# Chapter 3 Parallel Algorithm Design

## Parallel Programming

### in C with MPI and OpenMP

Michael J. Quinn

Wonyong Sung Modification

# Parallel Programming

- Load balancing
  - Best with data-level parallel processing
- Low communication overhead
  - Architecture dependent
- Precedence relation and scheduling
- Memory/cache/IO consideration

# Partitioning

- Dividing both computation and data into pieces
- Domain (x-axis, data) decomposition
  - Divide data into pieces – in many case best partitioning
  - SPMD (Single Processor Multiple Data) paradigm
  - Good for massively parallel distributed memory multi-computer systems
- Functional decomposition
  - Divide computation into pieces
  - May good for heterogeneous multiprocessors

# 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 (scalable partitioning)

# 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
  - Eliminate communication between primitive tasks agglomerated into consolidated task Maintain scalability of program
  - Combine groups of sending and receiving tasks
- In MPI programming, goal often to create one agglomerated task per processor

# 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
- MPI and OpenMP programming model
  - MPI: purely parallel, need parallel algorithm from the start-up
  - OpenMP: Fork-join model, starting-from sequential version and incremental parallelization
- Conflicting goals of mapping
  - Maximize processor utilization
  - Minimize interprocessor communication
- NP-hard problem

# 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 – analyzes the current tasks and produces a new mapping of tasks to processors
  - Many short-lived tasks
    - Use a run-time task-scheduling algorithm

# Task Scheduling

- Centralized method
  - One manager and many workers
  - When a worker processor has nothing to do, it requests a task from the manager.
  - Sometimes, the manager can be a bottleneck
- Distributed
  - Each processor maintains its own list of tasks
  - Push (processors with too many send to some others) and pull strategies
- Hybrid method

# 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

# 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

# Sequential execution time

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

# 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

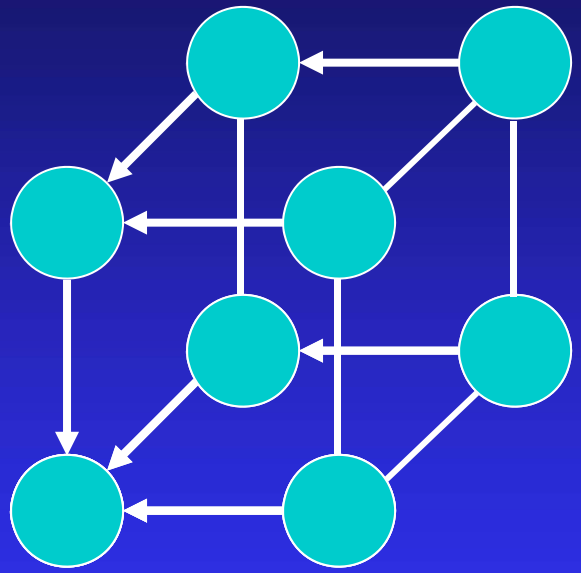
| | | | | |
|---|---|---|---|---|
| Computed | 0.15 | 0.16 | 0.16 | 0.19 |
| Correct | 0.15 | 0.16 | 0.17 | 0.18 |
| Error (%) | 0.00% | 0.00% | 6.25% | 5.26% |

