Parallel Programming
in C with MPI and OpenMP

Michael J. Quinn

McGraw Hill

Chapter 3
Parallel Algorithm Design

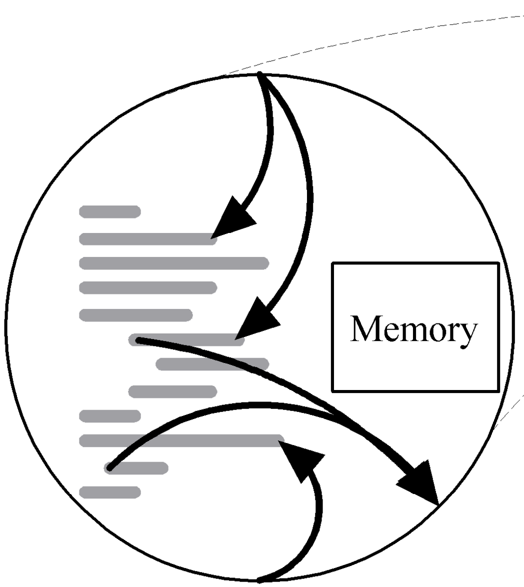
Outline

- Task/Channel Model
- Algorithm Design Methodology
- Case Studies
  - Boundary value problem
  - Finding the maximum
  - The n-body problem
  - Adding data input

Task/Channel Model

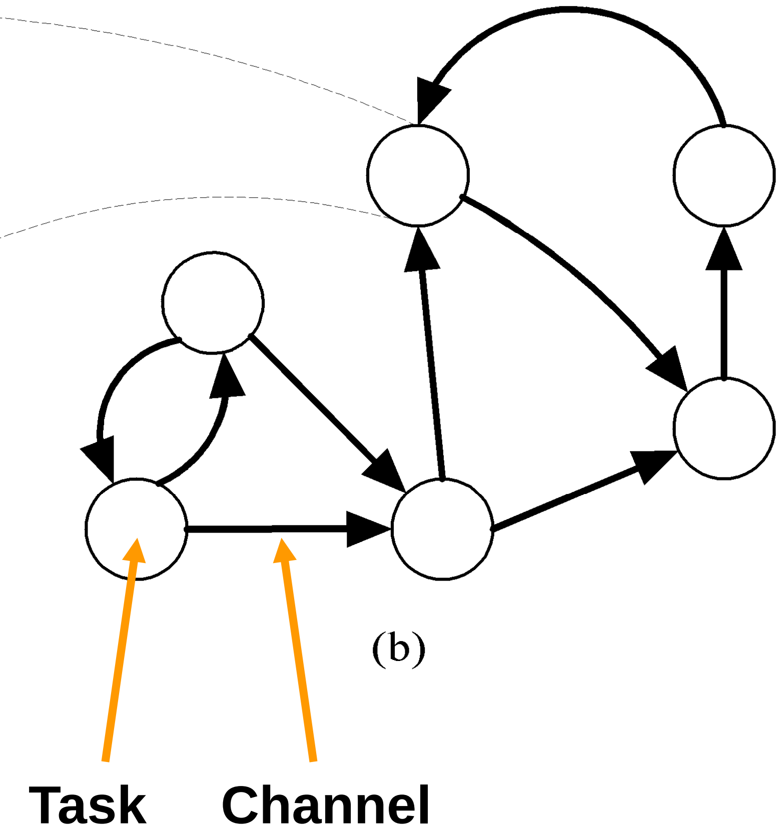
- Parallel computation
  - a set of tasks interact by sending messages through channels
- Task
  - Program + Local memory + Collection of I/O ports
- Channel
  - a message queue (output port → input port)
  - Receiving is a synchronous operation (wait→block)
    Sending is an asynchronous operation

Task/Channel Model



(a)

Task
= Program
+ Local Memory
+ I/O Ports



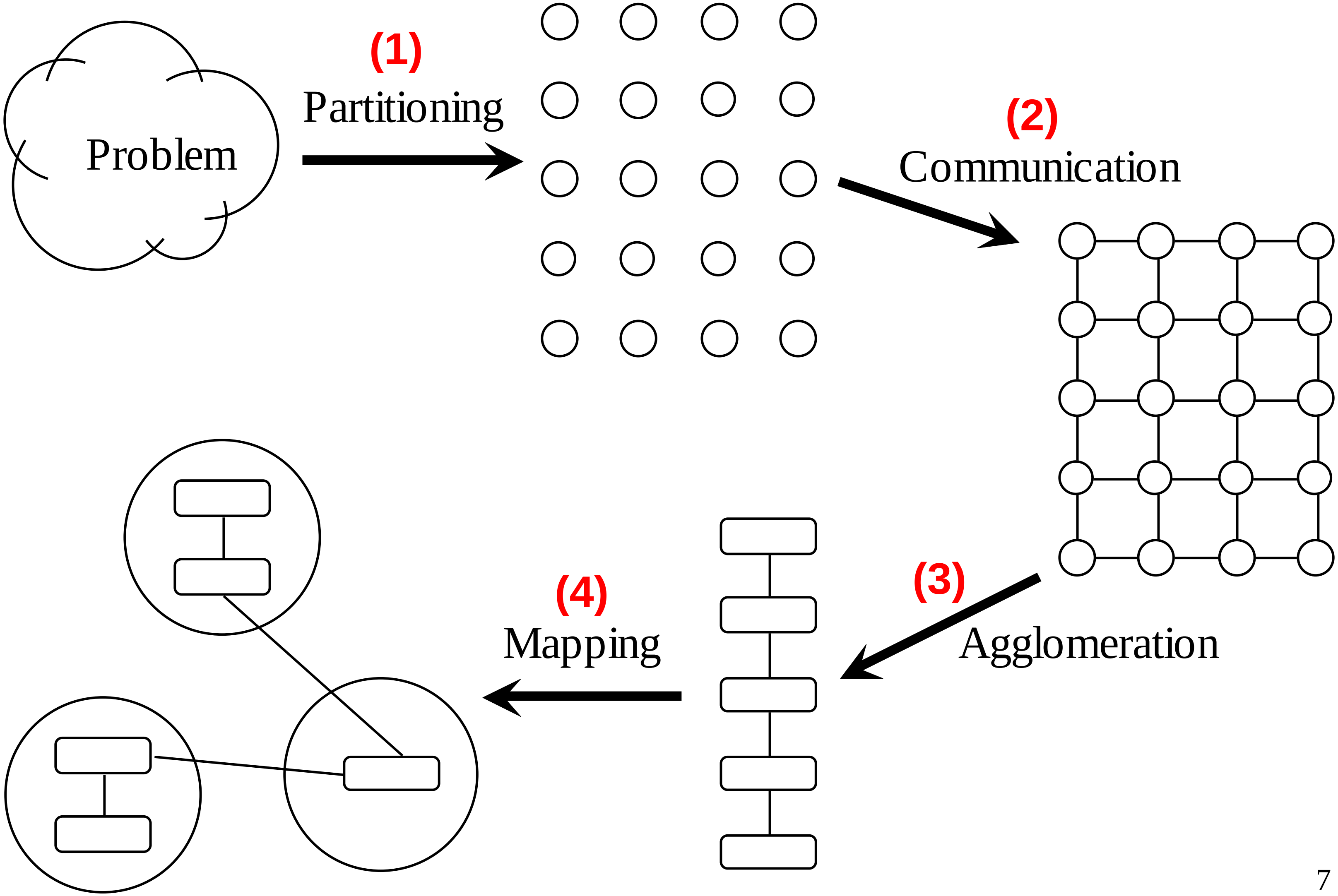
(b)

