COMS4040A&COMS7045A: High Performance & Scientific Data Management Lecture 1: Parallel Computers and Parallel Algorithm Design

Hairong Wang

School of Computer Science & Applied Mathematics
University of the Witwatersrand, Johannesburg

2019-2-7

WITS
UNIVERSITY

Contents

1. Objectives
2. Parallel computers
   - Parallel Computing — What is it?
3. Discussion
   - Supercomputers
4. Classification of Parallel Computers
   - Control Structure Based Classification
5. A parallel programming model
6. Foster's Design Methodology
7. Parallel Algorithm Examples

Outline

1. Objectives
2. Parallel computers
    - Parallel Computing — What is it?
3. Discussion
    - Supercomputers
4. Classification of Parallel Computers
    - Control Structure Based Classification
5. A parallel programming model
6. Foster’s Design Methodology
7. Parallel Algorithm Examples

Objectives

- Understand the basics of task/channel model.
- Design and analyse parallel algorithm using Foster's design methodology.

Outline

1. Objectives
2. Parallel computers
    - Parallel Computing — What is it?
3. Discussion
    - Supercomputers
4. Classification of Parallel Computers
    - Control Structure Based Classification
5. A parallel programming model
6. Foster's Design Methodology
7. Parallel Algorithm Examples

WITS
UNIVERSITY

Outline

1. Objectives
2. Parallel computers
   - Parallel Computing — What is it?
3. Discussion
   - Supercomputers
4. Classification of Parallel Computers
   - Control Structure Based Classification
5. A parallel programming model
6. Foster's Design Methodology
7. Parallel Algorithm Examples

Parallel computing

- Parallel Computer: A parallel computer is a computer system that uses multiple processing elements simultaneously in a cooperative manner to solve a computational problem.
- Parallel computing (or processing): Parallel processing includes techniques and technologies that make it possible to compute in parallel.
    - Hardware, networks, operating systems, parallel libraries, languages, compilers, algorithms, tools etc.
- Parallel computing is an evolution of serial computing.
    - Parallelism is natural.
    - Computing problems differ in level or type of parallelism.

Serial computing

- A problem is broken into a discrete series of instructions;
- Instructions are executed sequentially one after another;
- Executed on a single processor;
- Only one instruction may execute at any moment in time.

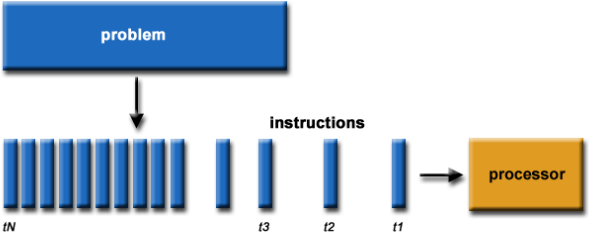


Figure: Serial computing

Parallel computing

- A problem is broken into discrete parts that can be solved concurrently;
- Each part is further broken down to a series of instructions;
- Instructions from each part execute simultaneously on different processors;
- An overall control/coordination mechanism is employed.

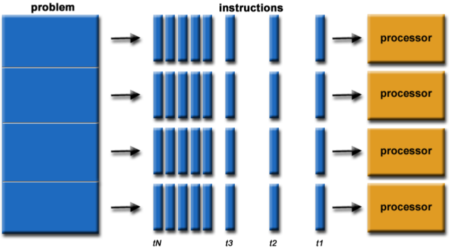


Figure: Parallel computing

Parallel computing cont.

An example of parallelizing an addition of two vectors is shown in Figure 3.

