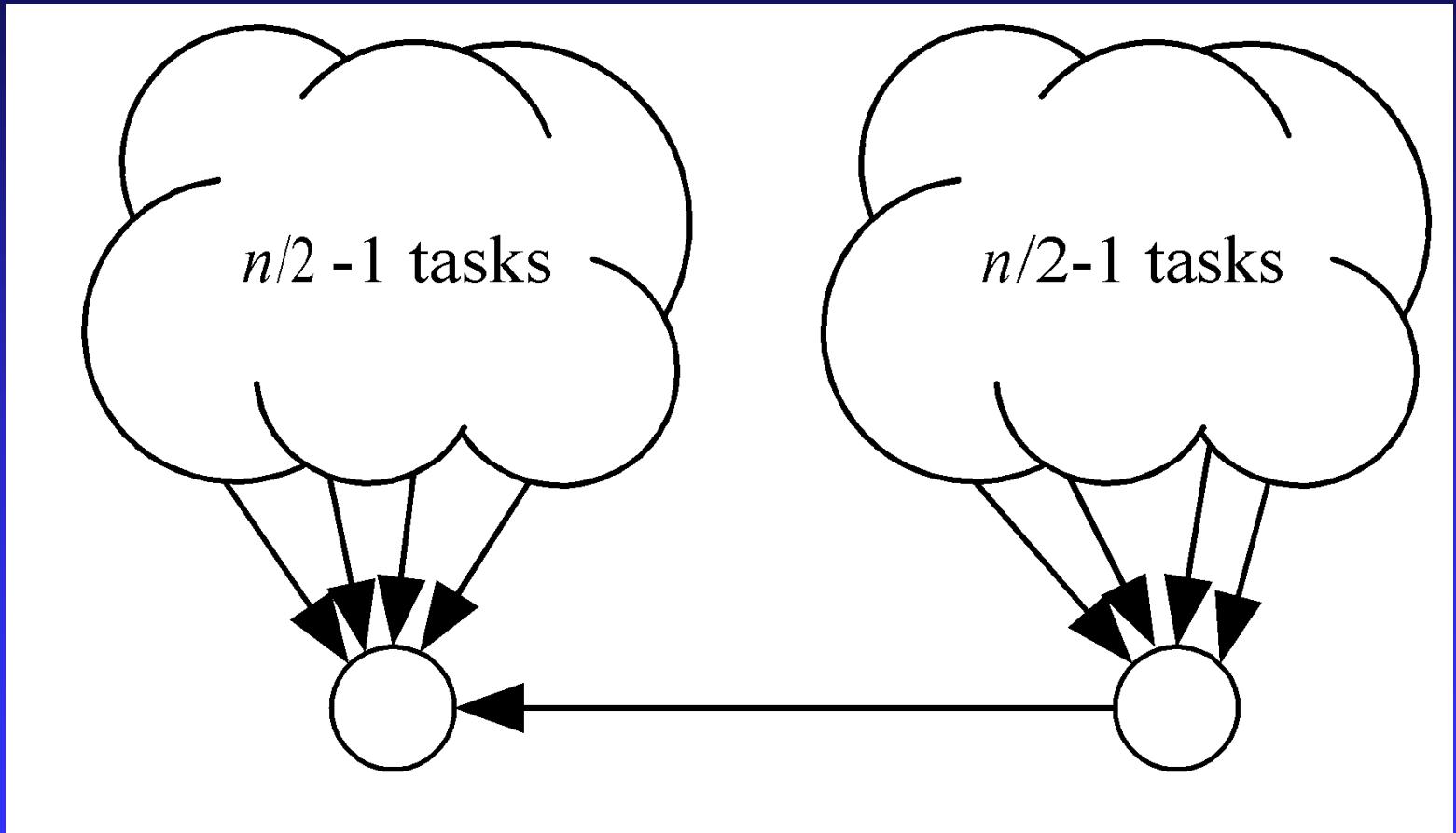
# Reduction

- Given associative operator  $\oplus$
- $a_0 \oplus a_1 \oplus a_2 \oplus \dots \oplus a_{n-1}$
- Examples
  - ◆ Add
  - ◆ Multiply
  - ◆ And, Or
  - ◆ Maximum, Minimum