6.25%

# Reduction

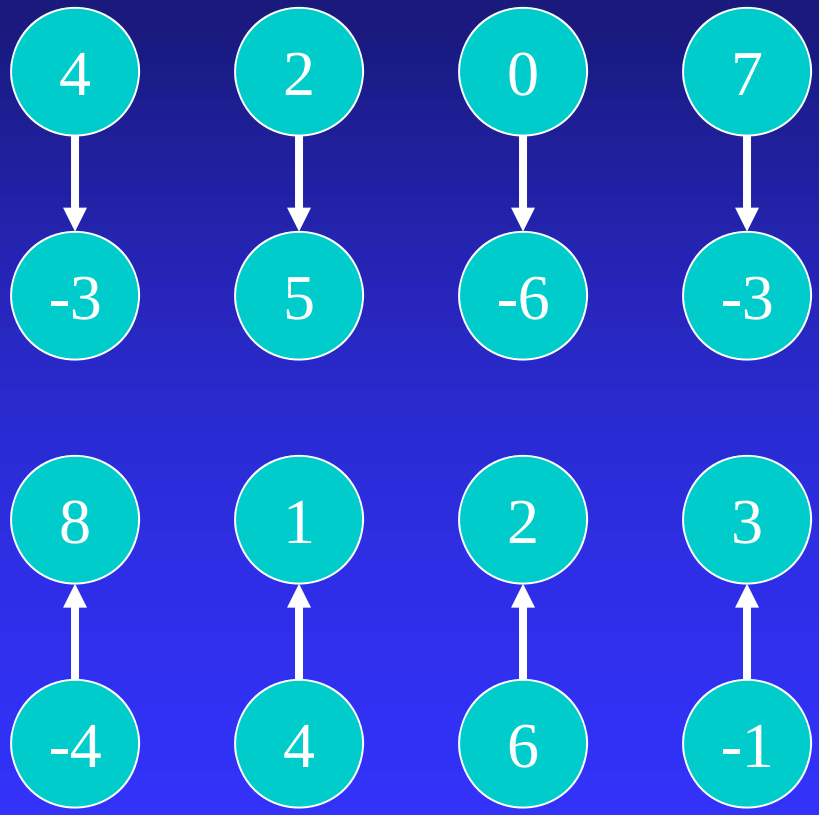
- Given associative operator $\oplus$
- $a_0 \oplus a_1 \oplus a_2 \oplus \ldots \oplus a_{n-1}$
- Examples
  - Add
  - Multiply
  - And, Or
  - Maximum, Minimum