Foster's Design Methodology

- Ian Foster has proposed a 4-steps process for designing parallel algorithms

1. Partitioning
2. Communication
3. Agglomeration
4. Mapping

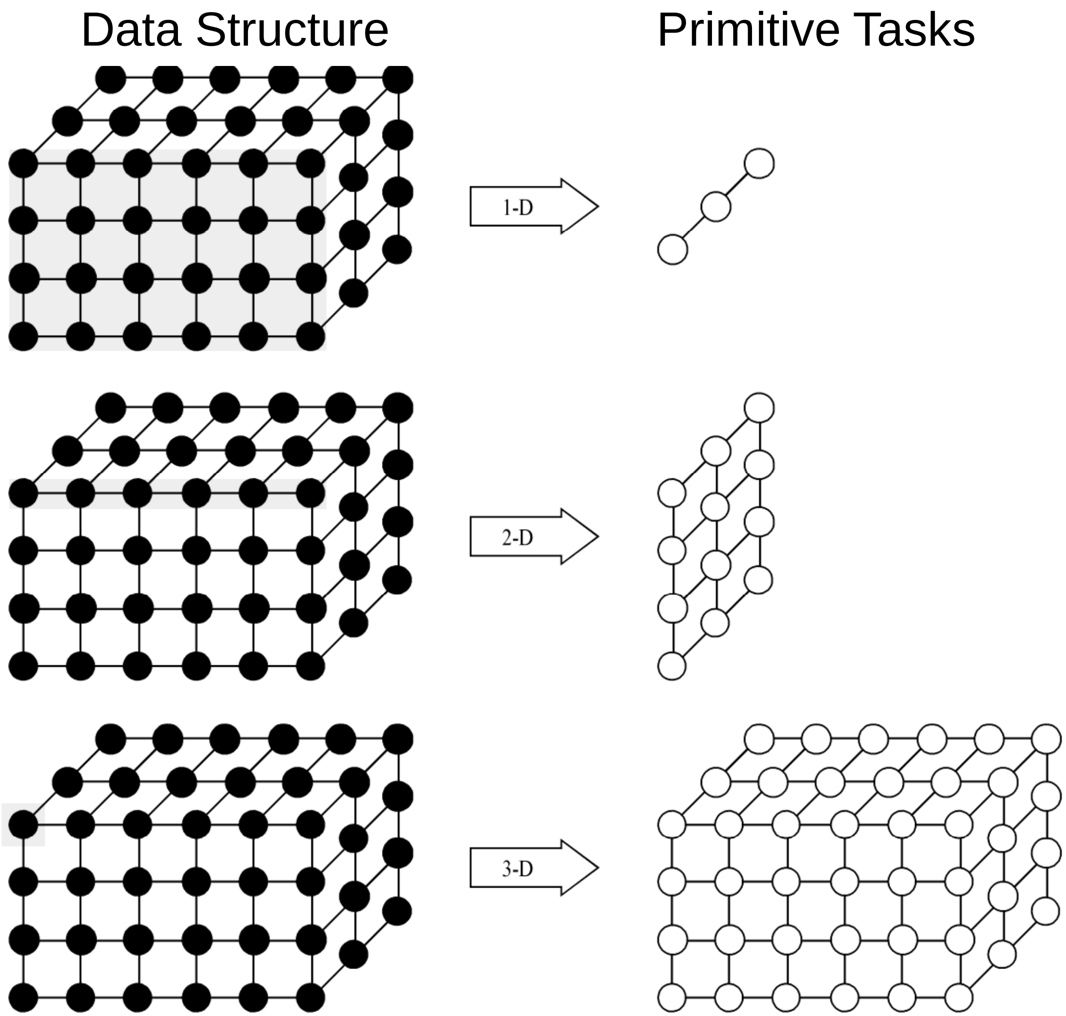
Foster's Methodology



Partitioning

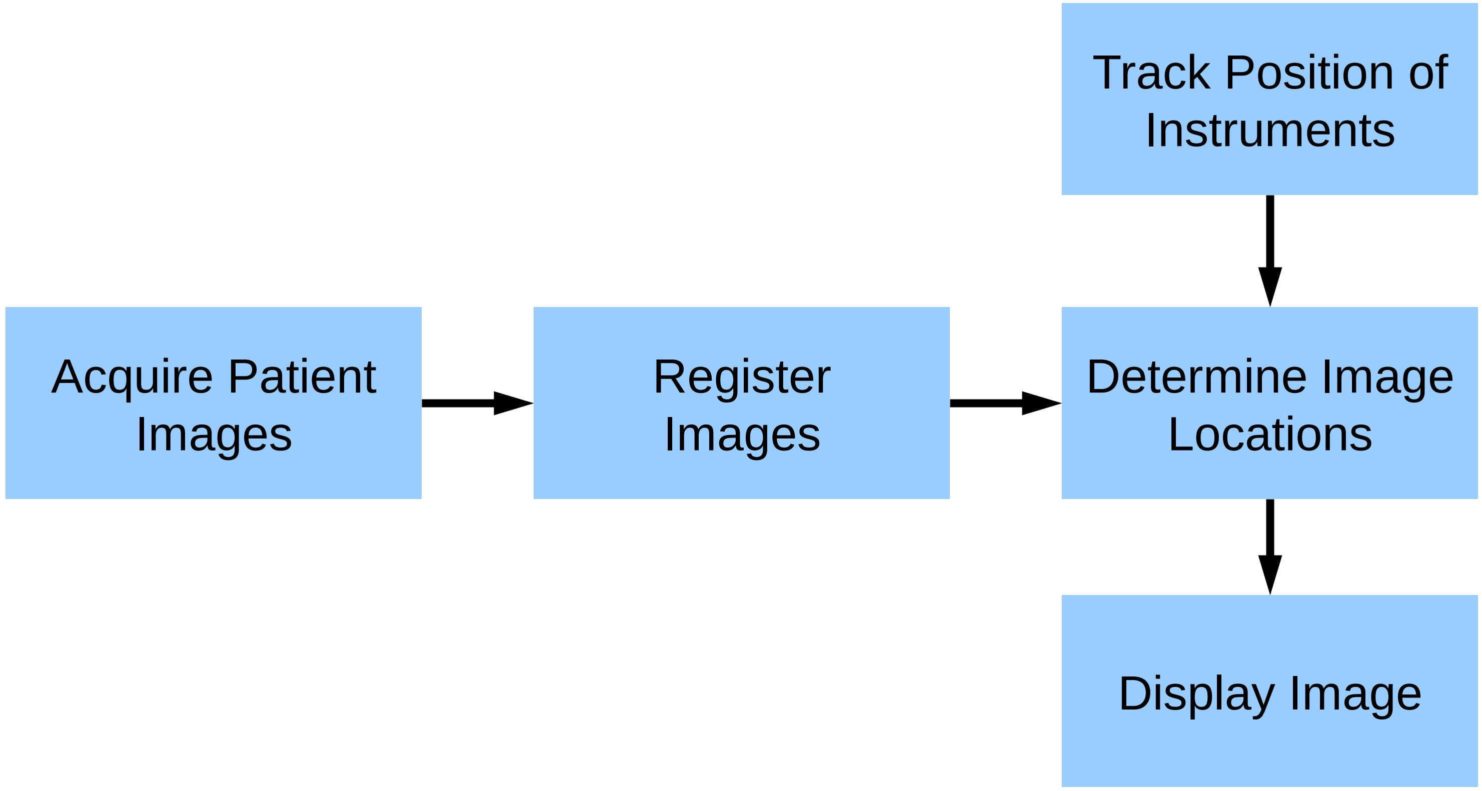
- Dividing computation and data into pieces.
  - Data-centric approach or computation-centric approach
- Domain decomposition
  - Divide data into pieces
  - Determine how to associate computations with the data
- Functional decomposition
  - Divide computation into pieces (e.q. pipelining)
  - Determine how to associate data with the computations

Example Domain Decompositions



Three domain decompositions of a 3D matrix

Example Functional Decomposition



Functional decomposition of a system supporting interactive image-guided surgery

Partitioning Checklist

- **More tasks:** At least 10× more primitive tasks than processors in target parallel computer.
- **Less redundancy:** Minimize redundant computations and redundant data structure storage.
- **Same size:** Primitive tasks are roughly the same size.
- **Scalability:** The number of tasks is an increasing function of the 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

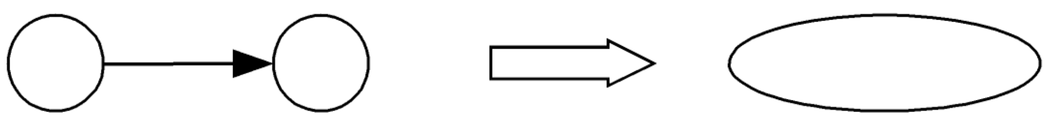
- **Equal traffic:** Communication operations balanced among tasks
- **Less traffic:** Each task communicates with only small group of neighbors
- **Talking:** Tasks can perform communications concurrently
- **Working:** Tasks can perform computations concurrently

Agglomeration

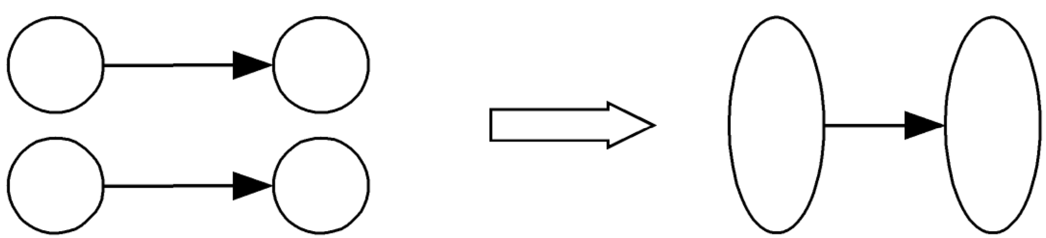
- Grouping tasks into larger tasks
  - Improve performance (a.)
  - Simplify programming (b.)
- Goals
  - Lower communication overhead (a.)
  - Maintain scalability of the parallel design (a.)
  - Reduce software engineering cost (b.)
- In MPI programming, goal often to create one agglomerated task per processor

Agglomeration Can Improve Performance

- Increasing the locality: Eliminate communication between primitive tasks agglomerated into consolidated task
- Reducing channels: Combine groups of sending and receiving tasks



(a) Increasing the locality



(b) Reducing the transmissions

Agglomeration Checklist

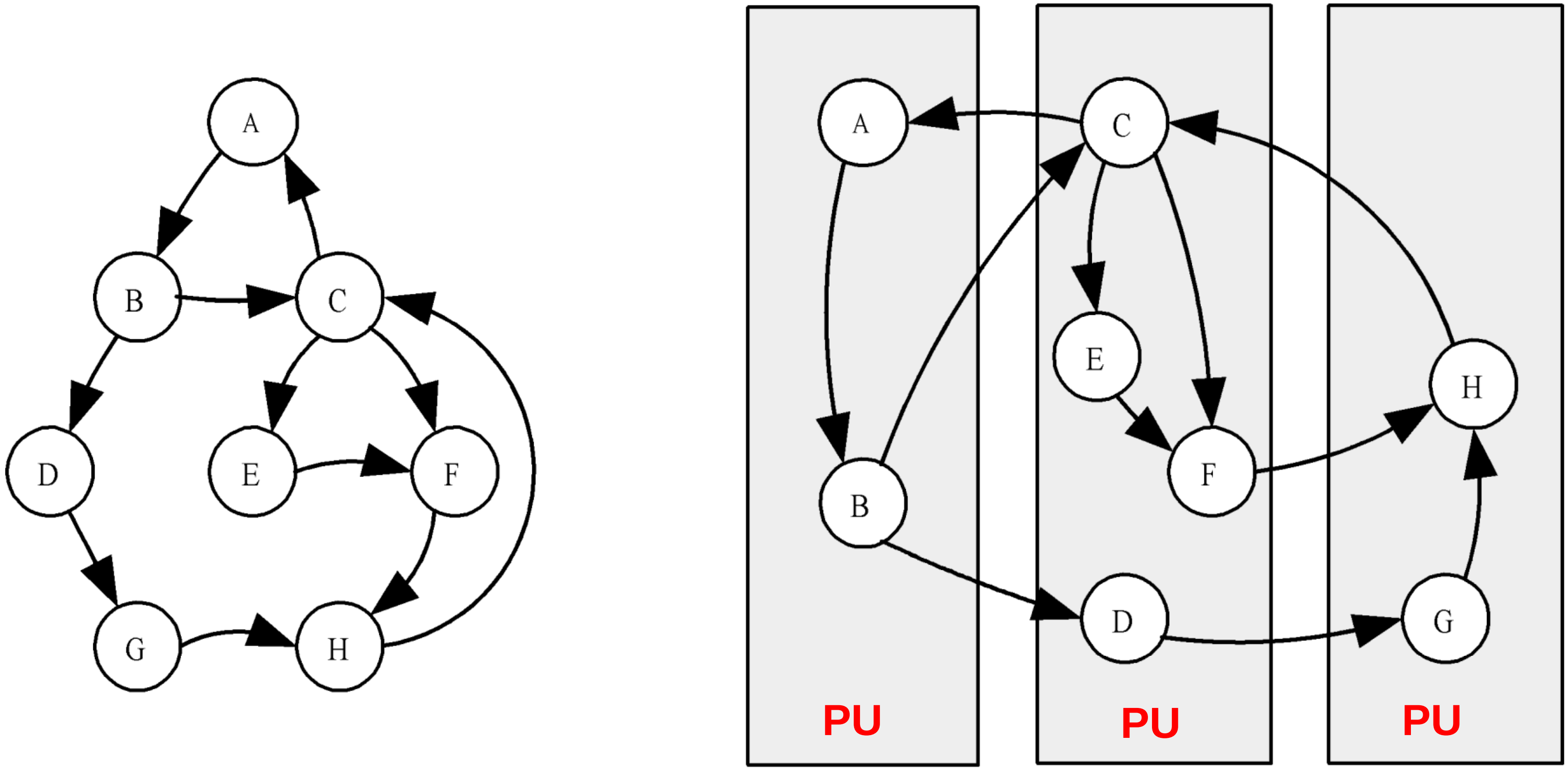
Tuning partitioning and **communication**.