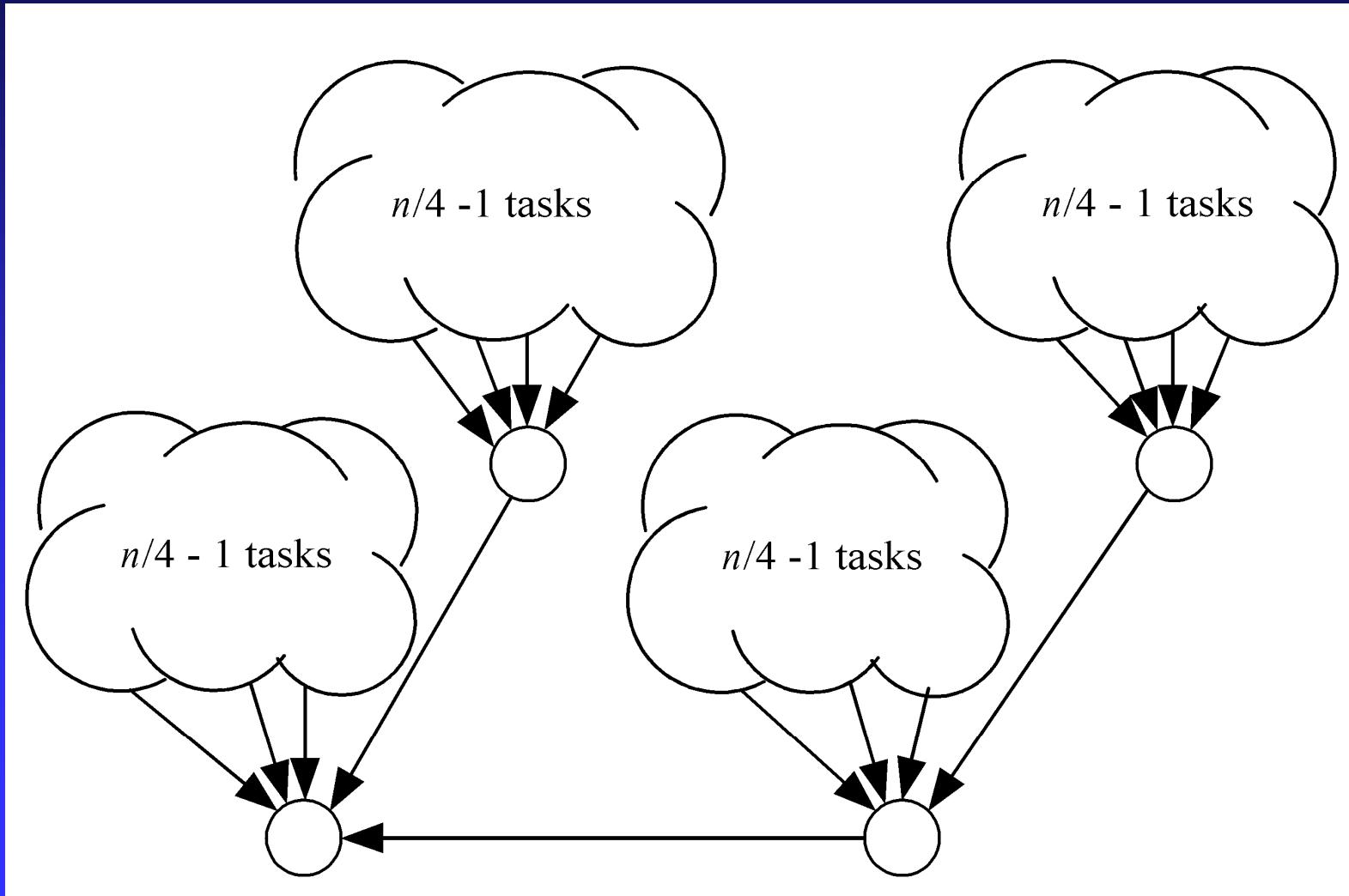
# Parallel Reduction Evolution



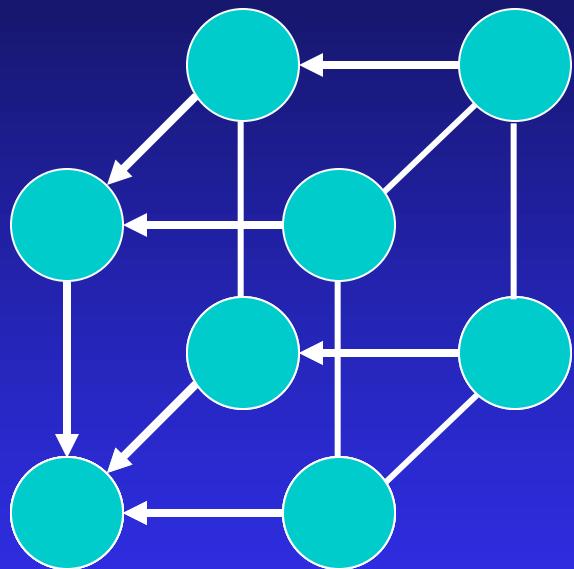
# Parallel Reduction Evolution



# Parallel Reduction Evolution

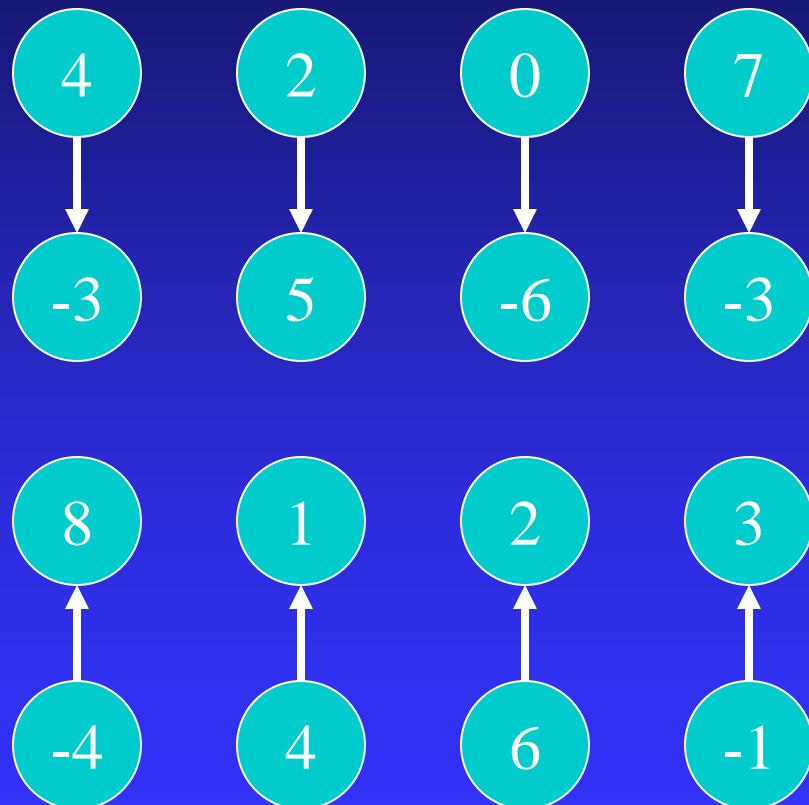