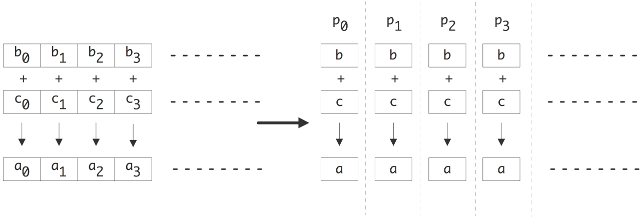


Figure: Parallelization of a vector addition

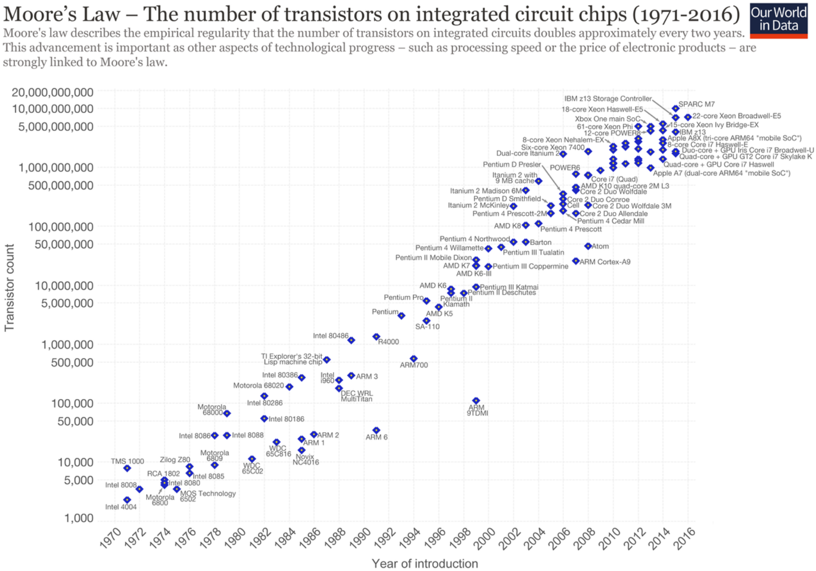
Outline

1. Objectives
2. Parallel computers
   - Parallel Computing — What is it?
3. Discussion
   - Supercomputers
4. Classification of Parallel Computers
   - Control Structure Based Classification
5. A parallel programming model
6. Foster's Design Methodology
7. Parallel Algorithm Examples

WITS
UNIVERSITY

Why parallel computing?

- Why parallel computing?



Moore's Law – The number of transistors on integrated circuit chips (1971-2016)

Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important as other aspects of technological progress – such as processing speed or the price of electronic products – are strongly linked to Moore's law.

Why parallel computing? cont.

- In the past, processing chip manufacturers increased processor performance by increasing CPU clock frequency, until the chips got too hot! Greater clock frequency, greater electrical power.
- Add multiple cores to add performance, keep clock frequency same or reduced.
    - The performance gap of processor and memory widens.

Outline

1. Objectives
2. Parallel computers
   - Parallel Computing — What is it?
3. Discussion
   - Supercomputers
4. Classification of Parallel Computers
   - Control Structure Based Classification
5. A parallel programming model
6. Foster's Design Methodology
7. Parallel Algorithm Examples

No 1 Supercomputer in 11/2018

Site	DOE/SC/Oak Ridge National Laboratory
Manufacturer	IBM
Cores	2,397,824
Linpack Performance (Rmax)	143,500 TFlop/s
Theoretical Peak (Rpeak)	200,795 TFlop/s
HPCG	2,925.75 TFlop/s
Nmax	16,693,248
Power	9,783.00 kW (Submitted)
Memory	2,801,664 GB
Processor	IBM POWER9 22C 3.07GHz
Interconnect	Dual-rail Mellanox EDR Infiniband
Operating System	RHEL 7.4
Compiler	IBM XLC, nvcc
Math library	ESSL, CUBLAS 9.2
MPI	Spectrum MPI

No 1 Supercomputer in 11/2016 and 11/2017

Site	National Supercomputing Center in Wuxi
Manufacturer	NRCPC
Cores	10,649,600
Linpack Performance (Rmax)	93,014.6 TFlop/s
Theoretical Peak (Rpeak)	125,436 TFlop/s
Nmax	12,288,000
Power	15,371.00 kW (Submitted)
Memory	1,310,720 GB
Processor	Sunway SW26010 260C 1.45GHz
Interconnect	Sunway
Operating System	Sunway RaiseOS 2.0.5

Lengau from CHPC SA — Rank 161 (11/2016)

Site	Centre for High Performance Computing
System URL	`http://www.chpc.ac.za/`
Manufacturer	Dell
Cores	24,192
Linpack Performance (Rmax)	782.886 TFlop/s
Theoretical Peak (Rpeak)	1,006.39 TFlop/s
Nmax	3,836,736
Power	540.00 kW (Submitted)
Memory	129,024 GB
Processor	Xeon E5-2690v3 12C 2.6GHz
Interconnect	Infiniband FDR
Operating System	CentOS
Compiler	Intel 16.0.1.150
Math Library	Intel MKL 11.3.1.150
MPI	Intel MPI 5.1.2.150

Lengau from CHPC SA — Rank 400 (11/2018)

Site	Centre for High Performance Computing
System URL	`http://www.chpc.ac.za/`
Manufacturer	Dell
Cores	32,856
Linpack Performance (Rmax)	1,029.3 TFlop/s
Theoretical Peak (Rpeak)	1,366.8 TFlop/s
Nmax	3,105,408
Power	685 kW (Submitted)
Memory	175,232 GB
Processor	Xeon E5-2690v3 12C 2.6GHz
Interconnect	Infiniband FDR
Operating System	CentOS
Compiler	Intel
Math Library	Intel MKL
MPI	Intel MPI

The same machine was ranked 165 in 11/2017

The Clusters at Mathematical Sciences Lab

- NVIDIA Cluster
  - 100 nodes
  - Each node has an i7 (with 4 cores), and 4GB RAM
  - Each node has a GPU750ti (640 cores, 2GB RAM),
  - Connected by Ethernet
  - OS: Ubuntu

The Clusters at Mathematical Sciences Lab

- NVIDIA Cluster
  - 100 nodes
  - Each node has an i7 (with 4 cores), and 4GB RAM
  - Each node has a GPU750ti (640 cores, 2GB RAM),
  - Connected by Ethernet
  - OS: Ubuntu
- A Sun server
  - 48 nodes, each has 16 cores
  - each node has 32GB RAM
  - Can only be used in sets of 12 nodes, where each set is connected via Infiniband
  - OS: Centos