# Binomial Trees

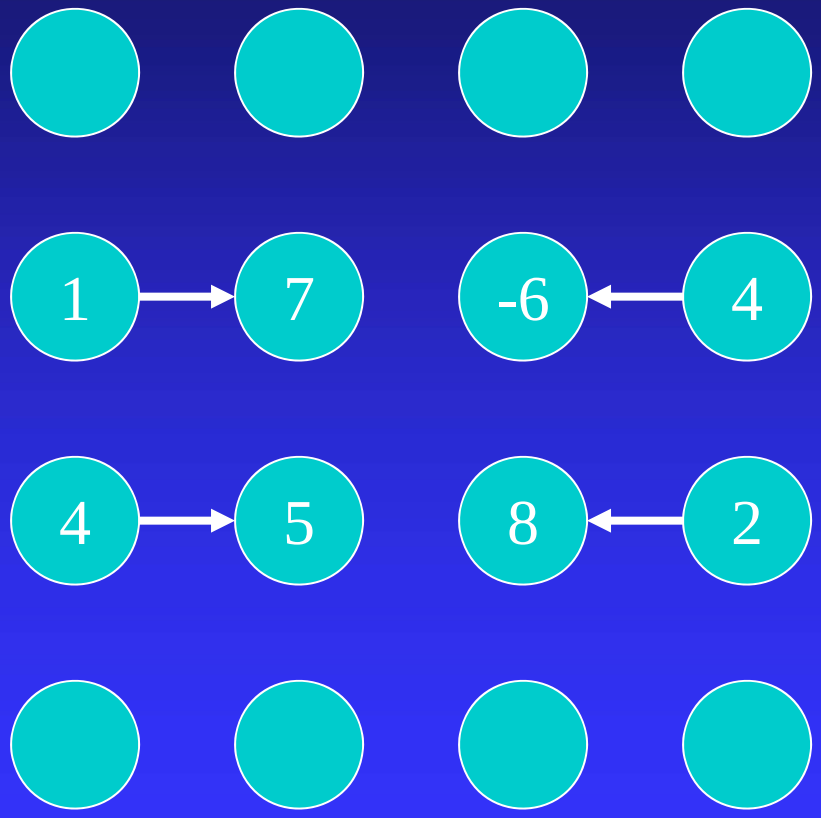


Subgraph of hypercube

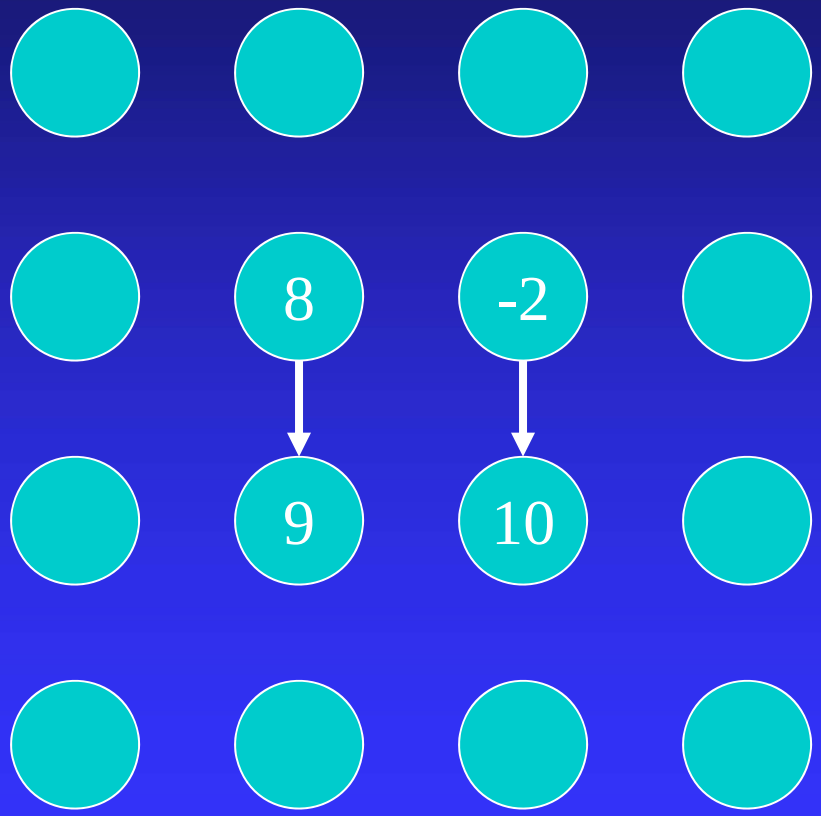