# Binomial Trees

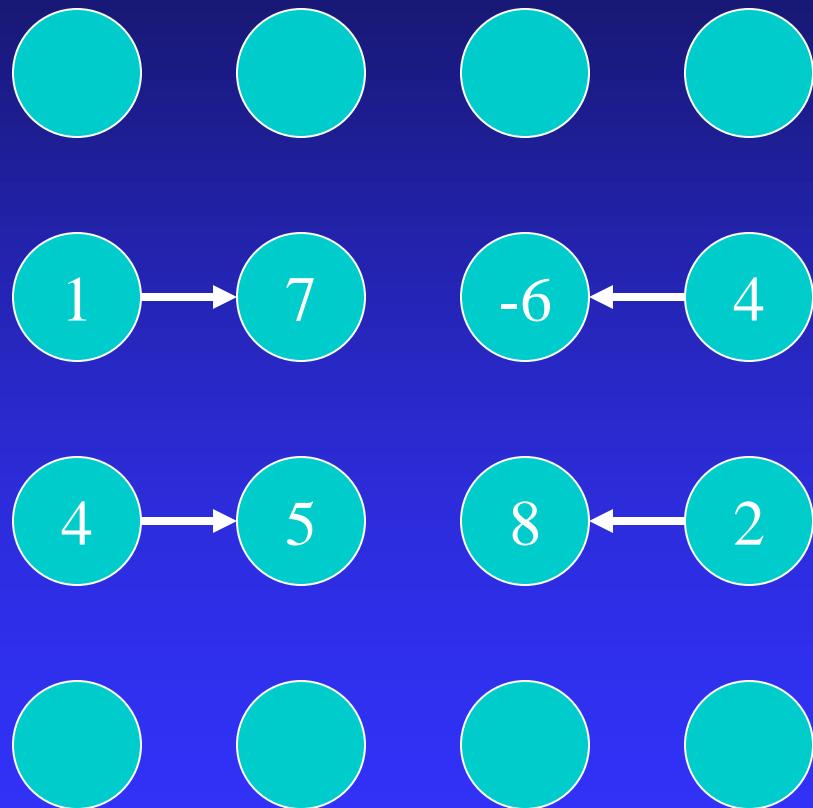


Subgraph of hypercube

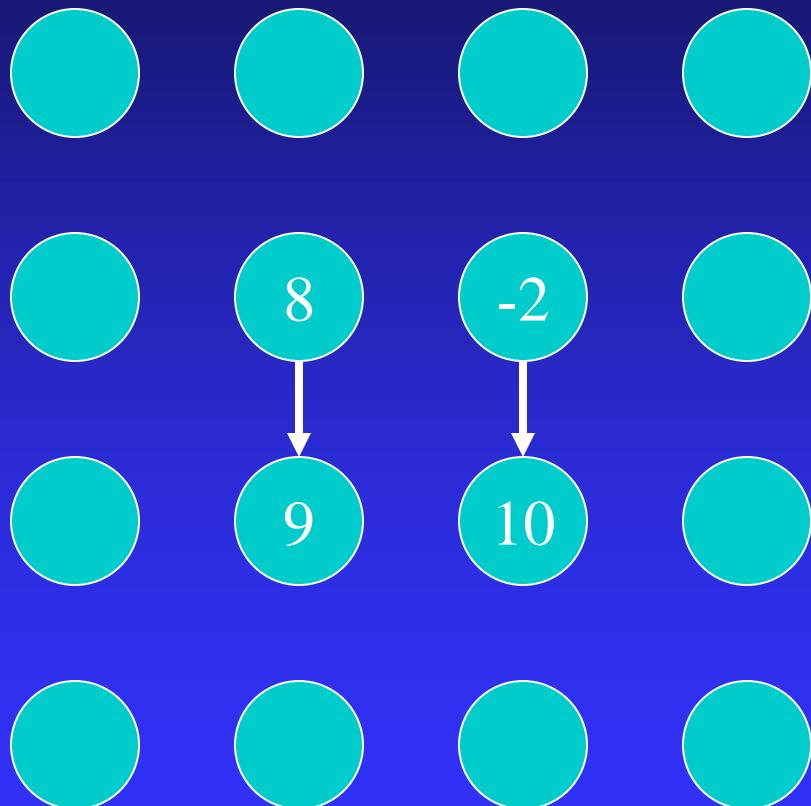
# Finding Global Sum



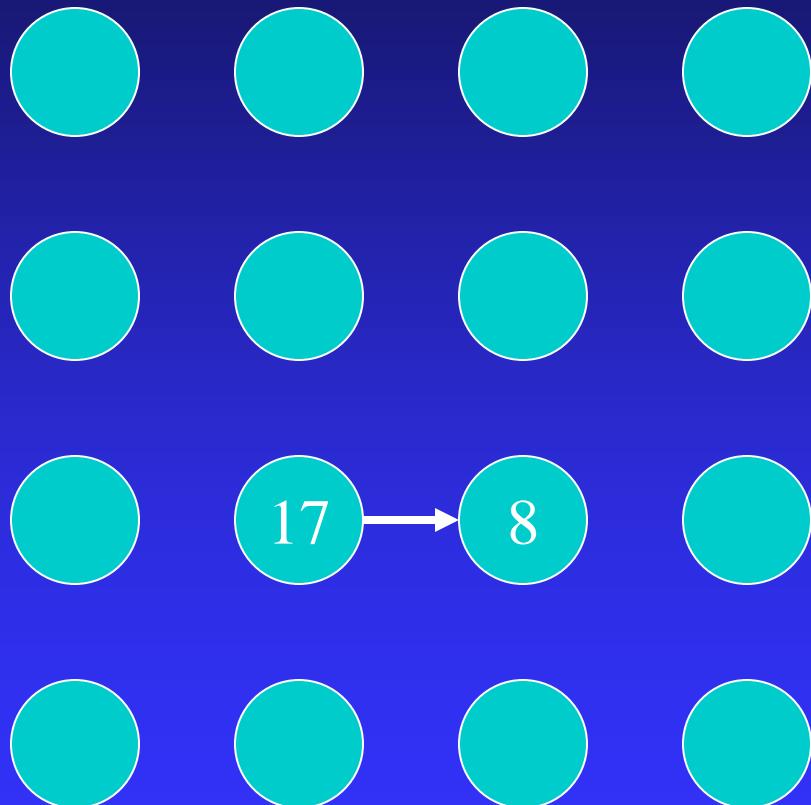