The Clusters at Mathematical Sciences Lab

- NVIDIA Cluster
  - 100 nodes
  - Each node has an i7 (with 4 cores), and 4GB RAM
  - Each node has a GPU750ti (640 cores, 2GB RAM),
  - Connected by Ethernet
  - OS: Ubuntu
- A Sun server
  - 48 nodes, each has 16 cores
  - each node has 32GB RAM
  - Can only be used in sets of 12 nodes, where each set is connected via Infiniband
  - OS: Centos
- 2 Xeon Phi servers, one for development, the other for production
  - Each Phi has 60 cores
  - 32 GB RAM each
  - OS: Centos

WITS UNIVERSITY

Outline

1. Objectives
2. Parallel computers
   - Parallel Computing — What is it?
3. Discussion
   - Supercomputers
4. Classification of Parallel Computers
   - Control Structure Based Classification
5. A parallel programming model
6. Foster's Design Methodology
7. Parallel Algorithm Examples

Outline

1. Objectives
2. Parallel computers
   - Parallel Computing — What is it?
3. Discussion
   - Supercomputers
4. Classification of Parallel Computers
   - Control Structure Based Classification
5. A parallel programming model
6. Foster's Design Methodology
7. Parallel Algorithm Examples

WITS UNIVERSITY

Flynn's taxonomy

Flynn's taxonomy is widely used since 1966 for classification of parallel computers. The classification is based on two independent dimensions of *instruction stream* and *data stream* with two possible states: *single* or *multiple*.

- SISD: Single instruction stream single data stream. This is the traditional CPU architecture: at any one time only a single instruction is executed, operating on a single data item.

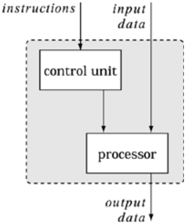


Figure: The SISD architecture

SIMD

- SIMD: Single instruction stream multiple data stream. In this computer type there can be multiple processors, each operating on its own data item, but they are all executing the same instruction on that data item. SIMD computers excel at operations on arrays, such as

$$for(i = 0; i < N; i++) \quad a[i] = b[i] + c[i];$$

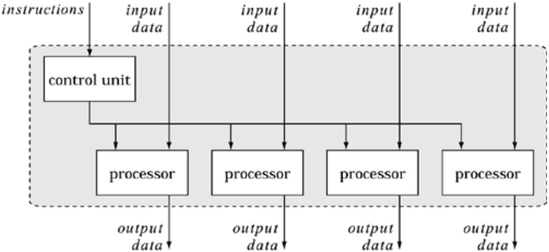


Figure: The SIMD architecture

MIMD

- MISD: Multiple instruction stream single data stream. Each processing unit executes different instruction streams on a single data stream. Very few computers are in this type.
- MIMD: Multiple instruction stream multiple data stream. Multiple processors operate on multiple data items, each executing independent, possibly different instructions. Most current parallel computers are of this type.

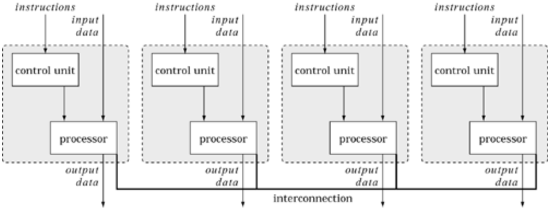


Figure: The MIMD architecture

MIMD cont.

- Most of MIMD machines operate in *single program multiple data* (SPMD) mode, where the programmers starts up the same executable on the parallel processors.

A Further Decomposition of MIMD

The MIMD category is typically further decomposed according to memory organization: shared memory and distributed memory.

- **Shared memory:** In a shared memory system, all processes share a single address space and communicate with each other by writing and reading shared variables.
- One class of shared-memory systems is called SMPs(symmetric multiprocessors).

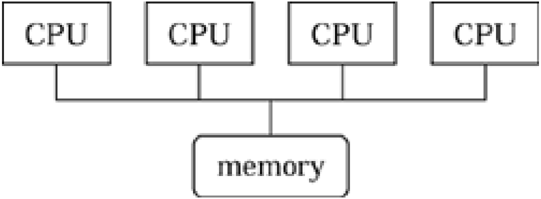


Figure: The SMP architecture

NUMA

- The other main class of shared-memory systems is called non-uniform memory access (NUMA). The memory is shared, it is uniformly addressable from all processors, but some blocks of memory may be physically more closely associated with some processors than others.
- To mitigate the effects of non-uniform access, each processor has a cache, along with a protocol to keep cache entries coherent — cache-coherent non-uniform memory access systems (ccNUMA).

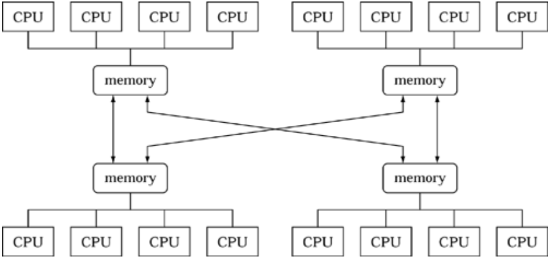


Figure: The NUMA architecture

Distributed memory systems

- Each process has its own address space and communicates with other processes by message passing (sending and receiving messages).