# Finding Global Sum



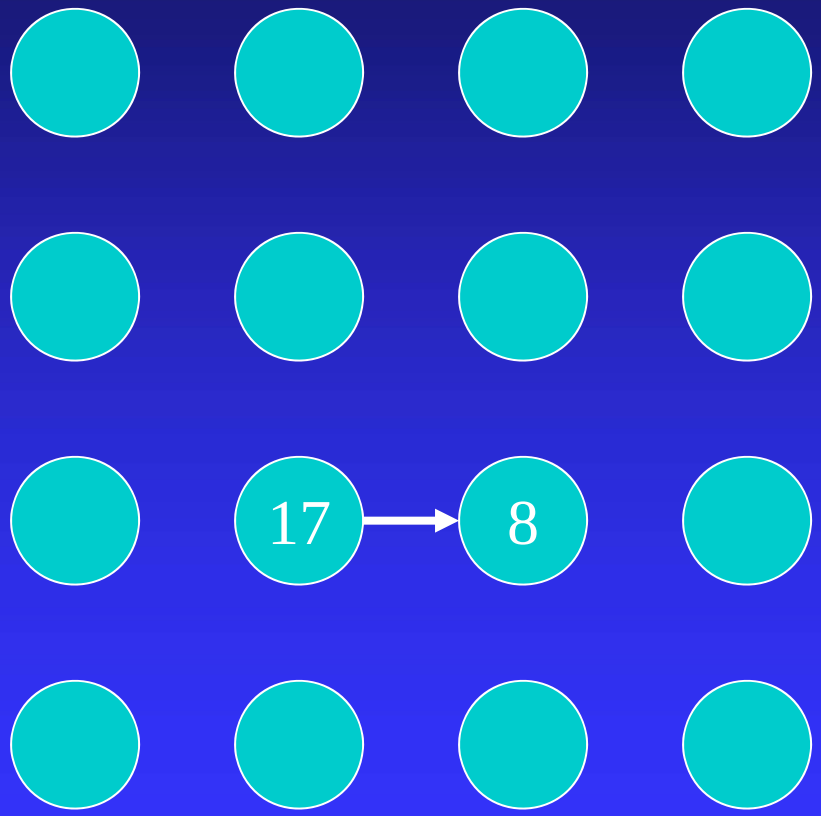
# Finding Global Sum



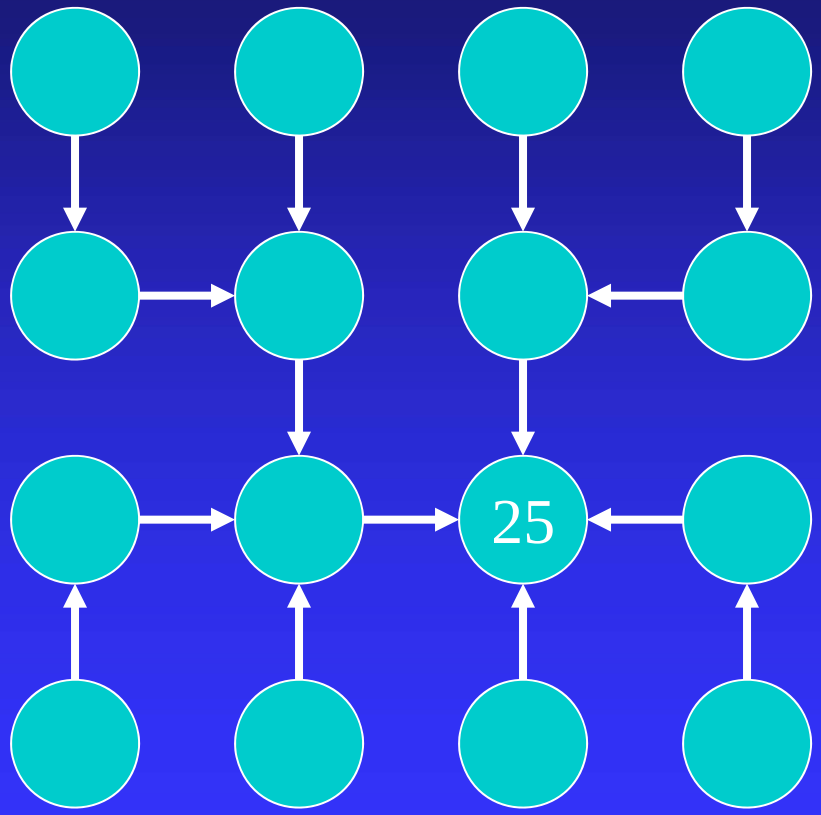