- 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
- **Trade-off** 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** (our target)
- Conflicting goals of mapping
  - Maximize processor utilization
  - Minimize inter-processor communication

Mapping Example



(a) Task/channel graph

(b) Mapping of tasks to three processors

Optimal Mapping

- Finding optimal mapping is NP-hard
  - There are no known polynomial-time algorithms to map tasks to processors to minimize the execution time.
  - NP-hard
    - NP-hard, NIST, http://www.nist.gov/dads/HTML/nphard.html
    - NP-hard, Wikipedia, http://en.wikipedia.org/wiki/NP-hard
- Must rely on heuristics

Decision Tree of Mapping Strategy (1)

- Static number of tasks
  - Structured communication
    - Constant computation time per task
      - **Agglomerate tasks to minimize communication**
      - **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

Decision Tree of Mapping Strategy (2)

- 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**

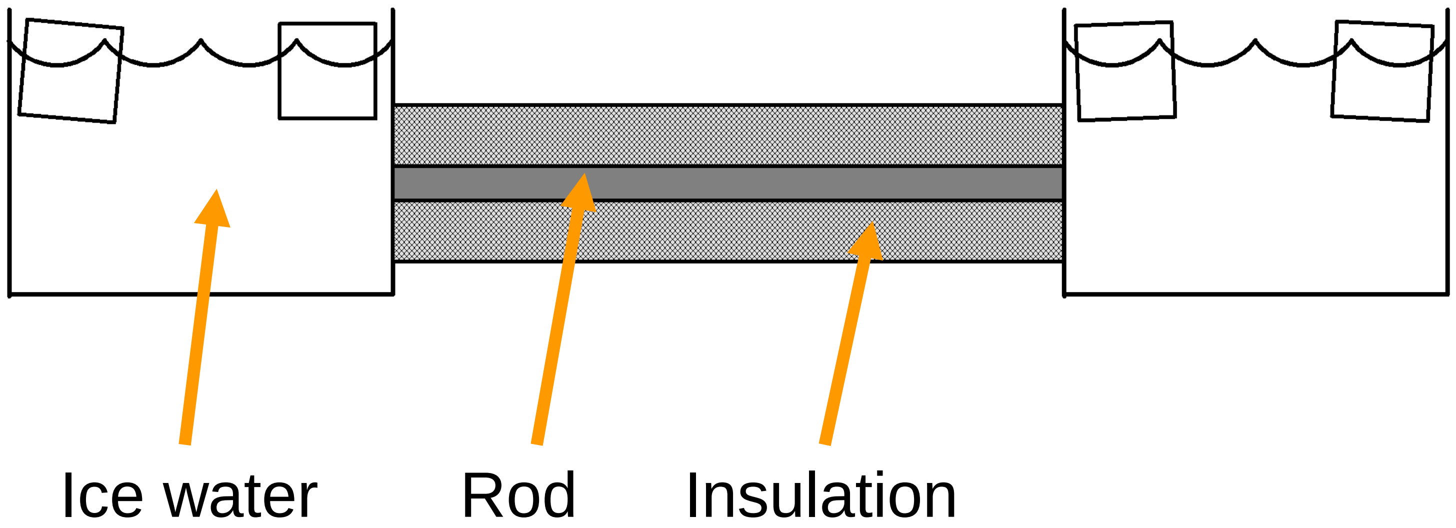
Mapping Checklist

- **1 vs. M:** Considered designs based on one task per processor and multiple tasks per processor
- **S vs. D:** 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

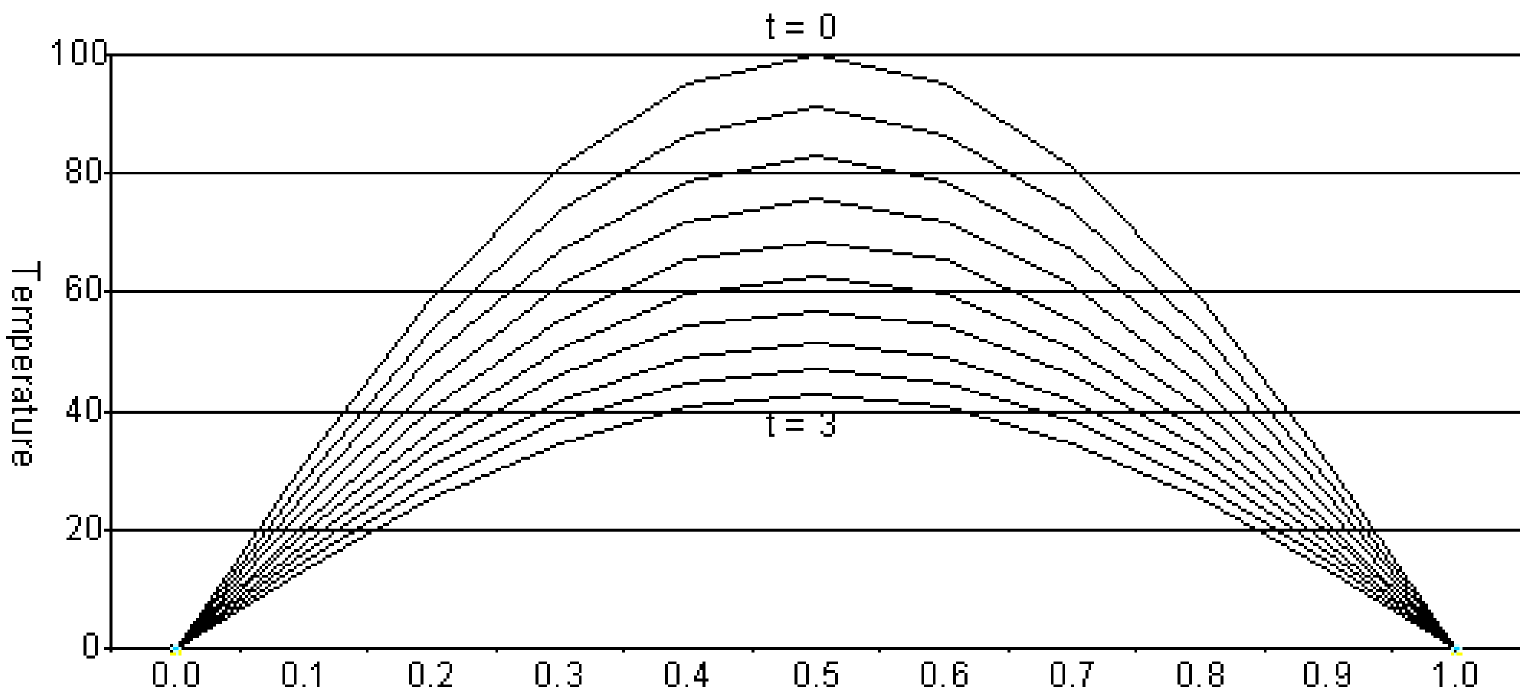
- Boundary value problem
- Finding the maximum
- The n-body problem
- Adding data input

Boundary Value Problem



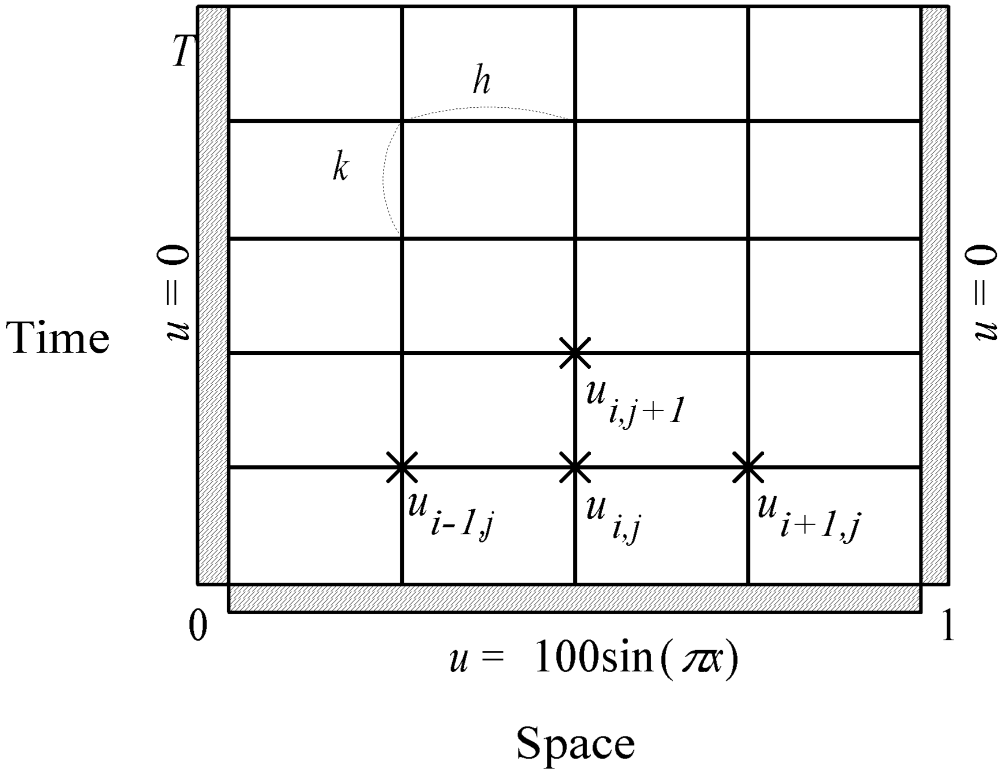
A thin rod is suspended between two ice baths.
Then ends of the rod are in contact with the icewater.
The rod is surrounded by a thick blanket of insulation.
We can use a partial differential equation to model the temperature at any point on the rod as a function of time

Rod Cools as Time Progresses



The finite difference method finds the temperature at a fixed number of points in the rod at certain time intervals. Decreasing the size of the steps in space and time can lead to more accurate solutions.