# Finding Global Sum



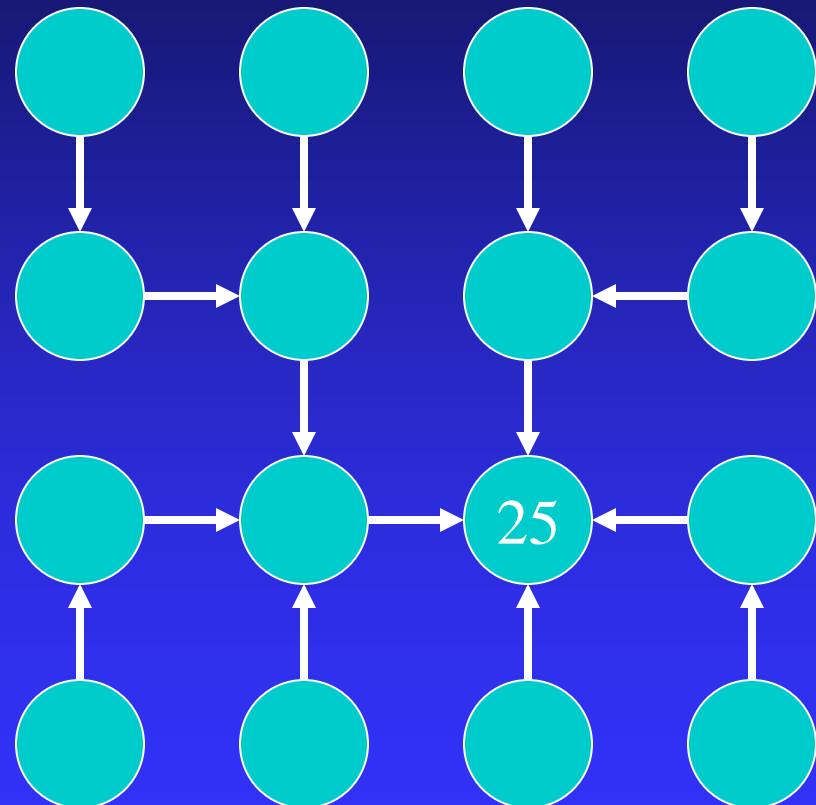
# Finding Global Sum



# Finding Global Sum

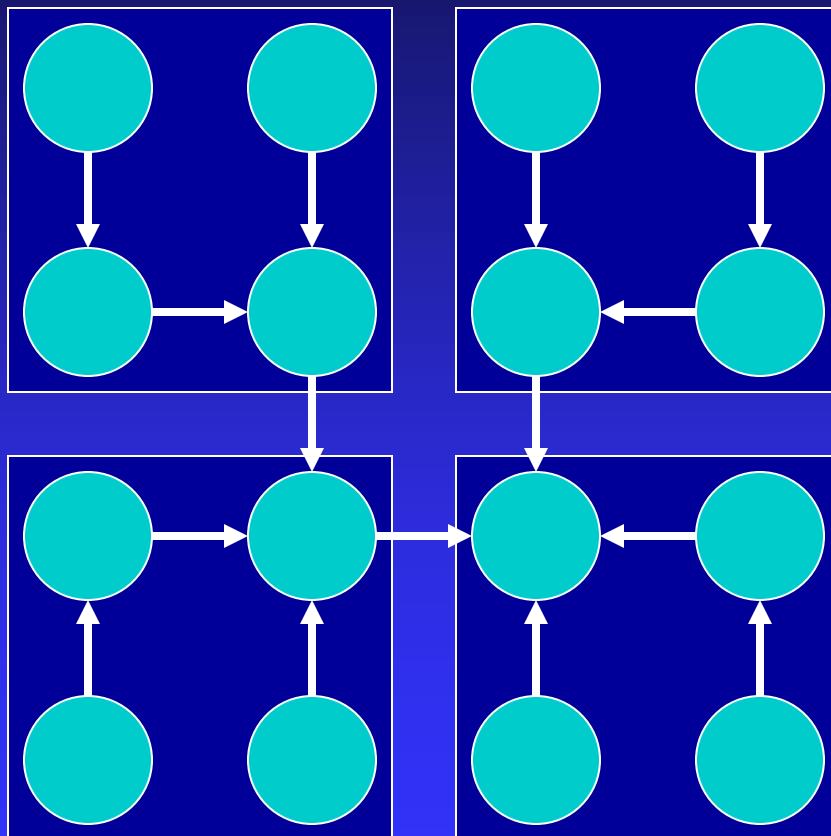


# Finding Global Sum

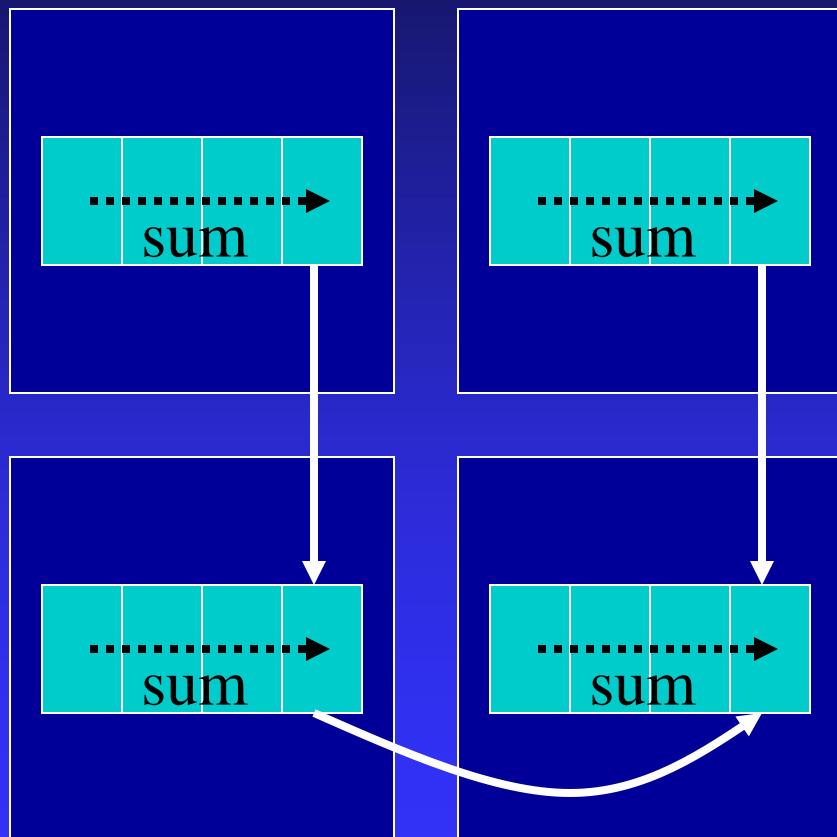