# Finding Global Sum



# Finding Global Sum

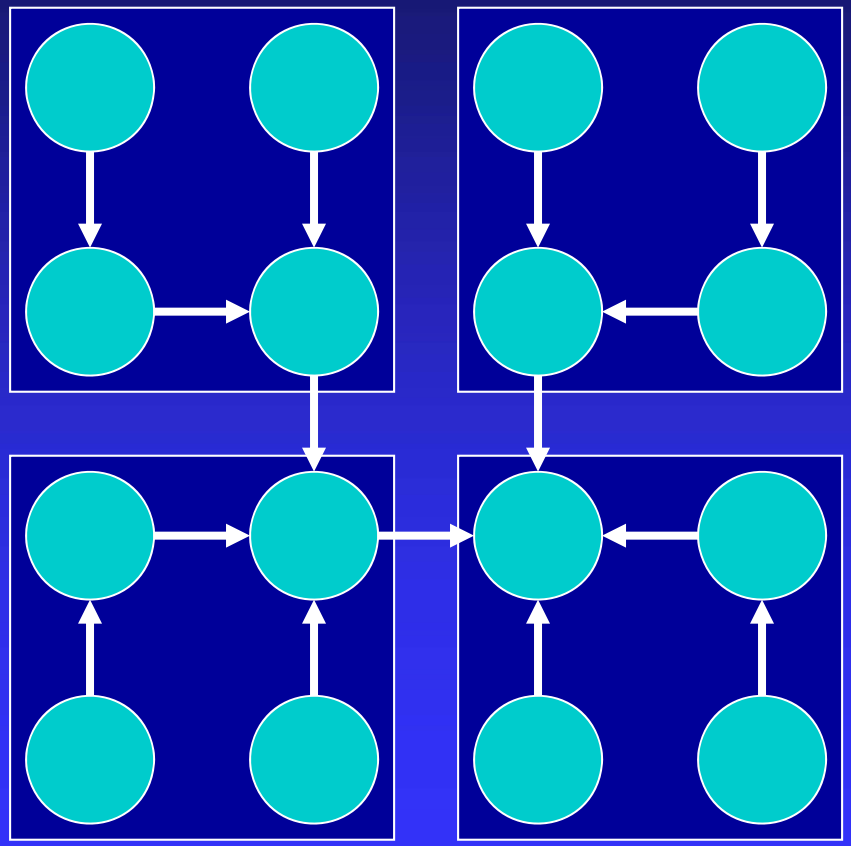


# Finding Global Sum

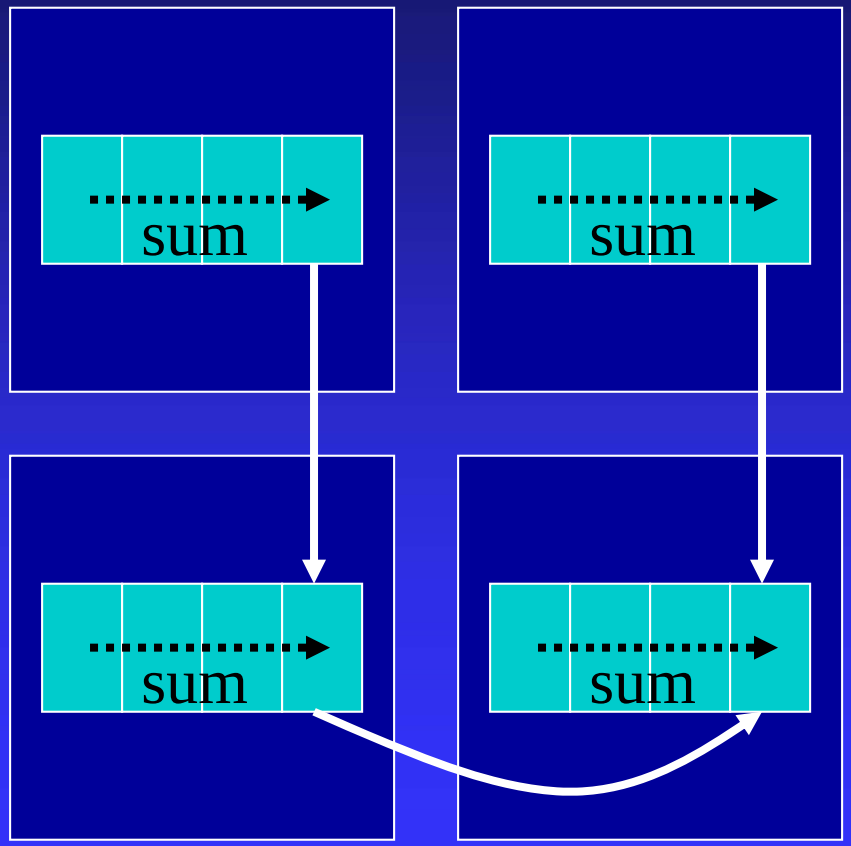