Finite Difference Approximation



$$u = 100\sin(\pi x)$$

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

$$r = \frac{k}{h^2}$$

Data structure used in a finite difference approximation to the rod-cooling problem. Every point $\boldsymbol{u_{i,j}}$ represents a matrix element containing the temperature at position $\boldsymbol{i}$ on the rod at time $\boldsymbol{j}$. At each end of the rod the temperature is always **0**. At time **0** the temperature at point $\boldsymbol{x}$ is $\mathbf{100\sin(\pi x)}$.

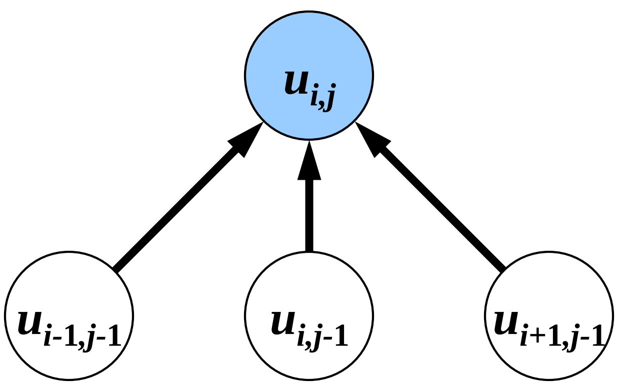
1. Partitioning

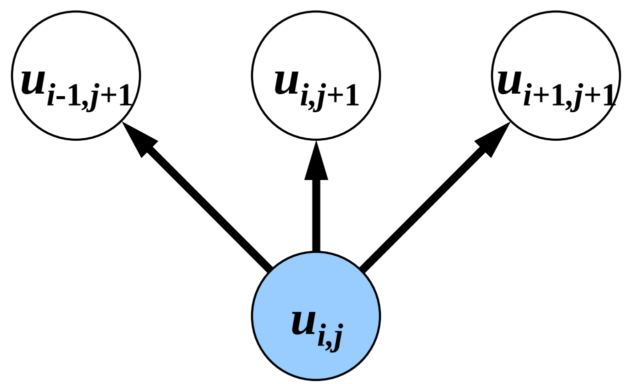
- One data item per grid point – easy
- Associate one primitive task with each grid point
- Two-dimensional domain decomposition

2. Communication

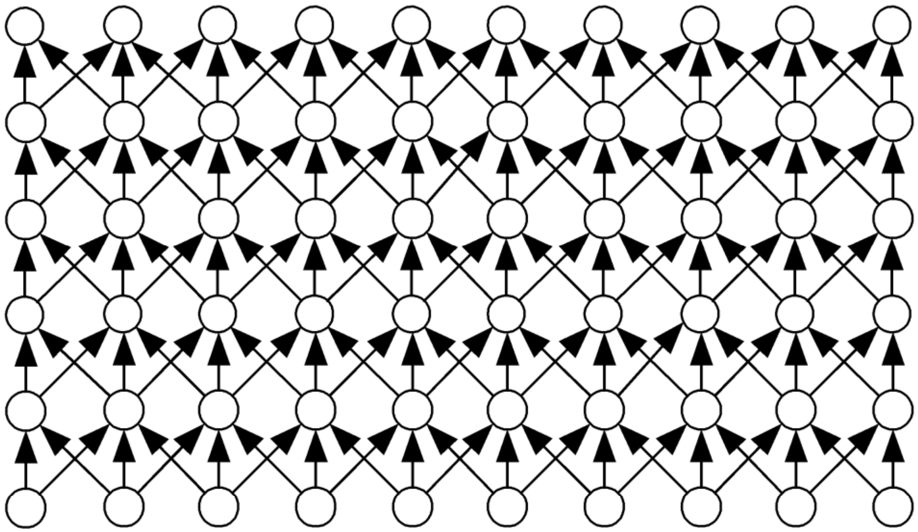
- Identify communication pattern between primitive task
- Each interior primitive task has three incoming and three outgoing channels

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

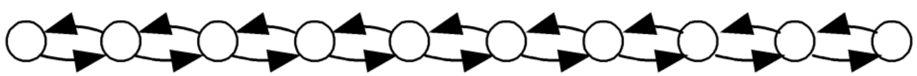




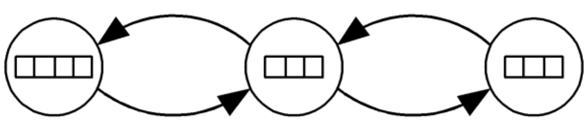
3. Agglomeration and 4. Mapping



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



(b) Element ***i*** for all time steps.



(c) Over all time steps.

Sequential Execution Time

- The rod has been divided into n pieces of size h.
  - χ – time to update element $u_{i,j+1}\left(=ru_{i-1,j}+(1-2r)u_{i,j}+ru_{i+1,j}\right)$
  - n – number of elements
  - m – number of iterations
- Sequential execution time: $\boxed{m(n-1)\chi}$

Parallel Execution Time

- If each processor is responsible for an equal-sized portion of the rod's elements.

  p – number of processors

  λ – message latency

- The computation time for each iteration: $\boxed{\chi\left\lceil \frac{n-1}{p} \right\rceil}$
- Necessary communication time: 2λ
- Parallel execution time: $\boxed{m\left(\chi\left\lceil \frac{n-1}{p} \right\rceil + 2\lambda\right)}$

Finding the Maximum Error

- The error between the computed solution x and correction solution is $|(x - c) / c|$.
- Enhance previous parallel algorithm to find 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%

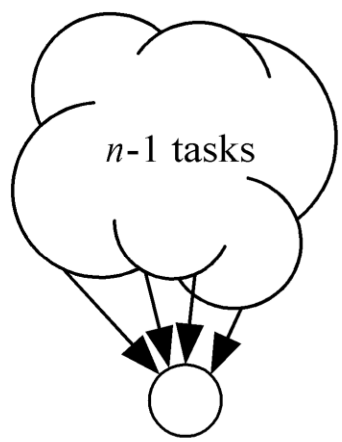
6.25%

Reduction

- Given
  - a set of n values $a_0, a_1, a_2, \ldots, a_{n-1}$
  - an associative operator $\oplus$
- Reduction is the process of computing $a_0 \oplus a_1 \oplus a_2 \oplus \ldots \oplus a_{n-1}$
  - Examples: add, multiply, AND, OR, maximum, minimum
- Reduction requires exactly $n-1$ operations, it has $\Theta(n)$ time complexity on a sequential computer.
  - How to perform a reduction on parallel computer quickly?

Parallel Reduction Evolution (1)

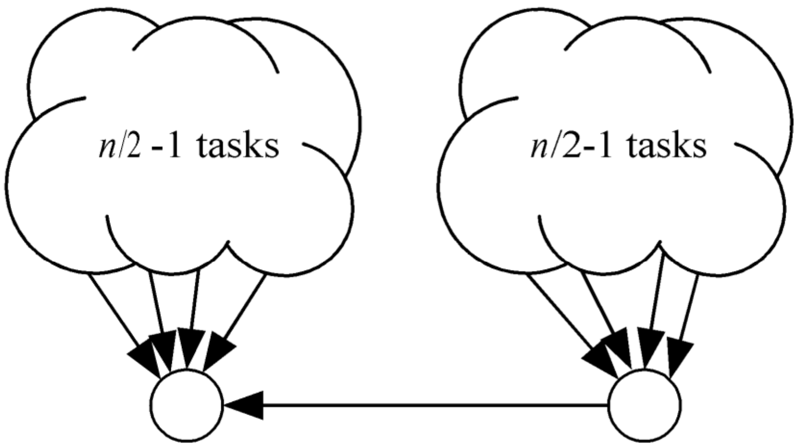
- Partitioning
  - Divide it into ***n*** pieces, one task per piece.
- Communication
  - χ – time to perform an addition
  - λ – message latency



Only one task receives all other ***n*–1** results.

$$\text{Time complexity} = (n-1)(\lambda+\chi)$$

Parallel Reduction Evolution (2)



Two tasks work together.

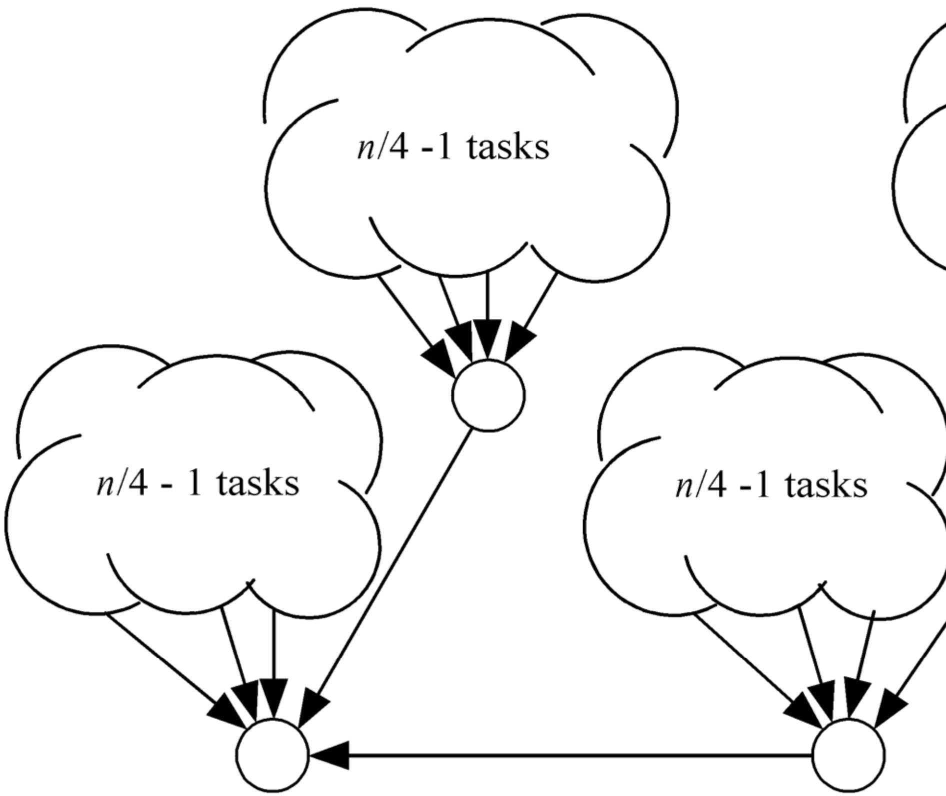
$$2\left(\frac{n}{2}-1\right)+1=n-1$$

Time complexity

$$=\left(\frac{n}{2}-1\right)(\lambda+\chi)+(\lambda+\chi)$$

$$=\frac{n}{2}(\lambda+\chi)$$

Parallel Reduction Evolution (3)