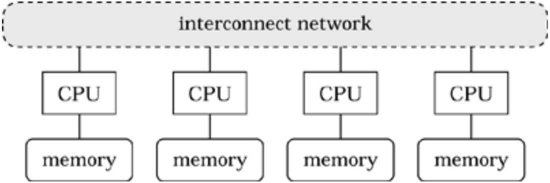


Figure: The distributed memory architecture

Clusters

- Clusters are distributed memory systems composed of off-the-shelf computers connected by an off-the-shelf network.

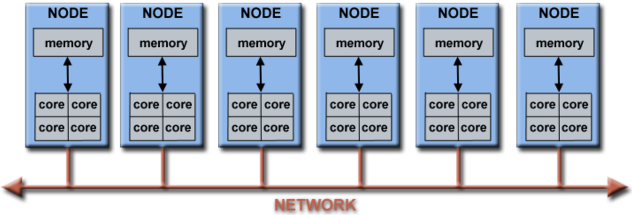


Figure: A cluster

Outline

1. Objectives
2. Parallel computers
   - Parallel Computing — What is it?
3. Discussion
   - Supercomputers
4. Classification of Parallel Computers
   - Control Structure Based Classification
5. A parallel programming model
6. Foster's Design Methodology
7. Parallel Algorithm Examples

Task/Channel model

- What kind of abstraction can we use for parallel programming?
- Task/channel model represents a parallel computation as a set of tasks that may interact with each other by sending messages through channels.

Task: a program, its local memory, and a collection of I/O ports.

Channel: a message queue that connects one task's output port with another task's input port.

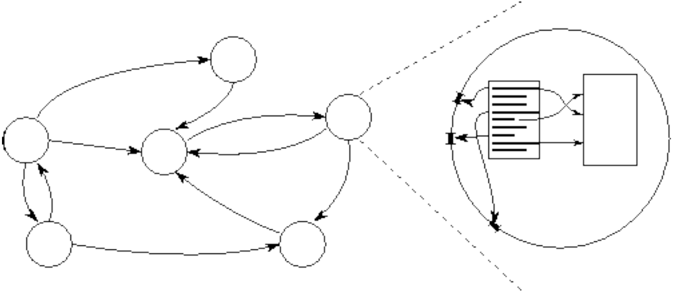


Figure: A task/channel parallel programming model.

Some properties of Task/Channel model

- A parallel computation consists of one or more tasks.
- A task encapsulates a sequential program and local memory.
- A task can perform actions such as reading and writing its local memory; send and receive messages; create and terminate tasks.
- Tasks can be connected through channels, and channels can be created and deleted dynamically.
- Tasks can be mapped to physical processors in various ways.

Outline

1. Objectives
2. Parallel computers
   - Parallel Computing — What is it?
3. Discussion
   - Supercomputers
4. Classification of Parallel Computers
   - Control Structure Based Classification
5. A parallel programming model
6. Foster's Design Methodology
7. Parallel Algorithm Examples

WITS UNIVERSITY

Foster’s Four-Step Design Methodology

Partitioning: Decompose problem into fine-grain tasks, maximizing number of tasks that can execute concurrently.

Communication: Determine communication pattern among fine-grain tasks, yielding task graph with fine-grain tasks as nodes and communication channels as edges

Agglomeration: Combine groups of fine-grain tasks to form fewer but larger coarse-grain tasks, thereby reducing communication requirements

Mapping: Assign coarse-grain tasks to processors, subject to tradeoffs between communication costs and concurrency

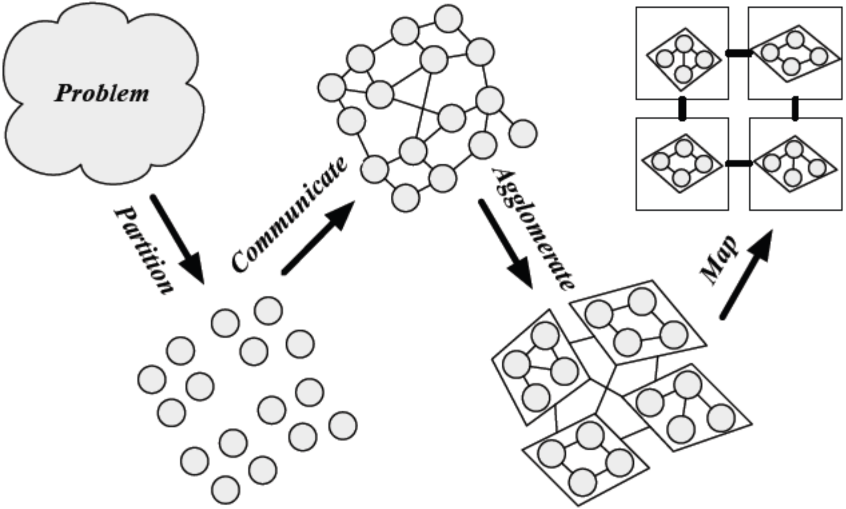


Figure: Foster's parallel algorithm design methodology.