Binomial Tree

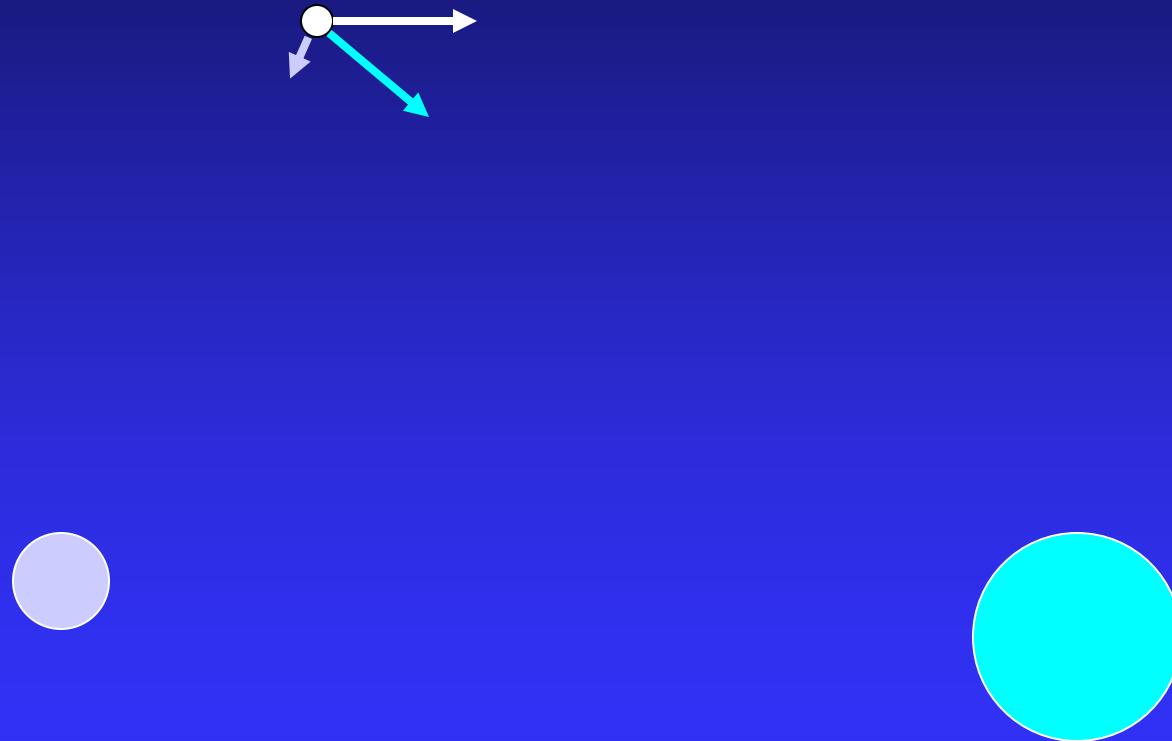
# Agglomeration



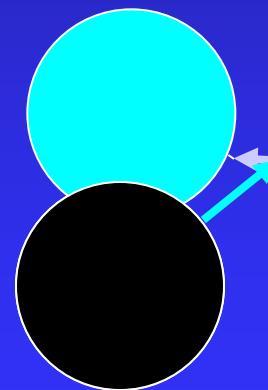
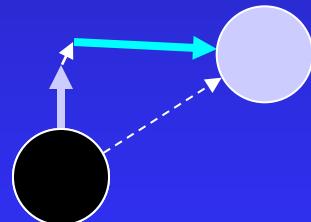
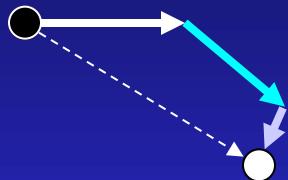
# Agglomeration