Binomial Tree

# Agglomeration



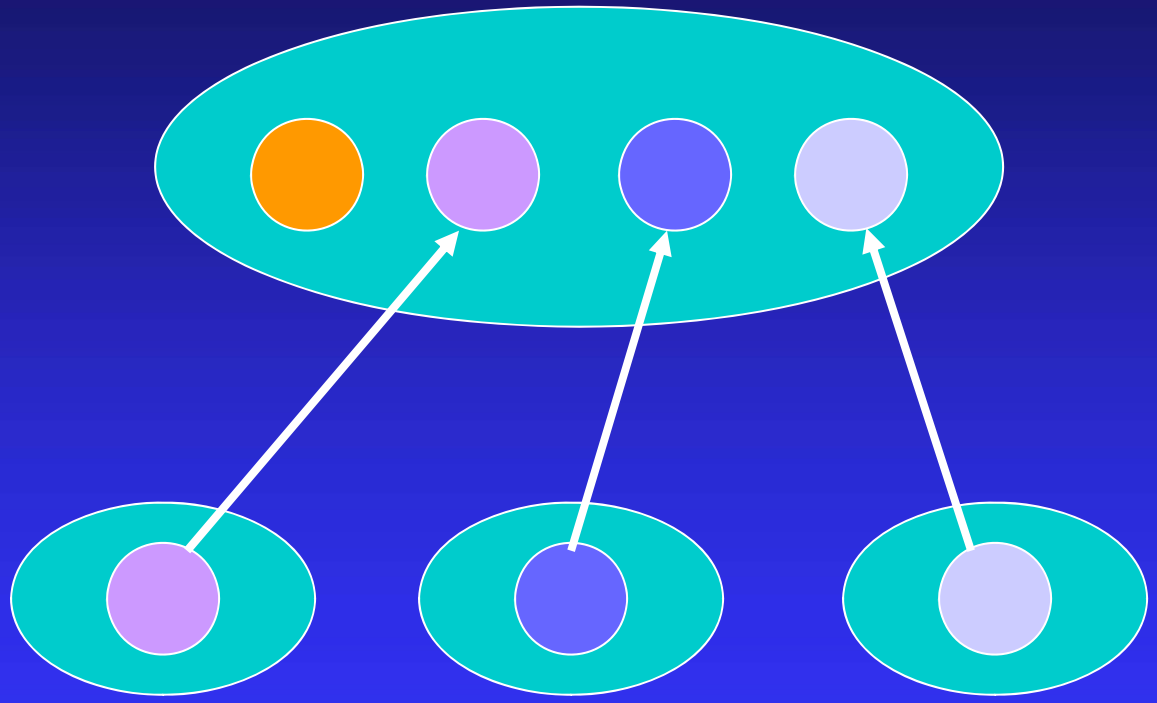
# Agglomeration



