Parallel Algorithm Design

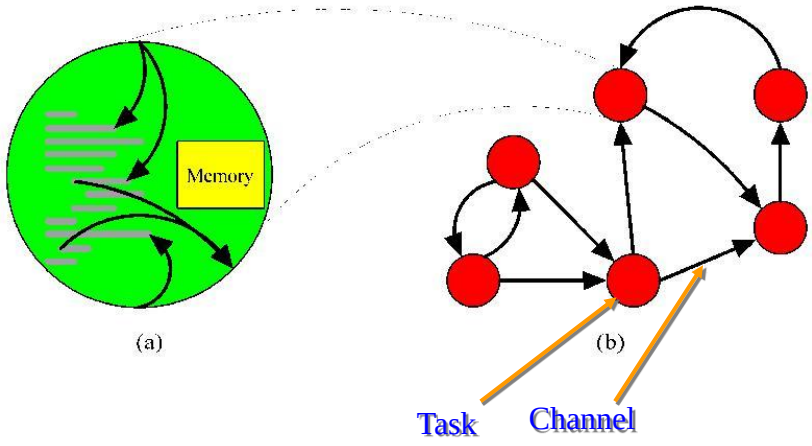
Outline

- 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
- At input port task must wait until the value appears, means task is **blocked**
- In this model receiving is a **synchronous,** while sending is an **asynchronous** operation

Task/Channel Model



Source: M.J. Quinn

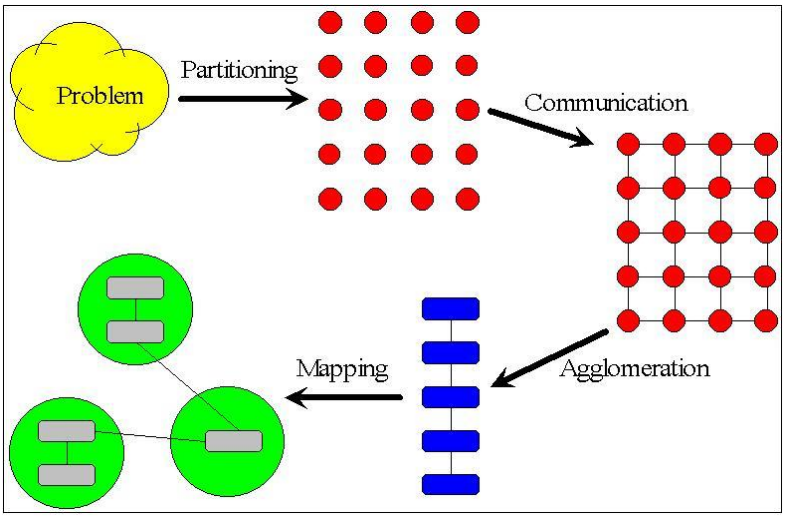
Foster's Design Methodology*

- Partitioning
- Communication (Concentration on inherent parallelism)
- Agglomeration
- Mapping (Concentration on implementation on real HW)

"It delays the machine dependent considerations at the later stage "

* Foster, Ian. *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*, Reading, MA: Addison-Wesley, 1995