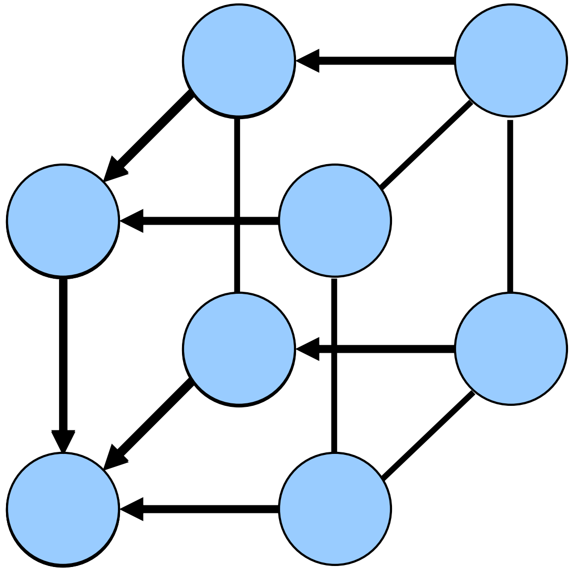
Four task cooperate. $4\left(\frac{n}{4}-1\right)+3=n-1$

Time complexity

$$=\left(\frac{n}{4}-1\right)(\lambda+\chi)+2(\lambda+\chi)$$

$$=\left(\frac{n}{4}+1\right)(\lambda+\chi)$$

Continue … Binomial Trees



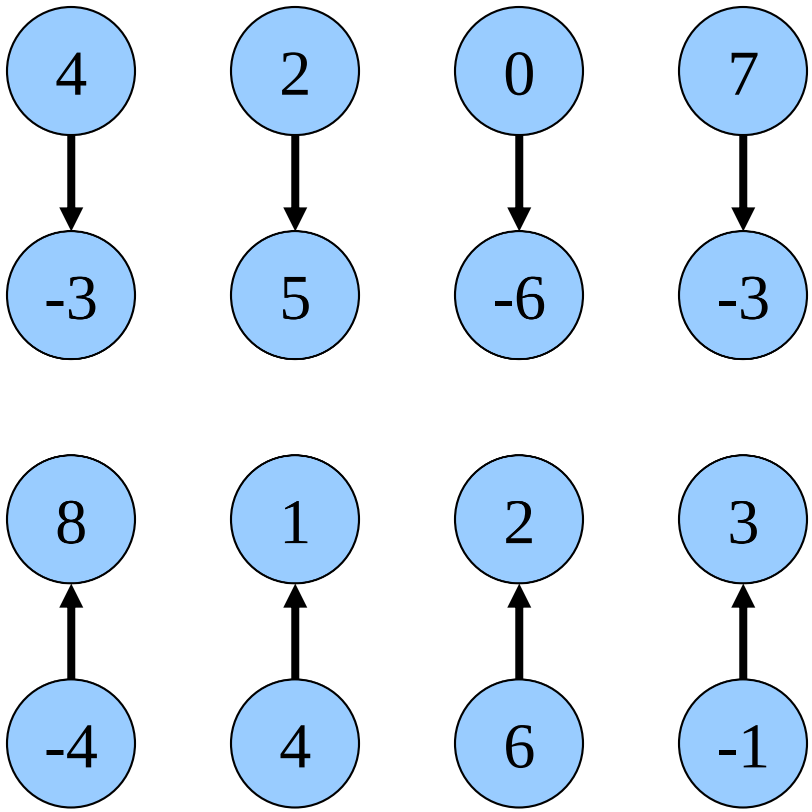
Subgraph of Hypercube

$n = 2^k$

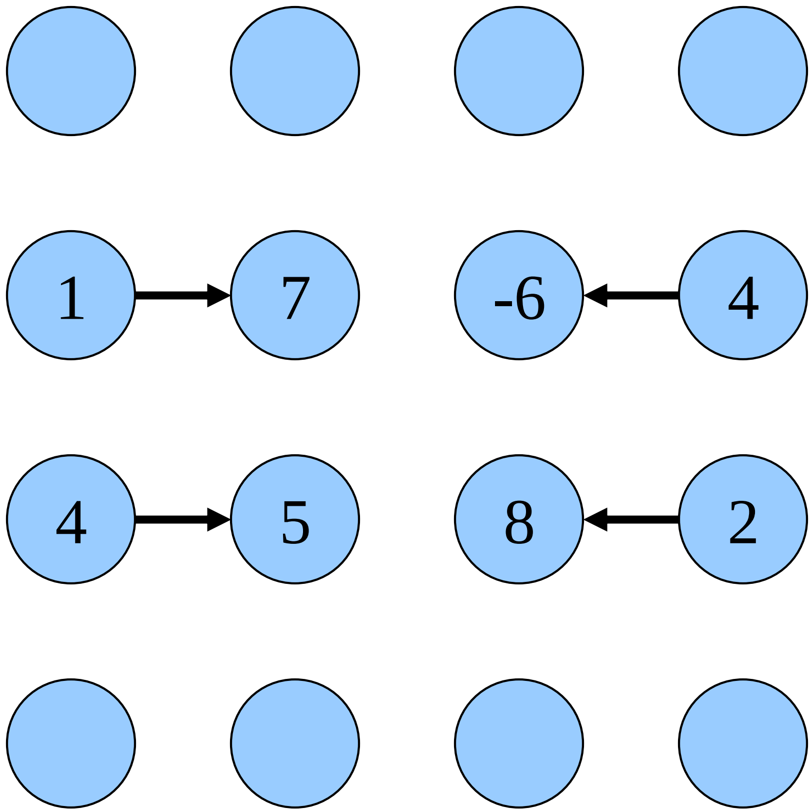
$k = \log n$

Binomial trees with 1, 2, 4, and 8 nodes.