Partitioning

- Domain Decomposition – data decomposition
- Functional Decomposition – computation decomposition

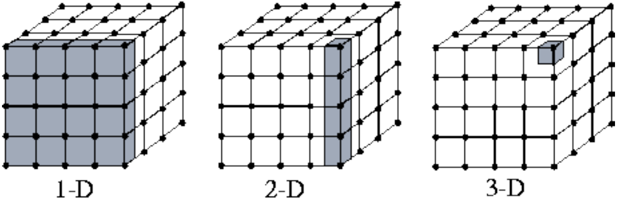


Figure: Domain decomposition of a problem involving a 3D grid

Partitioning Example

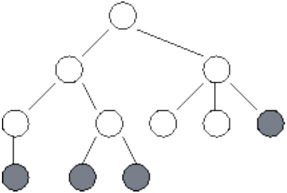


Figure: Functional decomposition of a problem involving recursive search. (We assume the algorithm does not have a regular data structure for data decomposition.) Each node in the search tree represents a call to the search procedure. A task is created for each node in the tree as it is explored. At any one time, some tasks are actively engaged in expanding the tree further (these are shaded in the figure); others have reached solution nodes and are terminating, or are waiting for their offspring to report back with solutions. The lines represent the channels used to return solutions.

Partitioning Outcome Evaluation

The partitioning should produce one or more possible decompositions of a problem. Desirable properties of partitioning

- Maximum possible concurrency in executing resulting tasks
- Many more tasks than processors
- Redundant computation or storage avoided
- Tasks reasonably uniform in size
- Number of tasks, rather than size of each task, grows as overall problem size increases
- Alternative partitions

Communication

The tasks generated by a partition are intended to execute concurrently but cannot, in general, execute independently – data dependence. Communication pattern may be

- local or global
- structured or random
- persistent or dynamically changing
- synchronous or sporadic

Communication Examples

- Local communication - communication occurs within a small number of tasks.

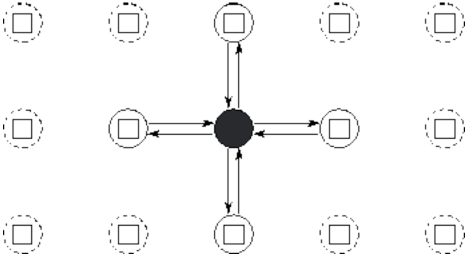


Figure: Task and channel structure for a two-dimensional finite difference computation with five-point update stencil.

Communication Examples

- Global communication - The majority of the tasks participate in the communication in order to complete a computation.

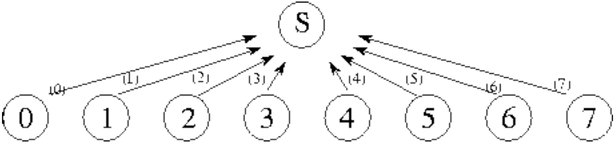


Figure: A centralized summation algorithm that uses a central manager task (S) to sum N numbers distributed among N tasks.

Communication Examples

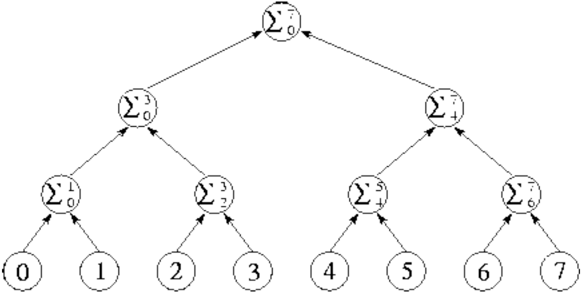


Figure: Tree structure for divide-and-conquer summation algorithm with N=8.

Communication Outcome Evaluation

Communication among tasks is the overhead of a parallel algorithm. Minimizing this overhead is an important goal of parallel algorithm design.

- all tasks perform about the same number of communication operations
- each task communicates only with a small number of neighbors
- communication operations able to proceed concurrently
- overlapped with computation as much as possible

Agglomeration

Combine, or agglomerate, tasks identified by the partitioning phase, so as to provide a smaller number of tasks, each of greater size.

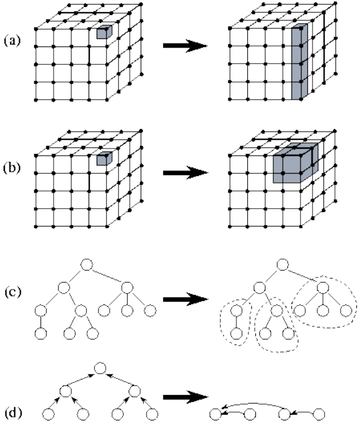


Figure: Examples of agglomeration.

Agglomeration

- Improve performance
- Maintain scalability of program
- Simplify programming – i.e. reduce software engineering costs.

