ProgAlg - Résumé

Sylvain Julmy

12 juin 2016

Chapitre 1

Introduction

Concurrent computing : A form of computing in which programs are designed as a collections of interaction computational processes that may be executed in parallel.

Parallel computing : A form of computing in which many calculations are carried out simultaneously, operating on the principle that large problems can be divided into smaller ones. The most popular form is the multi-core processors.

Distributed computing : Any computing that involves multiple computers remote from each other that each have a role in a computation problem or information processing. Motivation : High performance (HPC) to avoid processing issue (compute faster) and memory issue (compute larger problems).

There is different forms of parallel computing :
- Bit-level
- Instruction level
- Data parallelism
- Task parallelism

There is 3 ways to ger performance :
- More powerful processing unit (CPU)
- Better algorithms
- Parallelize / distribute

N computers performing the task T would compute the task faster than with 1 computer but never N times faster.

Flops : number of floating point operation over time. $1Flops = 1$ floating point operation in 1 second.

MIPS : million of instructions per second. $1MIPS = 1$ million of instructions in one second

Peak performance : the maximum computing performance of a computer.

Sustained performance : the observed performance on a benchmark program (usually LINPACK).

Due to the rapid increasing of the numbers of connected processors it rapidly appeared that purely "shared memory" supercomputer is not an option (memory bottleneck).

- Processing unit : Smallest sequential processing element, a core.
- SMP : Single Memory Multi Processor, shared memory multi processing unit like a multicore.
- Node : Component which are connected through the network in a distributed memory multiprocessors computers.

There is two option for getting the power on distributed computers :

- Many low power nodes : Massively Parallel Processor (MPP) → "army of ants".
- Few high power nodes : connecting SMP nodes → "herd of elephants".

Chapitre 2

Architecture

2.1 Parallel architecture

Two main class :

- Shared memory : several processors sharing through a bus or a dynamic network the same set of memory bank.
- Distributed memory : several processors, having each its own local memory, communicate through a static or dynamic network.

2.2 Shared-address-space plaforms

- UMA : Uniform Memory Access → shared-address-space computer with local caches and global memories. All memory access times (except cache) are identical.
- NUMA : Non Uniform Memory Access → shared-address-space computer with local memory only. Local memory access times are shorter.

2.3 Implicit and Explicit parallelism

Implicit parallelism :

- Higher level of device integration have made available a large number of transistors.
- Current processors use these resources in multiple functional units and execute multiple instructions in the same cycle.
- The precise manner in which these instructions are selected and executed provides impressive diversity in architectures.

Explicit parallelism :

- An explicit parallel program must specify concurrency and interaction between concurrent subtasks.

SISD MISD
SIMD MIMD

2.4 Parallel vs. Distributed computation

Parallel processing refers to multiple CPUs within the same shared-memory machine performing computation.

Distributed computation involves multiple computers with their own memory communicating over a network.

2.5 Interconnection network

static : network consist of point-to-point communication links among processing nodes.

dynamic : networks are build using switches and communication links.

2.6 Structural approach

Based on the four main components of a computer :
- The control unit
- The computing unit
- The program memory
- The data memory

2.6.1 Sequential computer

There is only one of each component : One program using one set of data. At each time step only one instruction of the program is executed by the unique processing unit.

2.6.2 Vectorial computer

There is one control unit, one program memory, multiple data memories and multiple computing units : One program executes several times the sames instruction on several data (vector) in a synchronous way (one control unit).

2.6.3 Parallel computer

One data memory, several program memories, several computing units, several control units and several different programs execute asynchronousy on the same data set. All programs share the same memory.

2.6.4 Distributed computer

All component are duplicated, a network is added to communicate between the different processor. Several programs execute asynchronousy on different data set : Each program has its own memory and the communications between the programs are done through the network.

2.7 Network topology

2.7.1 Bus

— Some of the simplest and earliest parallel machines used buses.
— All processors access a common bus for exchanging data.
— The distance between any two nodes is $O(1)$ in a bus.
— The bus provides a convenient broadcasr media.
— The bandwidth of the shared bus is a major bottleneck.
— Typical bus based machines are limited to dozens of nodes.