# The n-body Problem



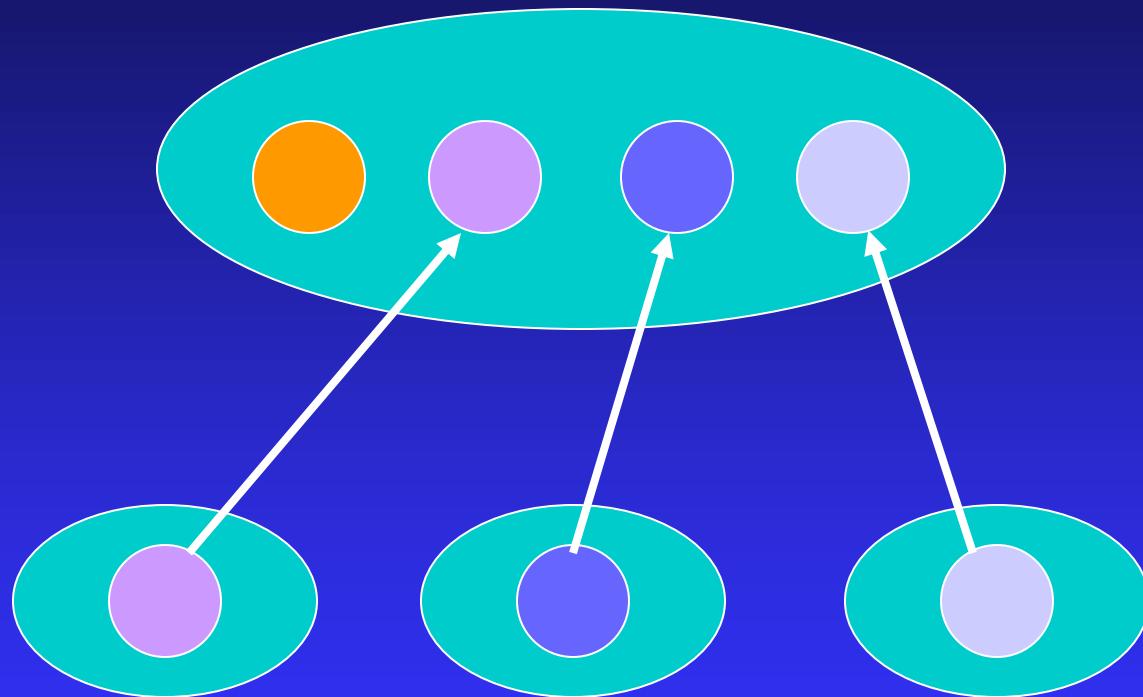
# The n-body Problem



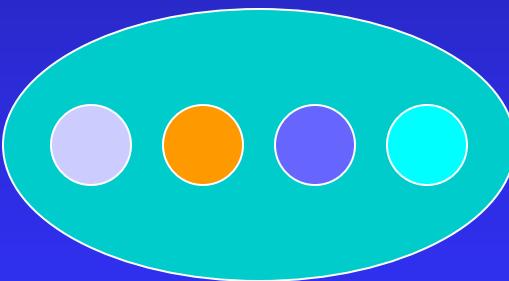
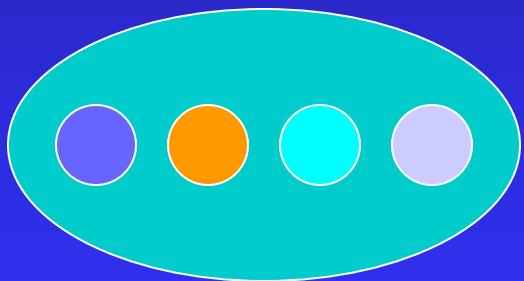
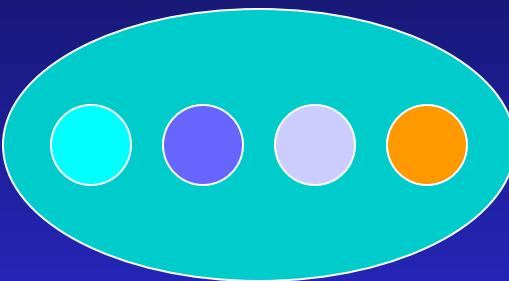
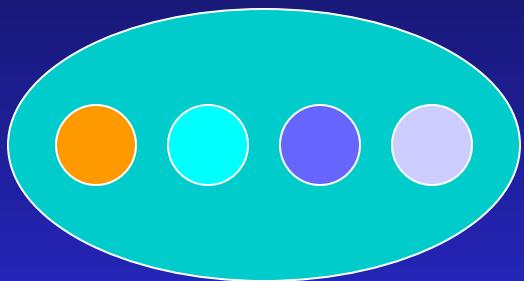
# Partitioning

- Domain partitioning
- Assume one task per particle
- Task has particle's position, velocity vector
- Iteration
  - ◆ Get positions of all other particles
  - ◆ Compute new position, velocity