Agglomeration

Guidelines concerning agglomeration

- Increasing Granularity
- Preserving Flexibility
- Reducing Software Engineering Costs

Agglomeration Examples

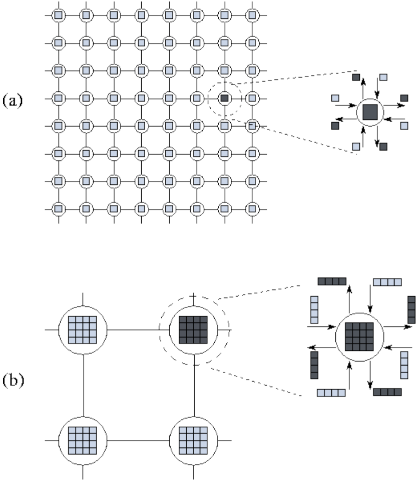
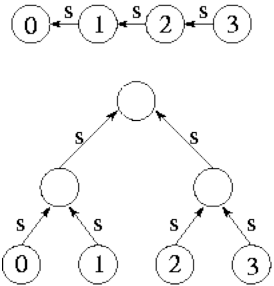


Figure: Effect of increased granularity on communication costs.

Agglomeration Examples



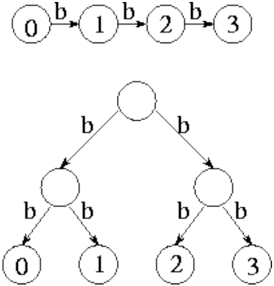


Figure: Using an array and a tree to perform a summation and a broadcast.

Agglomeration Examples

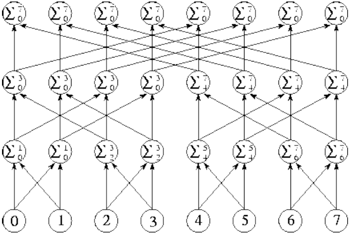

Σ_0^7 Σ_0^7 Σ_0^7 Σ_0^7 Σ_0^7 Σ_0^7 Σ_0^7 Σ_0^7
Σ_0^3 Σ_0^3 Σ_0^3 Σ_0^3 Σ_4^7 Σ_4^7 Σ_4^7 Σ_4^7
Σ_0^1 Σ_0^1 Σ_2^3 Σ_2^3 Σ_4^5 Σ_4^5 Σ_6^7 Σ_6^7
0 1 2 3 4 5 6 7

Figure: The butterfly communication structure can be used to sum N values.

Agglomeration Examples

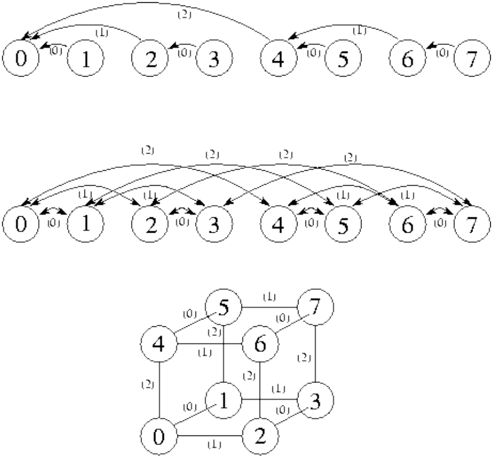


Figure: The communication structures that result when tasks at different levels in a tree or butterfly structure are agglomerated.

Agglomeration Outcome Evaluation

- Locality of parallel algorithm has increased
- Replicated computations take less time than communications they replace
- Data replication doesn't affect scalability
- All the 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

- Specify where each task is to execute.
- The goal of mapping: reduce the execution time.
- How? increase the concurrency; reduce the communication (or increase the locality).
- The mapping problem is known to be NP -complete, use specialized techniques or heuristics.
- But connectivity of coarse-grain task graph is inherited from that of fine-grain task graph, whereas connectivity of target interconnection network is independent of problem
- Communication channels between tasks may or may not correspond to physical connections in underlying interconnection network between processors

Mapping Techniques

- Cyclic mapping
- Block cyclic mapping

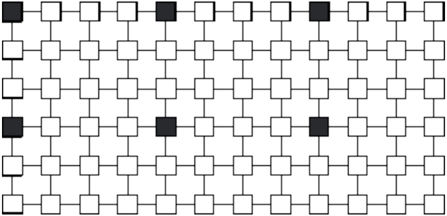


Figure: Cyclic mapping.

Mapping

- Optimal mapping
  - Optimality is with respect to processor utilization and interprocessor communication.
  - It is the interaction of the processor utilization and communication that is important.
- Mapping outcome evaluation
  - Consider designs based on one task per processor and multiple tasks per processor.

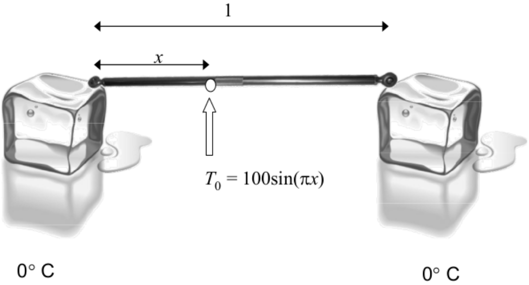
Outline

1. Objectives
2. Parallel computers
   - Parallel Computing — What is it?
3. Discussion
   - Supercomputers
4. Classification of Parallel Computers
   - Control Structure Based Classification
5. A parallel programming model
6. Foster's Design Methodology
7. Parallel Algorithm Examples