Foster's Methodology



Source: M.J. Quinn

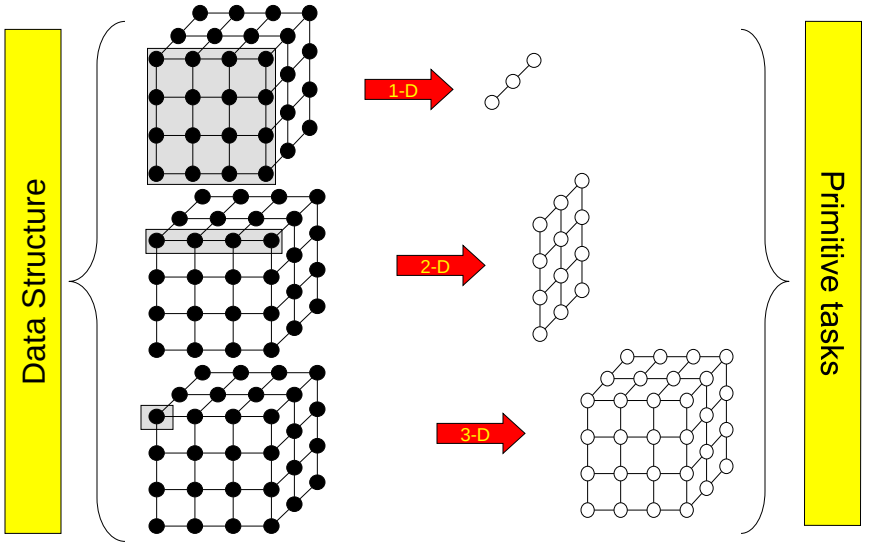
1.Partitioning

- 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

1.Partitioning

- Focus is on most frequently accessed data structure
- In the matrix, we can partition data into
  - Collection of 2-D slice: resulting in a 1-D collection of primitive tasks
  - Collection of 1-D slice: resulting in a 2-D collection of primitive tasks
  - Consider each elements of the matrix individually: 3-D collection of primitive tasks

Domain Decompositions



Source: M.J. Quinn

1.Partitioning Checklist

- At least 10x more primitive tasks than processors in target computer
- Minimize redundant computations and redundant data storage (If not, design does not work well when problem size increases)
- Primitive tasks roughly the same size (If not, load balancing problem)
- Number of tasks an increasing function of problem size (If not, not scale well)

2.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

2.Communication Checklist

Communications are overhead in parallel algorithms, minimizing them is an important goal

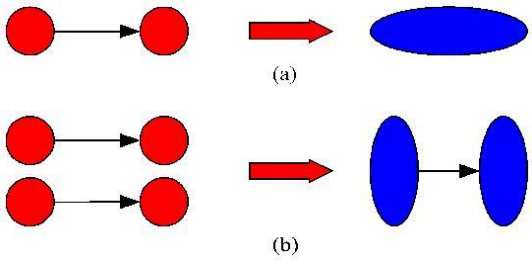
- Communication operations balanced among tasks
- Each task communicates with only small group of neighbors
- Tasks can perform communications concurrently
- Task can perform computations concurrently

3.Agglomeration

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

Agglomeration Can Improve Performance

- Eliminate communication between primitive tasks agglomerated into consolidated task (specially when the tasks cannot perform their operations in parallel)
- Combine groups of sending and receiving tasks (for reducing message latency)

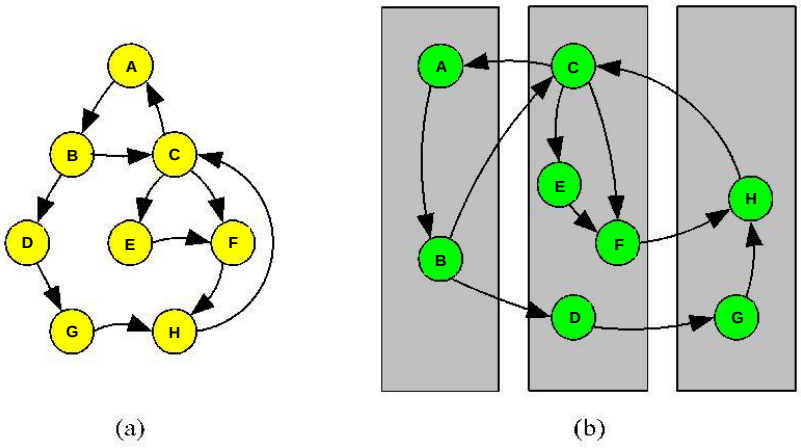


Source: M.J. Quinn

4.Mapping

- Process of assigning tasks to processors
- *Centralized multiprocessor: mapping done by operating system*
- *Distributed memory system: mapping done by user*
- Goals of mapping
  - Maximize processor utilization
  - Minimize inter-processor communication

Mapping Example



Source: M.J. Quinn

Optimal Mapping

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

Definition

- In *local* communication, each task communicates with a small set of other tasks (its ``neighbors''); in contrast, *global* communication requires each task to communicate with many tasks.
- In *structured* communication, a task and its neighbors form a regular structure, such as a tree or grid; in contrast, *unstructured* communication networks may be arbitrary graphs.
- In *static* communication, the identity of communication partners does not change over time; in contrast, the identity of communication partners in *dynamic* communication structures may be determined by data computed at runtime and may be highly variable.
- In *synchronous* communication, producers and consumers execute in a coordinated fashion, with producer/consumer pairs cooperating in data transfer operations; in contrast, *asynchronous* communication may require that a consumer obtain data without the cooperation of the producer.