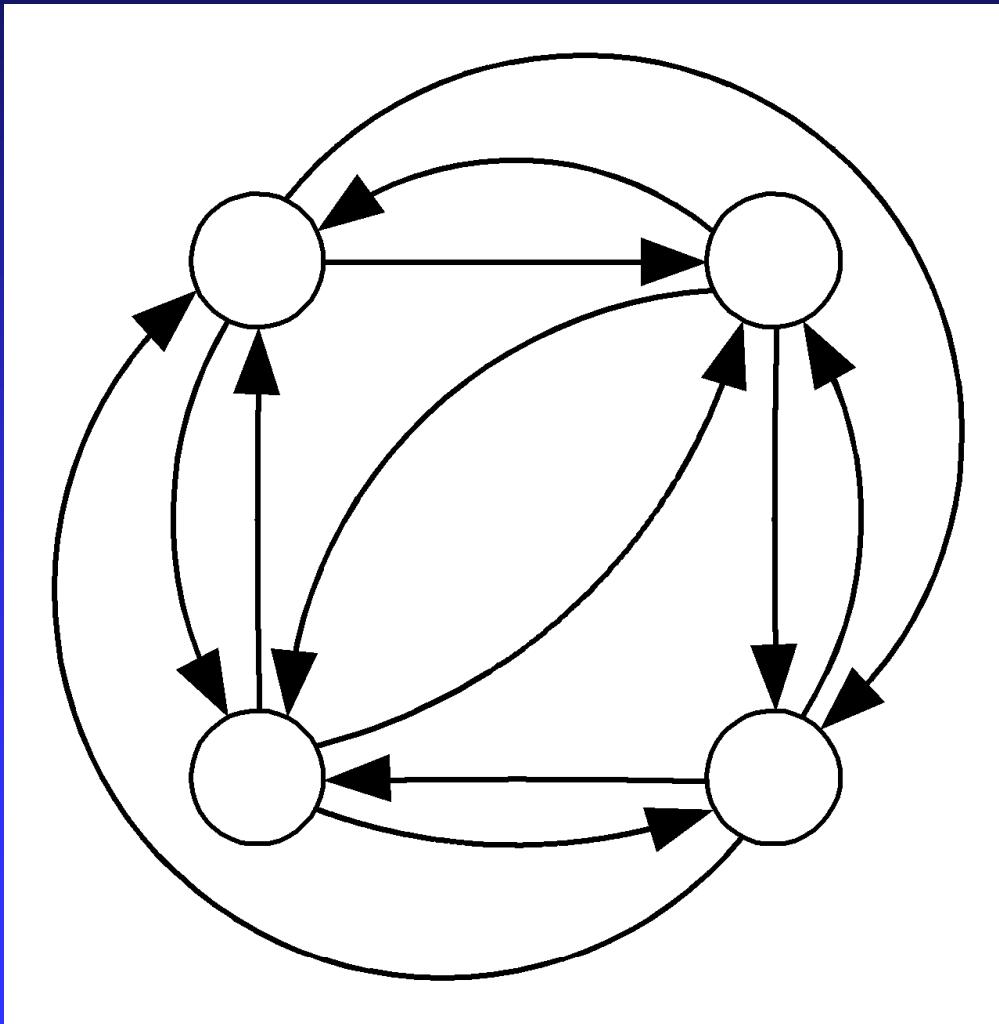
# Gather



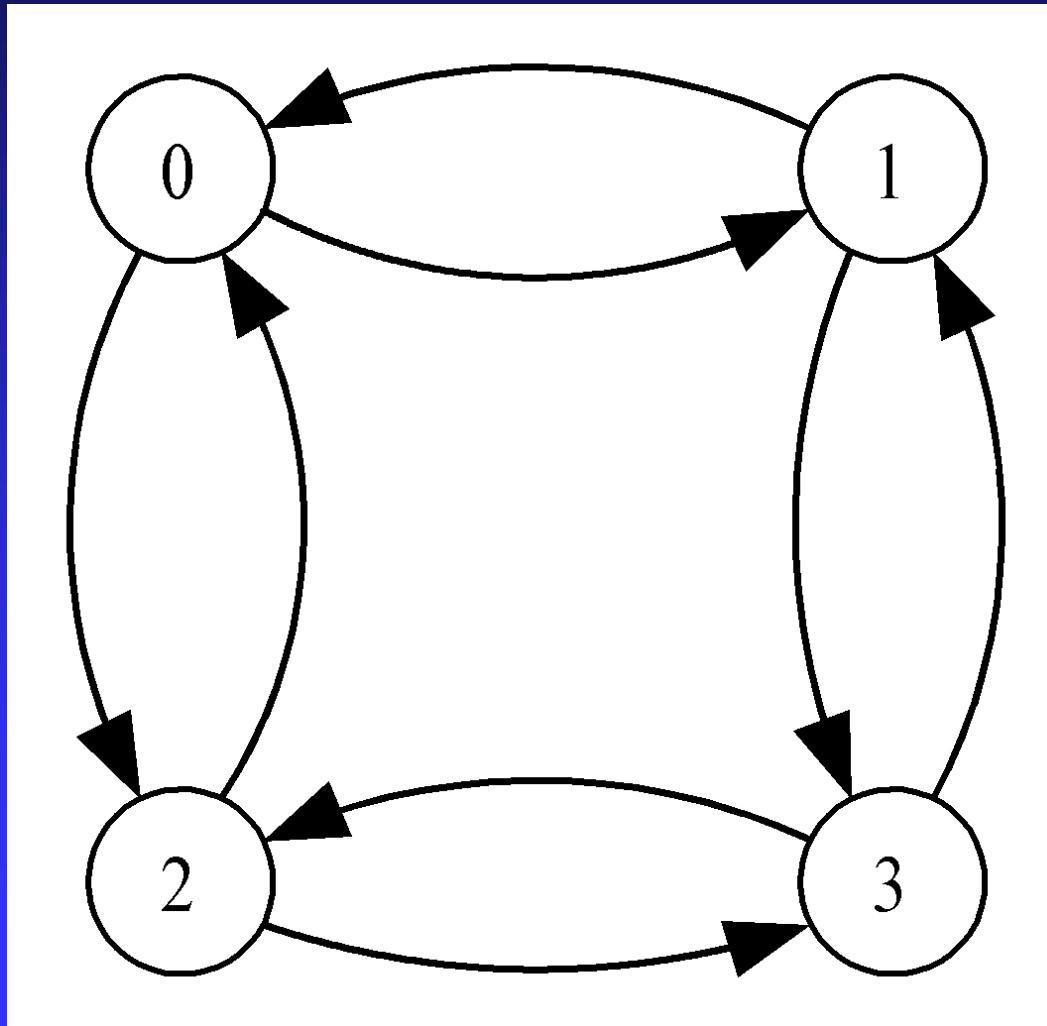
# All-gather



# Complete Graph for All-gather



# Hypercube for All-gather



# Communication Time

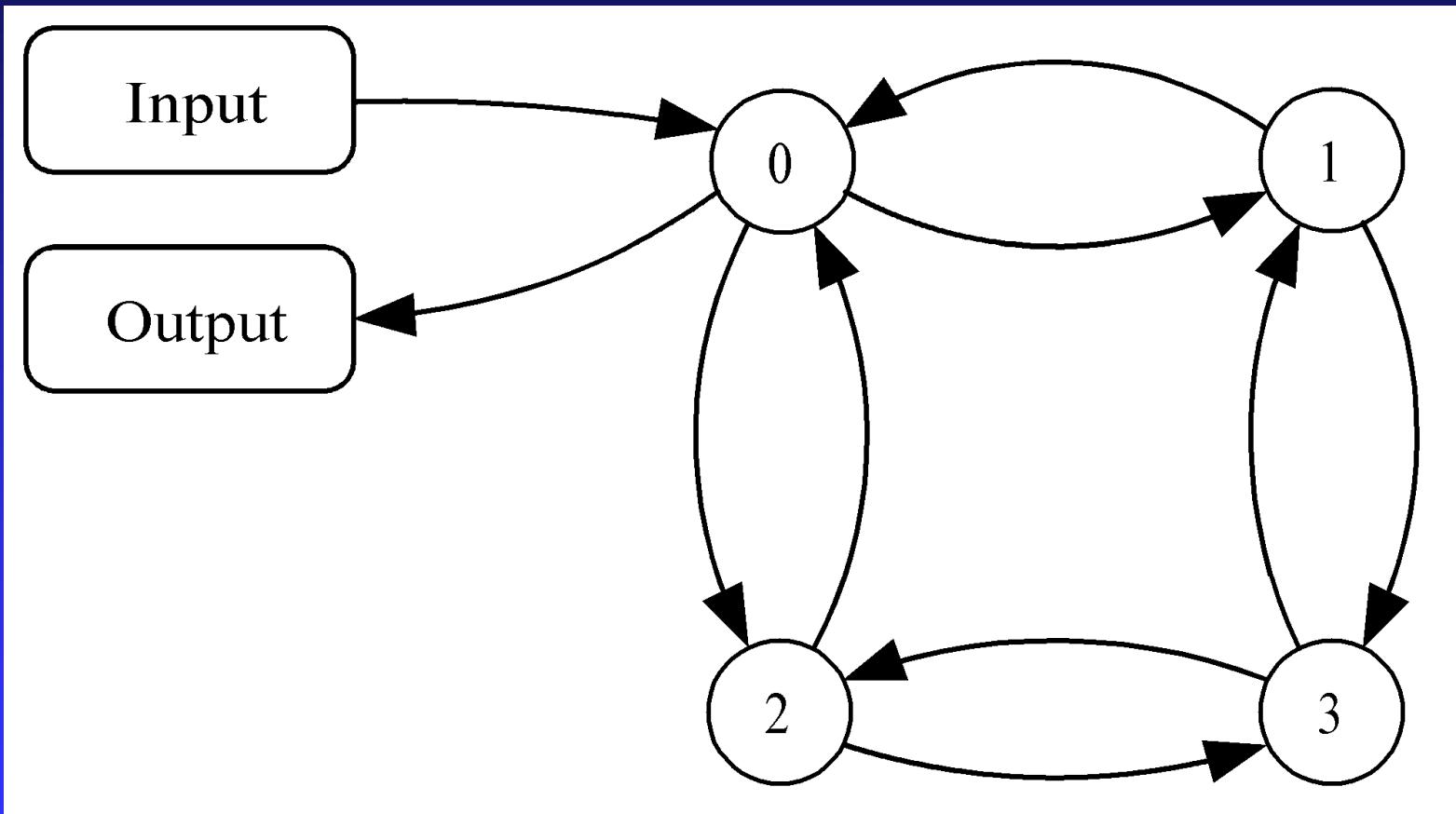
Complete graph

$$(p-1)(\lambda + \frac{n/p}{\beta}) = (p-1)\lambda + \frac{n(p-1)}{\beta p}$$

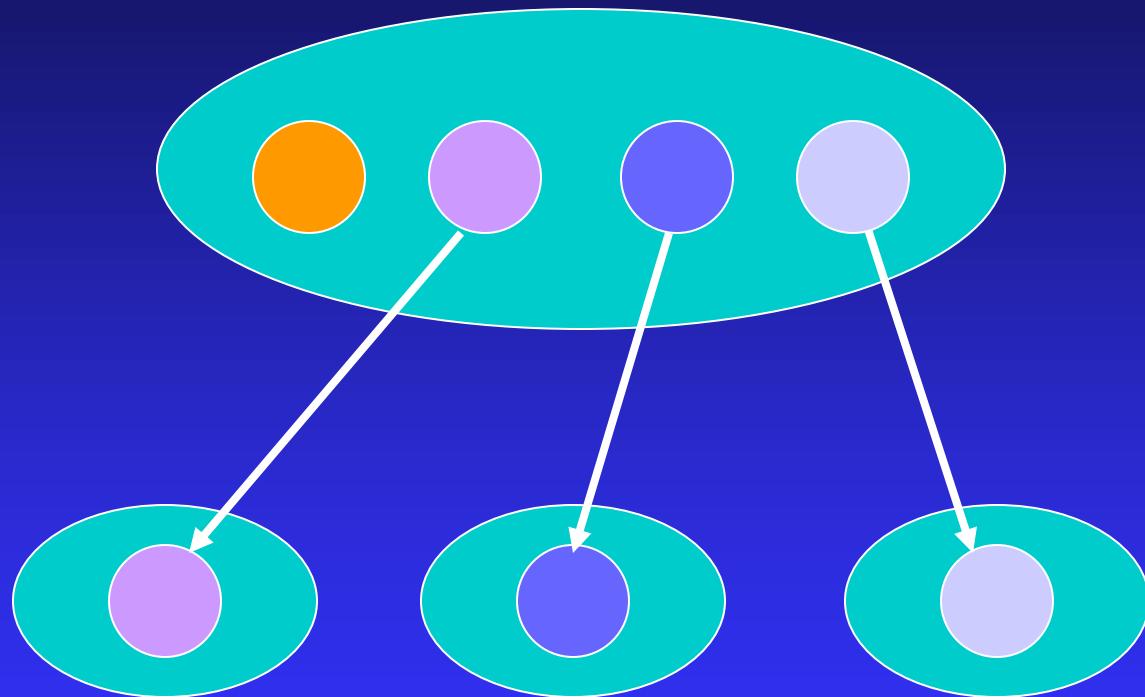
Hypercube

$$\sum_{i=1}^{\log p} \left( \lambda + \frac{2^{i-1} n}{\beta p} \right) = \lambda \log p + \frac{n(p-1)}{\beta p}$$

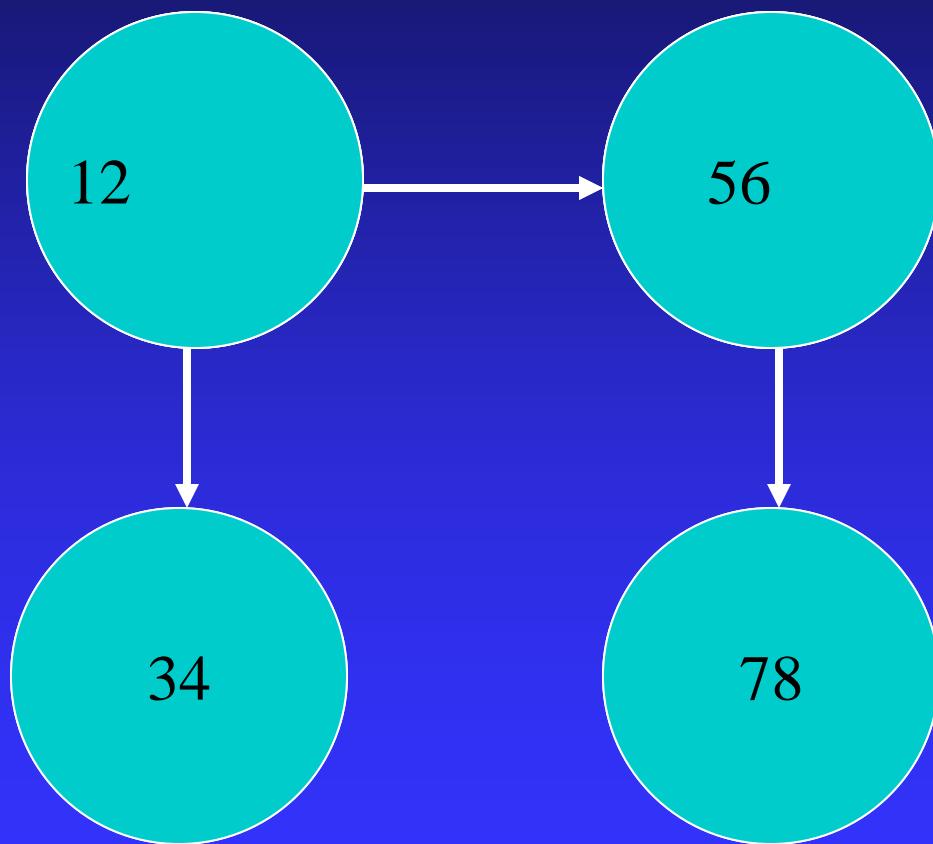
# Adding Data Input



# Scatter



# Scatter in log $p$ Steps



# Summary: Task/channel Model

- Parallel computation
  - ◆ Set of tasks
  - ◆ Interactions through channels
- Good designs
  - ◆ Maximize local computations
  - ◆ Minimize communications
  - ◆ Scale up

# Summary: Design Steps

- Partition computation
- Agglomerate tasks
- Map tasks to processors
- Goals
  - ◆ Maximize processor utilization
  - ◆ Minimize inter-processor communication

# Summary: Fundamental Algorithms

- Reduction
- Gather and scatter
- All-gather