WITS
UNIVERSITY

Example 1: Boundary Value Problem

- Introduction



Example 1: Boundary Value Problem

- Introduction

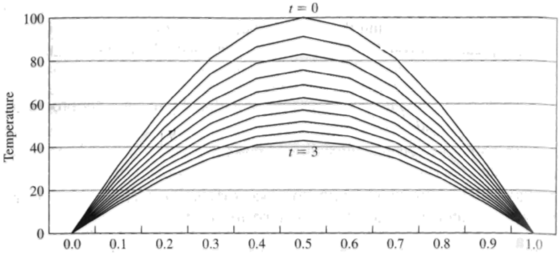


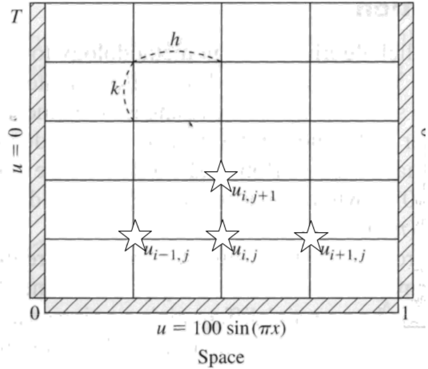
Figure 3.9 The rod cools as time progresses. The finite difference method finds the temperature at a fixed number of points in the rod at certain time intervals. Decreasing the size of the steps in space and time can lead to more accurate solutions.

Example 1: Boundary Value Problem

- Introduction: In the finite difference method, the algorithm steps forward in time, using values from time j to compute the value for time $j+1$ using the formula

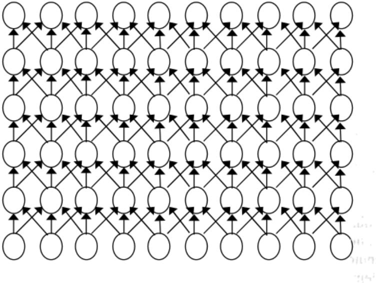
$$u_{i,j+1} = ru_{i-1,j} + (1-2r)u_{i,j} + ru_{i+1,j}$$

where $r = k/h^2$.



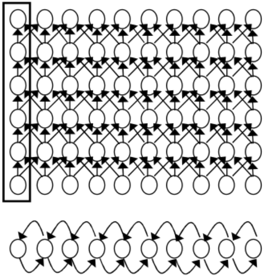
Example 1: Boundary Value Problem

- Partitioning
- Communication



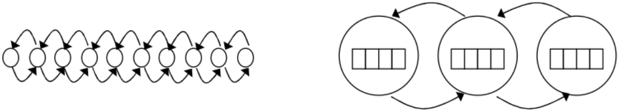
Example 1: Boundary Value Problem

- Agglomeration



Example 1: Boundary Value Problem

- Mapping



Example 1: Boundary Value Problem

- Analysis
  - Sequential:

$$m(n-1)x \tag{1}$$

    m, number of time steps; n, number of points; x, time for computing at a single point.
  - Parallel:

$$m(\lceil (n-1)/p \rceil x + 2\lambda) \tag{2}$$

    p, number of processors; λ, communication time for a single processor to send (or receive) a message.

Example 2: Finding the maximum

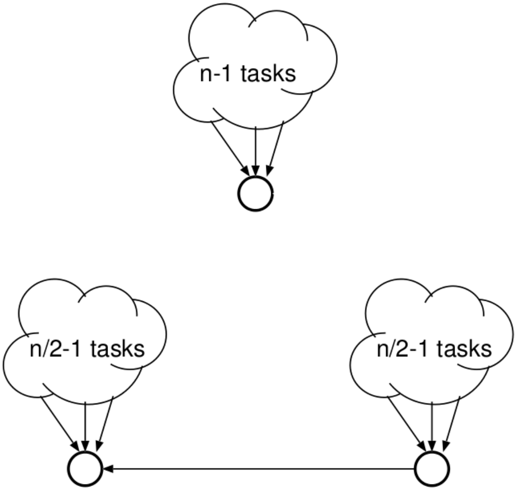
- Problem: Find the maximum value of a set of *n* values.
- This is a particular case of a reduction:

$$a_0 \oplus a_1 \oplus \cdots \oplus a_{n-1}$$

  where $\oplus$ can be any associative binary operator.
- A sequential reduction always takes $\Theta(n)$ time.