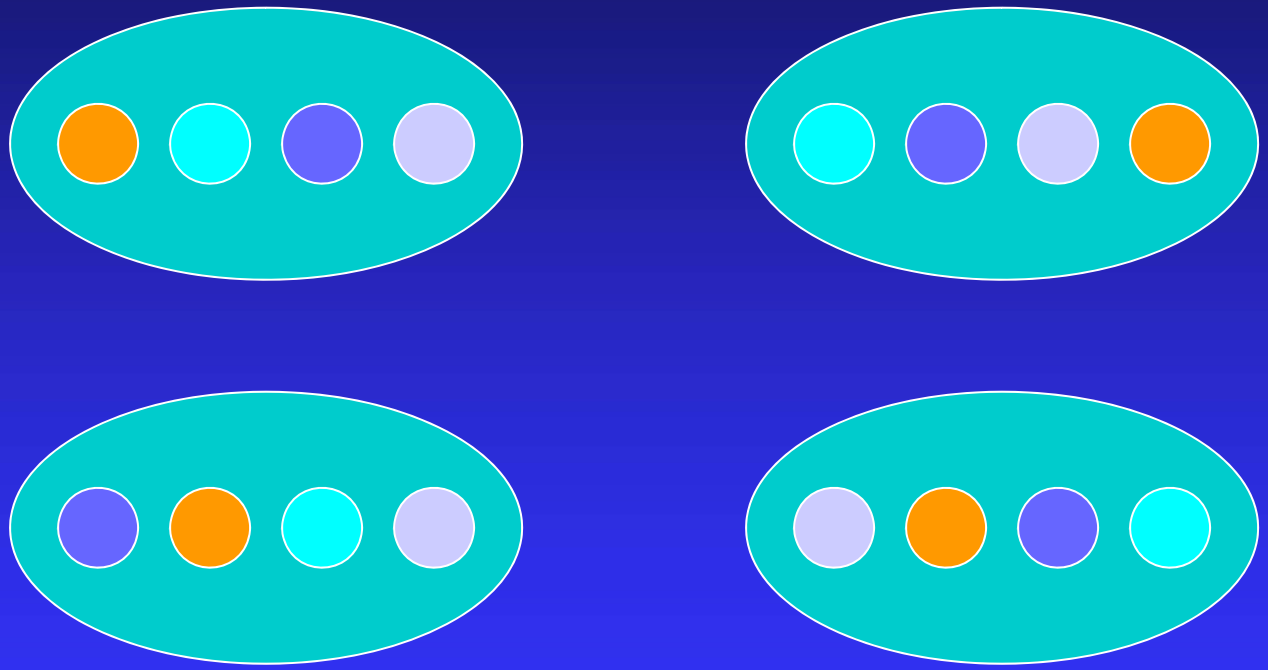
# Partitioning

- 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

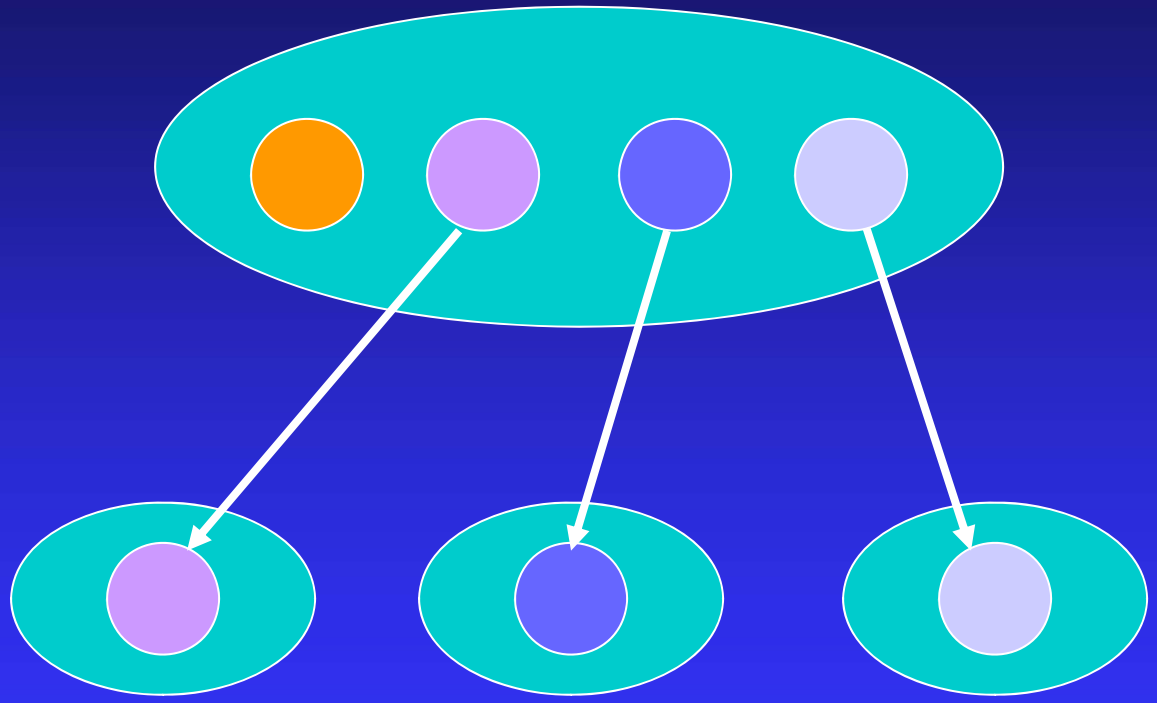