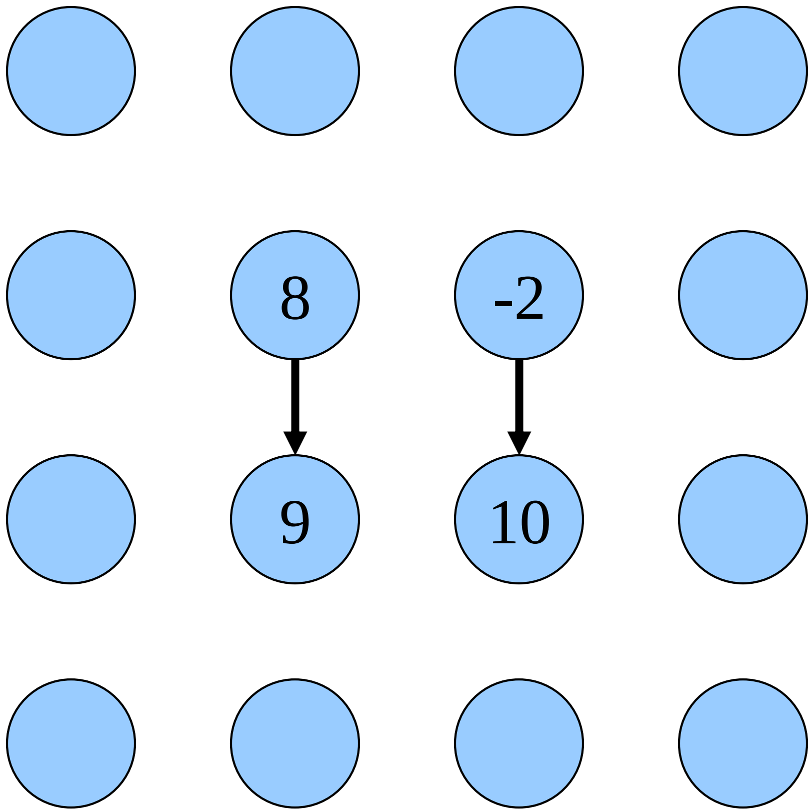
Finding Global Sum in logarithmic time



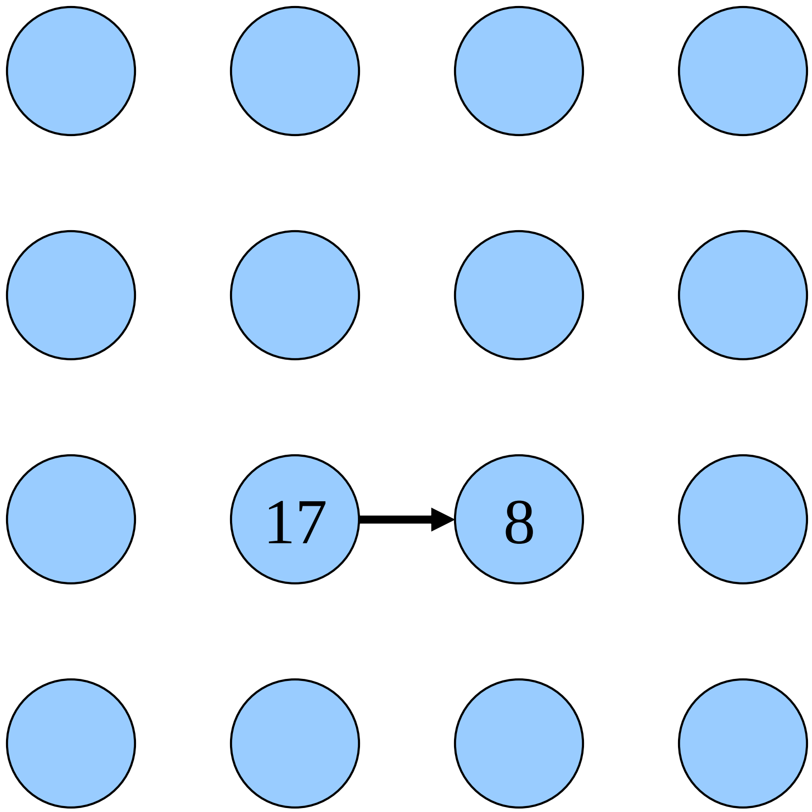
Finding Global Sum



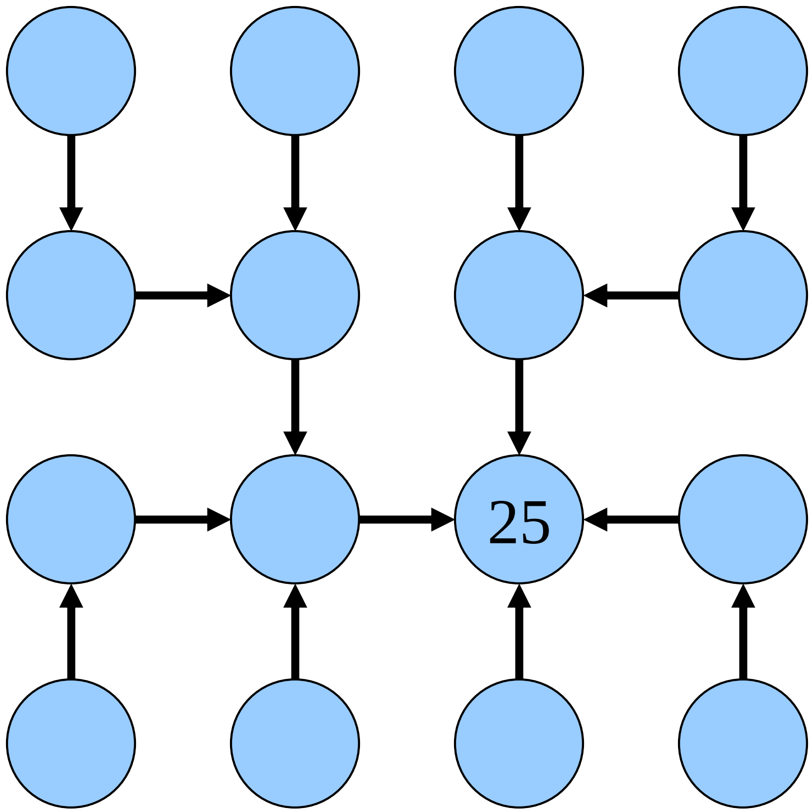
Finding Global Sum



Finding Global Sum

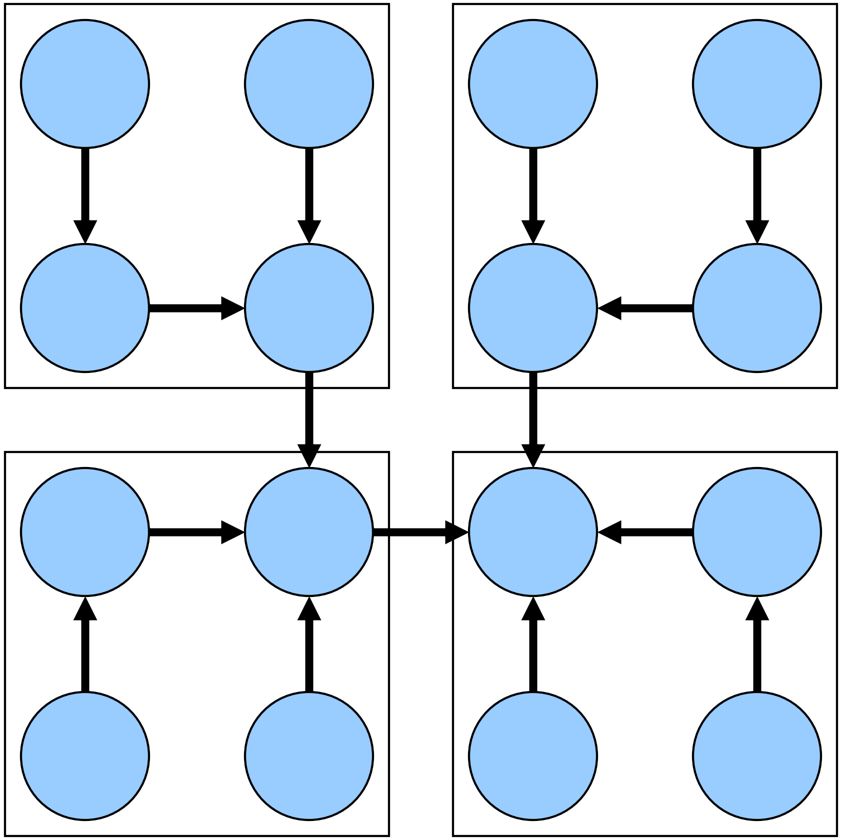


Finding Global Sum



Binomial Tree

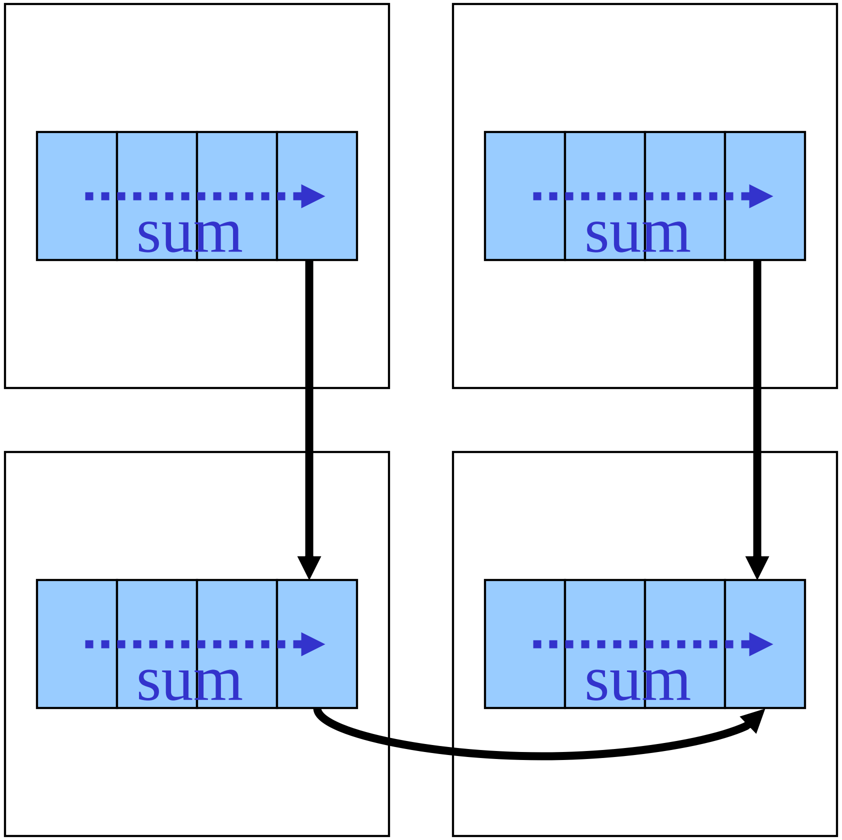
Agglomeration and Mapping



16 tasks are mapped to 4 processors.

n tasks mapped to
p processors

Agglomerate Primitive Tasks



4 tasks on each processor are agglomerated into a single task

n/p primitive tasks with **1** value

$\Downarrow$

1 primitive tasks with ***n/p*** values

Analysis

χ – time to perform the binary operation

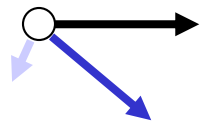
λ – message latency

- All tasks performance concurrently: $\boxed{\left(\left\lceil \frac{n}{p} \right\rceil - 1\right)\chi}$

- A reduction of p values distributed among p tasks can be preformed in $\lceil \log p \rceil$ communication steps.
- Each reduction stop requires time: $\lambda + \chi$
- Overall execution time: $\boxed{\left(\left\lceil \frac{n}{p} \right\rceil - 1\right)\chi + \lceil \log p \rceil (\lambda + \chi)}$

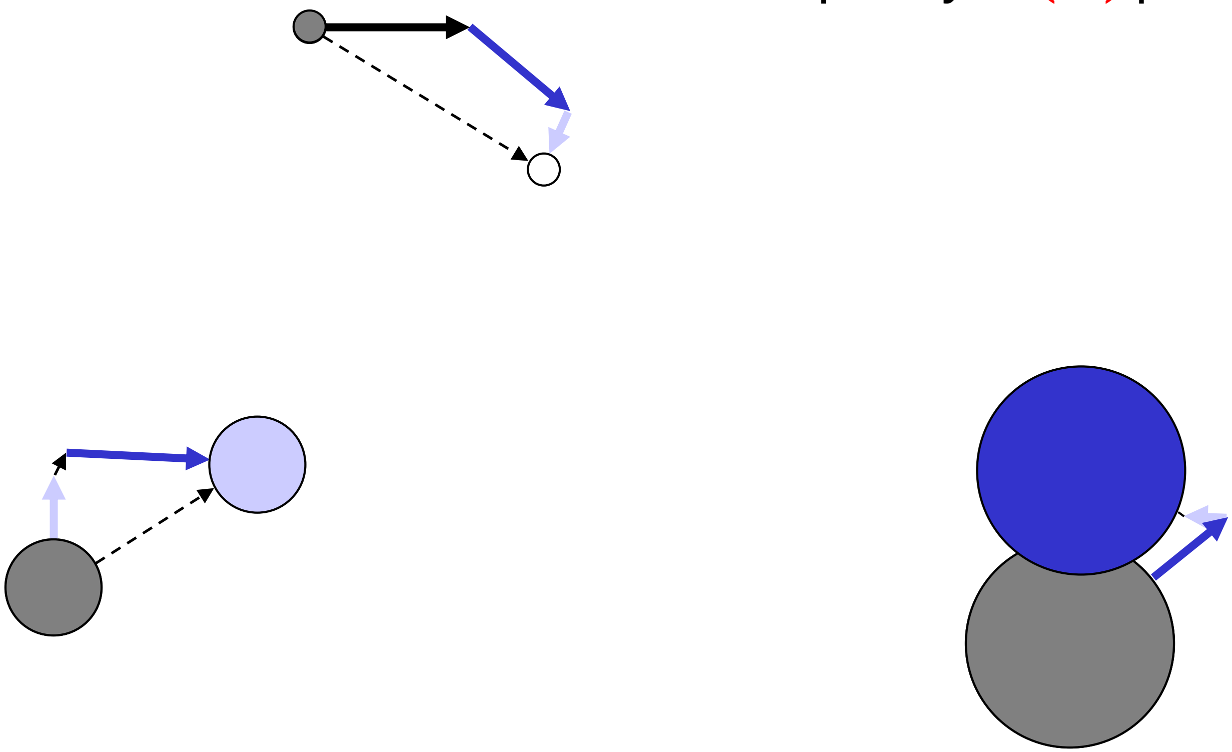
The n-Body Problem

- Newtonian n-body simulation.