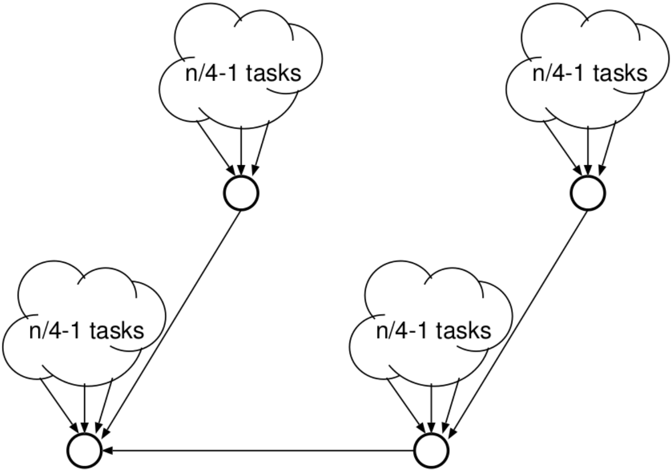
Example 2: Finding the maximum

- partitioning



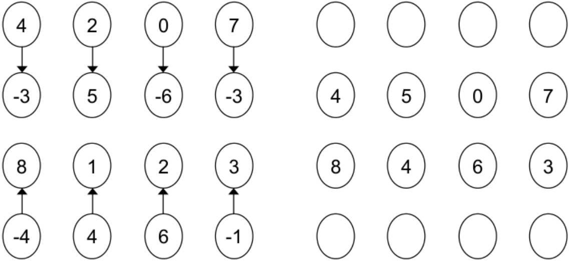
Example 2: Finding the maximum

- partitioning



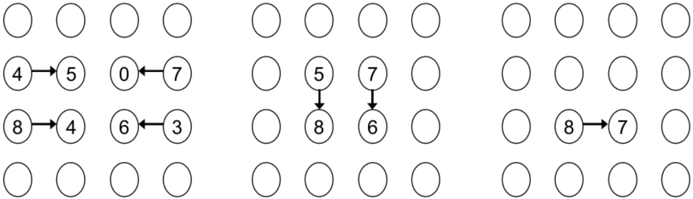
Example 2: Finding the maximum

- partitioning
- Communication



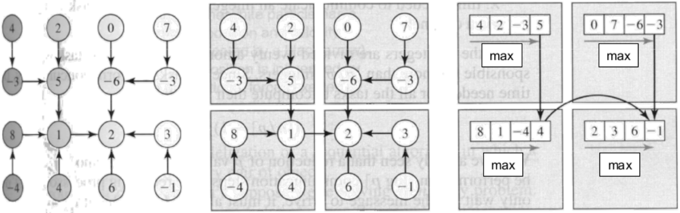
Example 2: Finding the maximum

- partitioning
- Communication



Example 2: Finding the maximum

- partitioning
- Communication
- Agglomeration
- Mapping



(a) (b) (c)

Example 2: Finding the maximum

- Analysis

Overall parallel execution time:

$$(\lceil n/p \rceil - 1)x + \lceil \log p \rceil (\lambda + x). \tag{3}$$

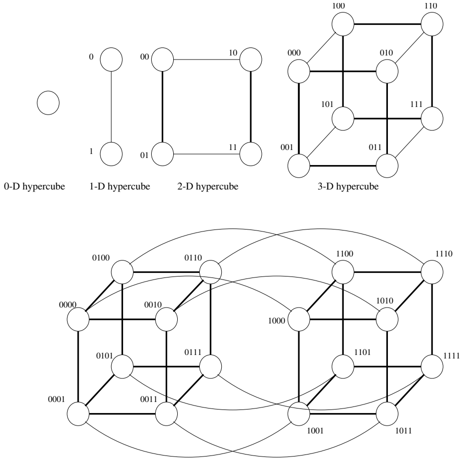
- x: time to perform the binary operation,
- λ: time to communicate an integer from one task to another.

The n-body problem

- In a Newtonian *n*-body simulation, gravitational forces have infinite range (*n* is the number of particles)
- We consider parallelizing a naive sequential solution where a computation is performed on every pair of objects.
- During each iteration, compute the new position and velocity vector of each particle, given the positions of all the other particles.
- In order to compute the new position of a particle, we perform a gather operation — Takes a dataset distributed among a group of tasks and collects the items on a single task.
- An all-gather operation is similar to gather, except at the end of the communication every task has a copy of the entire data set.
- If we want to update the location of every particle, an all-gather communication is needed.
- How to perform this operation?

The n-body problem

- Put a channel between every pair of particles? Not a good idea.
- Use a hypercube based communication to achieve all-gather communication.



0-D hypercube 1-D hypercube 2-D hypercube 3-D hypercube

4-D hypercube

The n-body problem

- Associate each particle with a task
- Agglomerate each n/p particles into a bigger task (assume p is an exact power of 2)
- After the agglomeration, all-gather communication requires $\log p$ communication steps.
- In the first step, the messages have length n/p, in the second step the messages have length $2n/p$, and $4n/p$ in the 3rd step, etc.
- Derive an expression for the expected execution time for this algorithm. Note that at each step the length of message is growing. Hence, we should consider the increased time for transmitting this message as well.

The n-body problem

- Communication time: assume λ is the latency, β the bandwidth. Sending a message of length n requires time $\lambda + n/\beta$.
- The communication time for each iteration is

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

- Suppose the time needed for gravitational force computation of one particle is x, then each task takes $x(n/p)$ to complete the computation for its share of particles.
- The expected time parallel execution time per iteration is

$$\lambda \log p + n(p-1)/(\beta p) + x(n/p) \tag{5}$$

References

- Parallel Programming in C with MPI and OpenMP, by Michael J. Quinn, Chapter 3.
- Designing and Building Parallel Programs, by Ian Foster, Chapter 2. Available online `http://www.mcs.anl.gov/~itf/dbpp/`.