# Gather



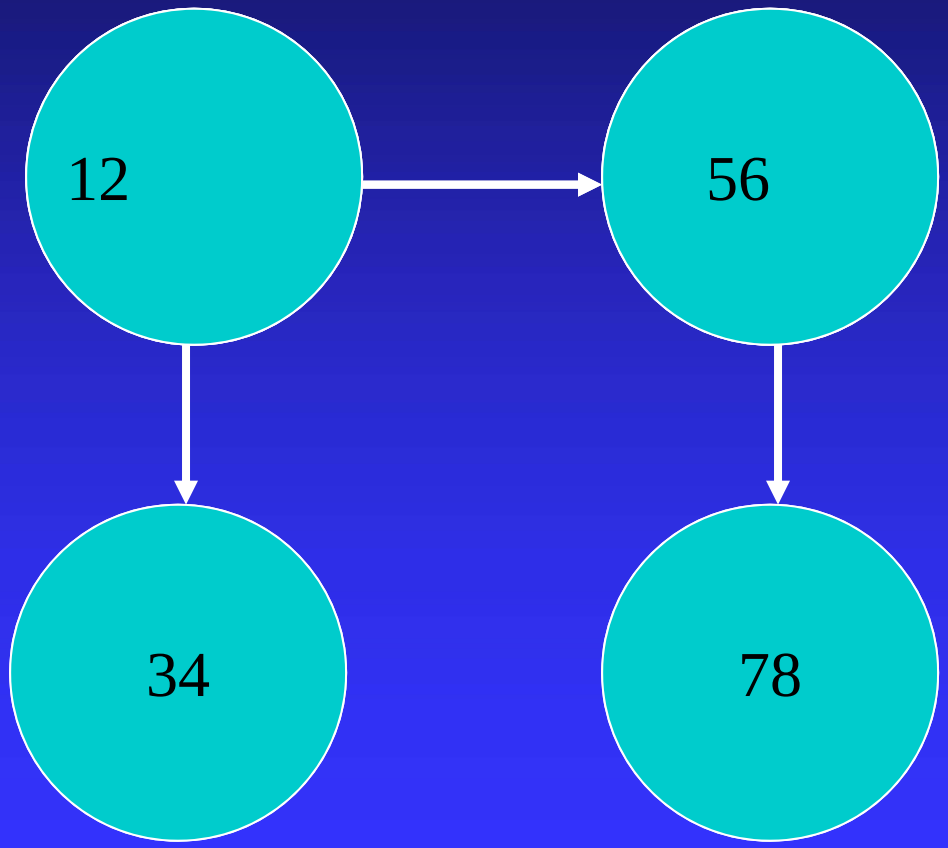
# All-gather



# 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