Straightforward sequential algorithms
Time complexity: $\Theta(n^2)$ per iteration.

The n-Body Problem

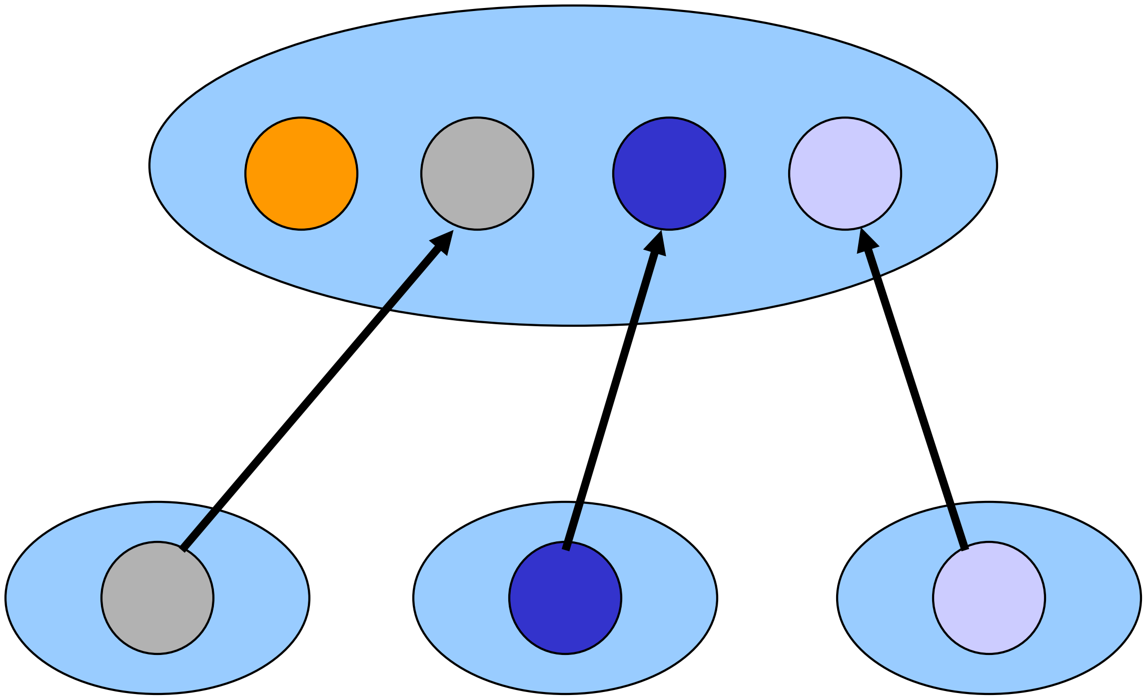


Straightforward sequential algorithms
Time complexity: $\Theta(n^2)$ per iteration.

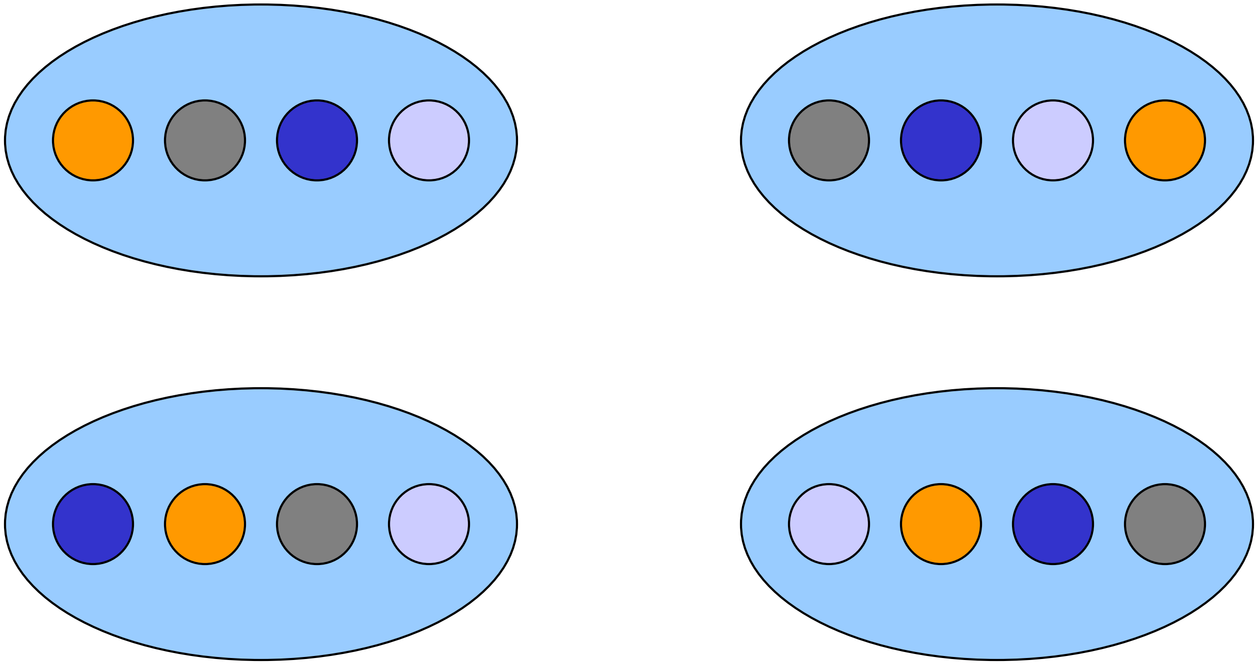
Partitioning

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

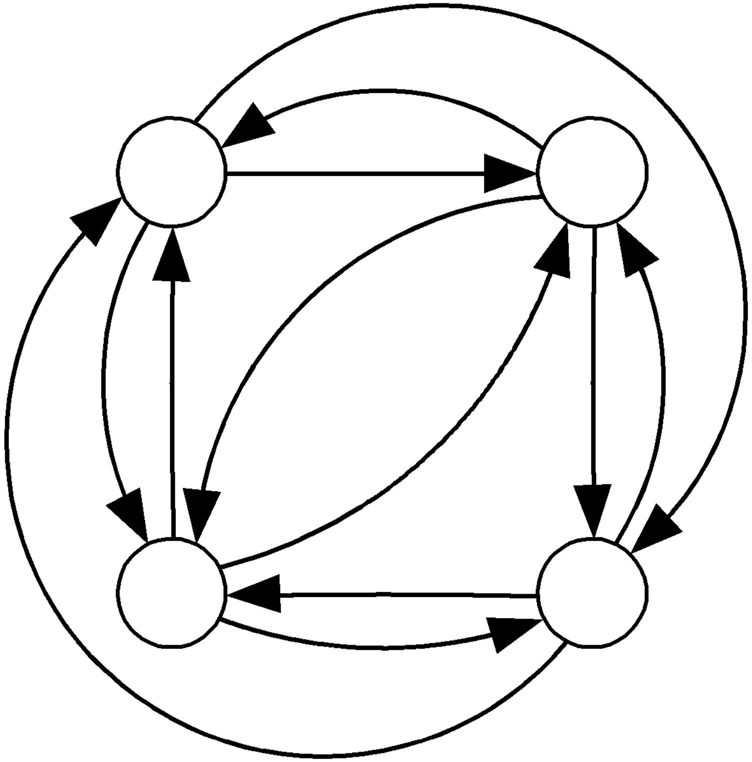
Gather



All-gather

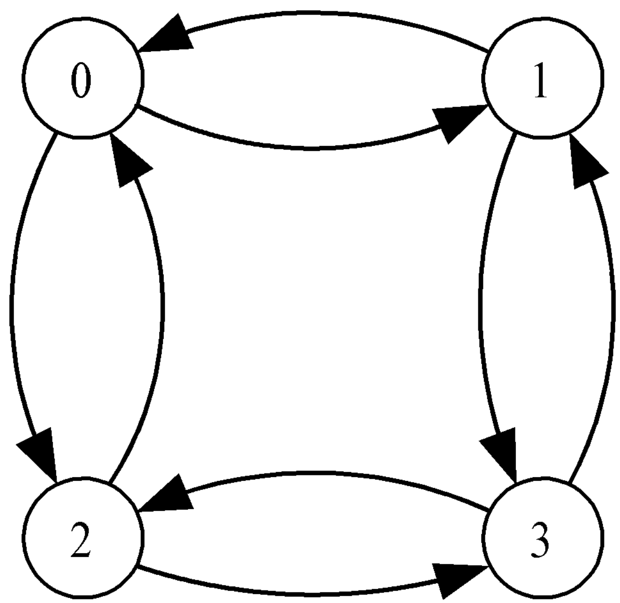


Complete Graph for All-gather



Set up a channel between every pair of tasks.