Mapping Decision Tree

- Partition is done using domain decomposition, tasks are small, communication is regular, create p agglomerated tasks on p processors
- 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
      - First run a static load balancing algorithm before determining the strategy
- Dynamic number of tasks

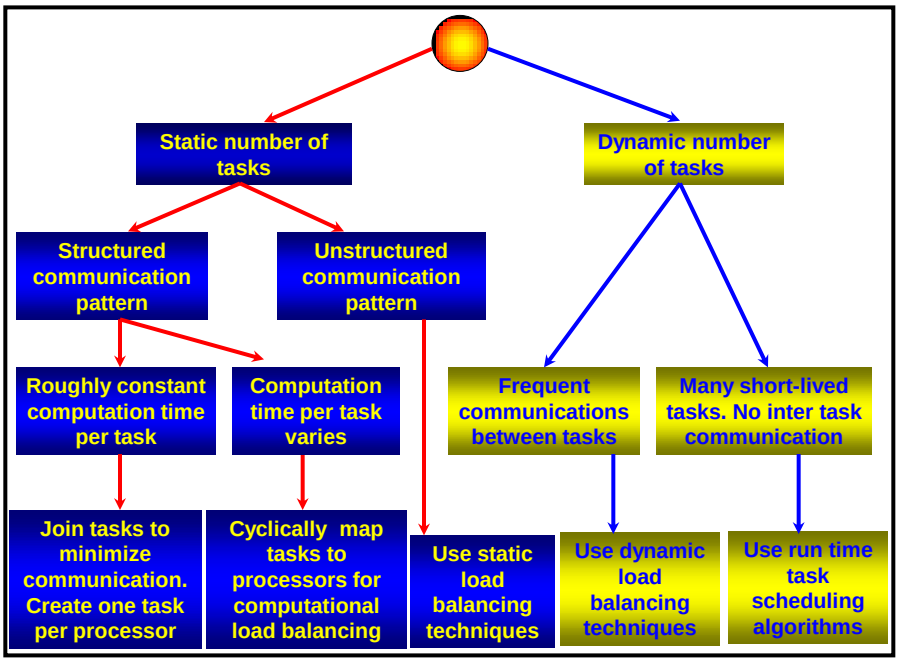
Mapping Strategy

- 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

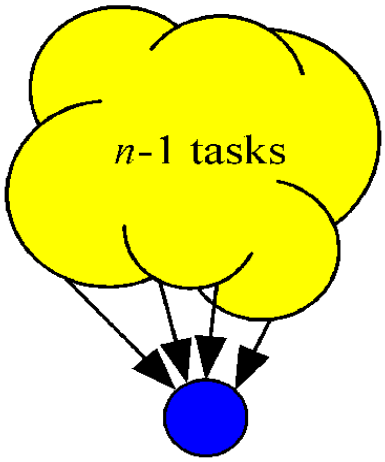
Decision tree to choose a mapping strategy



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

Parallel Reduction Evolution

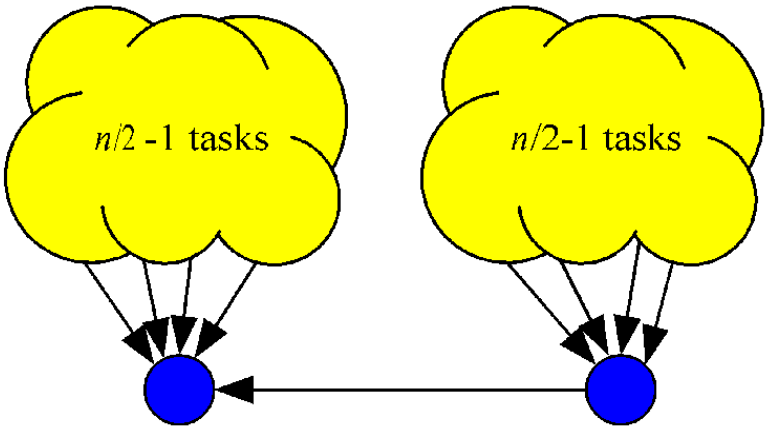


Observations:

1. If ***x*** time required for a task to communicate another task
2. ***y*** time is required for addition then
3. Total time will be (n-1)(x+y)

Source: M.J. Quinn

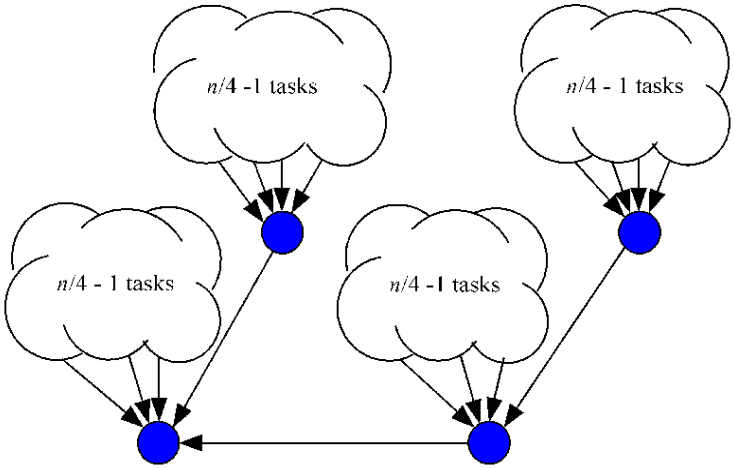
Parallel Reduction Evolution



Observations: Total time will be now (n/2-1)(x+y)

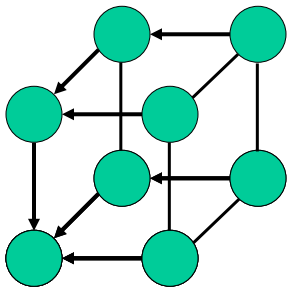
Source: M.J. Quinn

Parallel Reduction Evolution



Source: M.J. Quinn

Binomial Trees

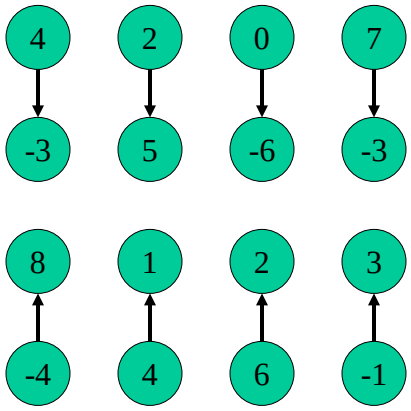


Subgraph of hypercube

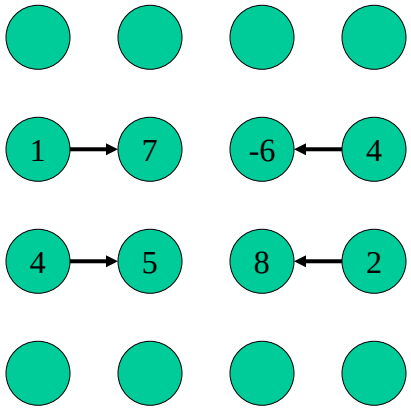
Observations:

Continuing this way we have n/2 semi-root tasks
Total time will be ***log n***

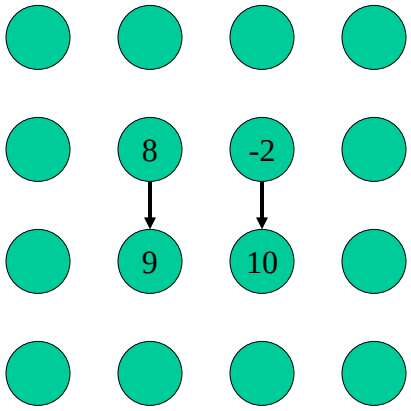
Finding Global Sum



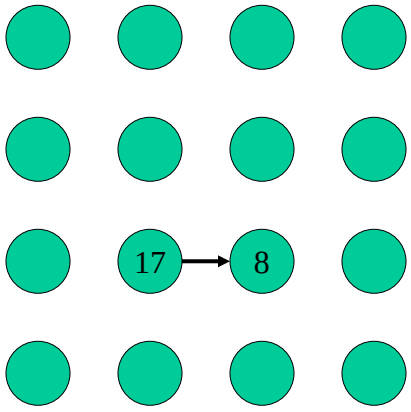
Finding Global Sum



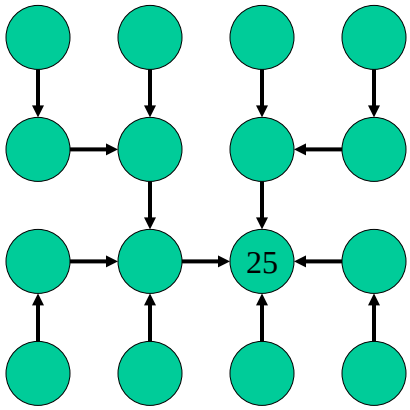
Finding Global Sum



Finding Global Sum

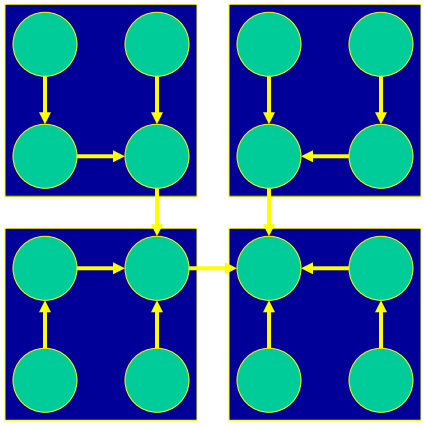


Finding Global Sum



Binomial Tree

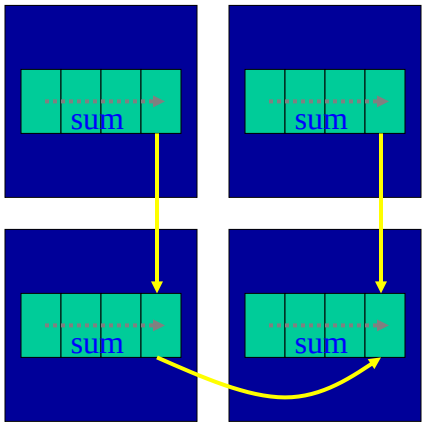
Agglomeration



Observations:

1. Number of tasks are static
2. Computations per task are trivial
3. Communication pattern is regular

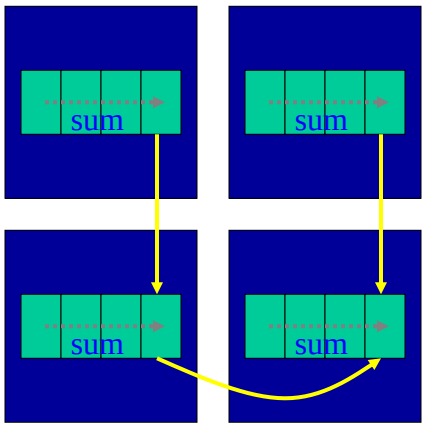
Agglomeration



Observations:

1. If ***x*** time required for a task to communicate another task
2. ***y*** time is required for binary operation then
3. Total time will be (n-1)(x+y)

Agglomeration

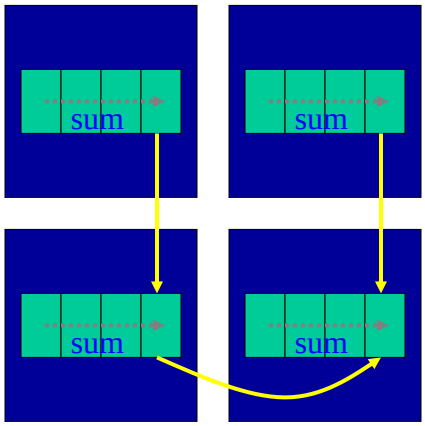


Observations:

1. If we have ***p*** processors
2. Time require to compute sub total is $(\lceil n/p \rceil - 1)y$
3. Reduction require x+y time
4. With $\lceil \log p \rceil$ communications overall time

Agglomeration

Total time:



$$(\lceil n/p \rceil - 1)y + \lceil \log p \rceil (x+y)$$