Hypercube for All-gather



Each task have only **log *p*** outgoing channels
and **log *p*** incoming channels

Analysis

β – bandwidth of each channel

- Communication time
  - Complete graph

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

  - Hypercube

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

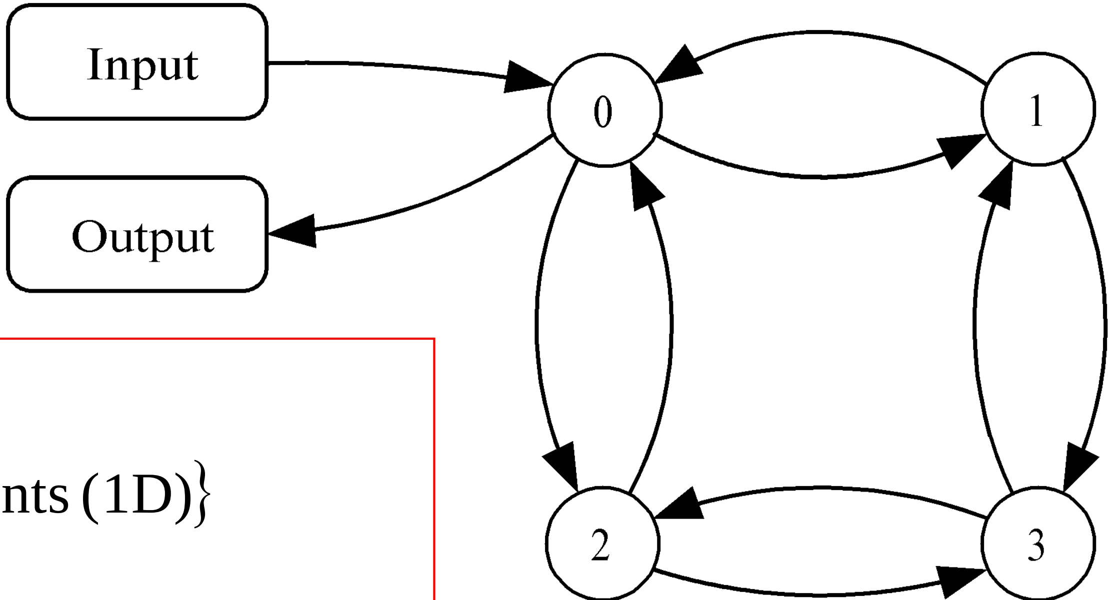
- Overall time complexity

$$\boxed{\lambda \log p + \frac{n(p-1)}{\beta p} + \chi\left(\frac{n}{p}\right)(n-1)}$$

Error in textbook

Adding Data Input

- Augment the task/channel graph for the n-body problem



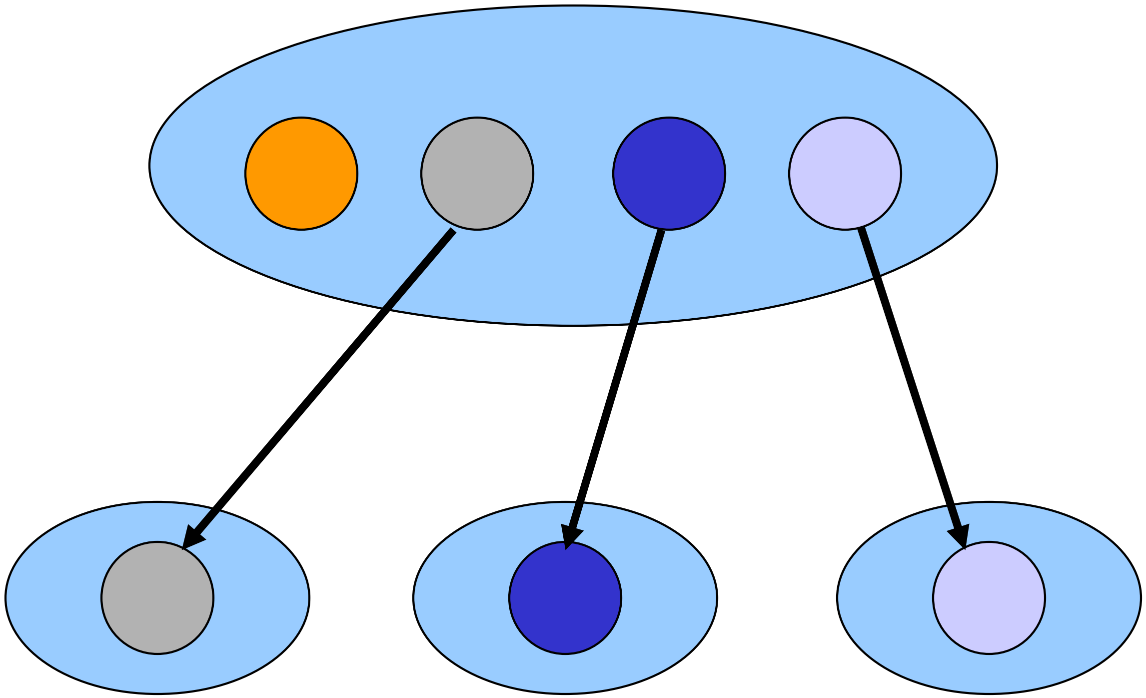
I/O Time Complexity:

$$\lambda_{io} + \frac{n}{\beta_{io}} \{\text{for } n \text{ data elements (1D)}\}$$

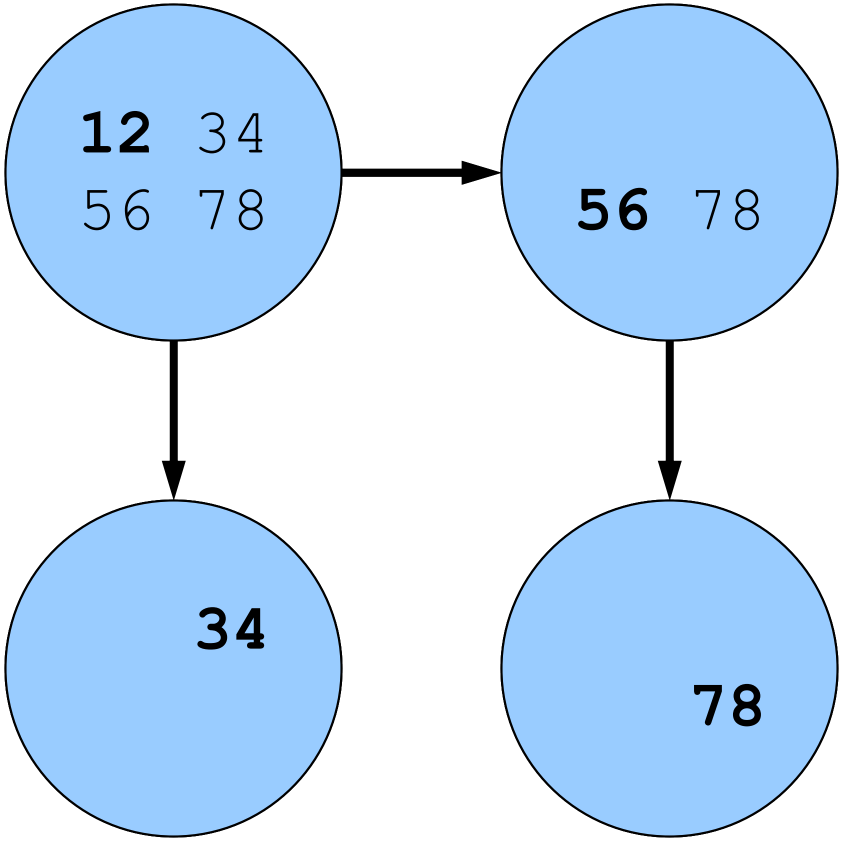
$$\lambda_{io} + \frac{4n}{\beta_{io}} \left\{ \begin{array}{l} \text{for positions (2D)} \\ \text{and velocities (2D) of } n \text{ particles} \end{array} \right\}$$

Task 0 is responsible for I/O.

Scatter



Scatter in log *p* Steps



Communication

- Time for scatter the particles (sequential)
  - Send p–1 message, each of length $4n/p$

$$(p-1)\left(\lambda+\frac{4n}{p\beta}\right)=(p-1)\lambda+\frac{4n(p-1)}{p\beta}$$

- Time for scatter the particles (in $\log p$ steps)

$$\sum_{i=1}^{\log p}\left(\lambda+\frac{4n}{2^i\beta}\right)=\lambda\log p+\frac{4n(p-1)}{p\beta}$$

Error in textbook

- The "$\log p$ steps" algorithm is better.

Analysis

- Overall execution time for ***m*** iterations:

$$2\left(\lambda_{io} + \frac{4n}{\beta_{io}}\right)$$

Input and Output

$$+2\left(\lambda \log p + \frac{4n(p-1)}{p\beta}\right)$$

Log p steps scattering

$$+m\left(\lambda \log p + \frac{2n(p-1)}{p\beta} + \chi\left\lceil \frac{n}{p} \right\rceil (n-1)\right)$$

All-gather and computation

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

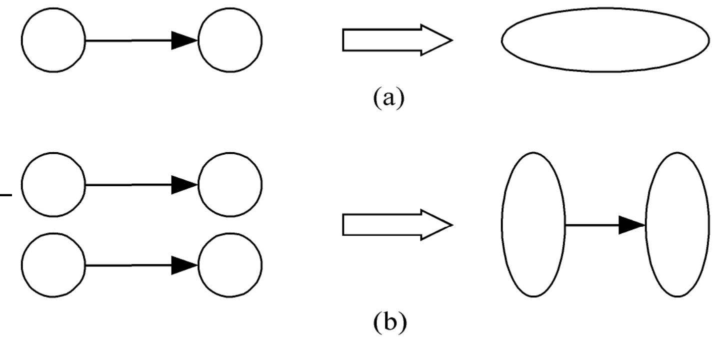
Exercise

- 3.11
- 3.13
- 3.18, 3.19

Discussion

- Real world
  - Processor
    - Single Core
    - Hyper-threading, multi-threading
      - http://www.intel.com/personal/desktop/dualcore/demo/popup/demo.htm
    - Dual Core
  - Programming
    - Multi-thread (share memory space)
    - Multi-process (no share memory space)
      → extend to MPI process.

Parallelization Policy



Primitive Task

Task

Agglomerated Task

1-to-1 mapping?

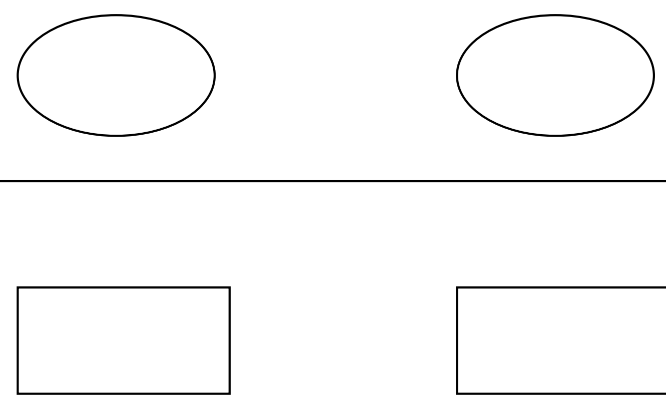
Inter-Process Communication

Broadcast/reduction

Gather/scatter

All-gather

Process



1-to-1 mapping?

Processor