2.7.2 Crossbars

It's like a matrix of connexion between the node, it use a $p \times m$ grid of switches to connect p inputs to m outputs in a non-blocking manner. The cost of a crossbar of p processors grows as $O(p^2)$, so it's difficult to scale for large values of p.

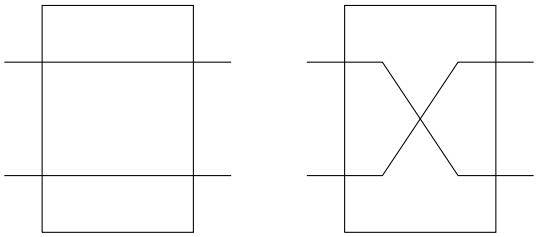
2.7.3 Multistage Networks

— Buses have excellent cost scalability, but poor performance scalability.
— Crossbars gave excellent performance scalability but poor cost scalability.
— Multistage interconnects strike a compromise between these extremes.
— It consists of $\log p$ stages, where p is hte number of inputs/outputs.
— At each stage, input i is connected to output j if :

$$j = \begin{cases} 2i & 0 \leq i \leq \frac{p}{2} - 1 \\ 2i + 1 - p & \frac{p}{2} \leq i \leq p - 1 \end{cases}$$

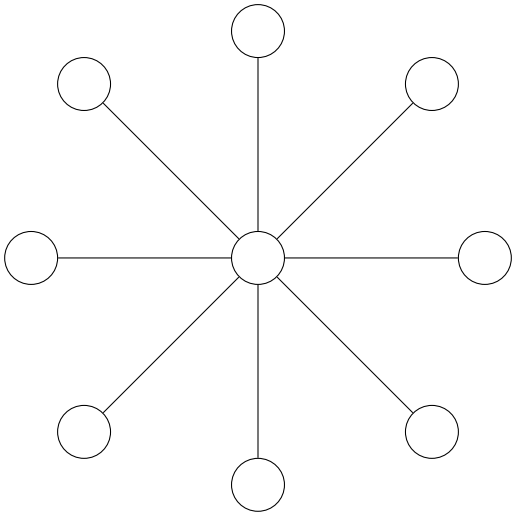
2.7.4 Multistage Omega Network

The perfect shuffle patterns are connected using 2×2 switches, there is two modes : cross-over or pass-through.



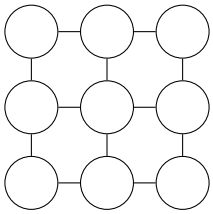
2.7.5 Stars

Every node is connected only to a common node at the center, the distance between any pair of nodes is $O(1)$. The bottleneck is the central node.



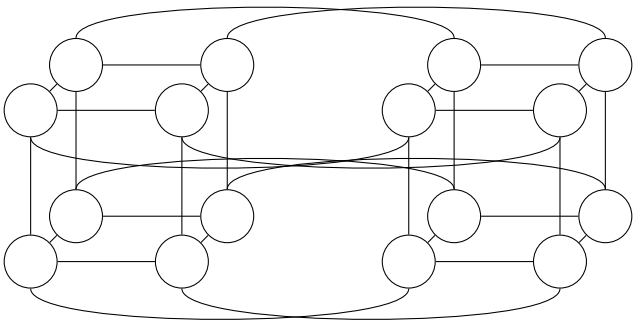
2.7.6 Linear arrays, Meshes and Torus

— In a linear array, each node has two neighbors, one to its left and one to its right.
— If the nodes at either end are connected, we refer to it as $1 - D$ torus or a ring.
— A generalization to $2d$ has nodes with 4 neighbors : north, south, east, west.
— A further generalization to d dimensions has nodes with $2d$ neighbors.



2.7.7 Hypercubes

— A special case of a d-dimensional mesh is a hypercube.
— $d = \log p$, where p is the total number of nodes.
— the distance between any two nodes is a most $\log p$.
— each node has $\log p$ neighbors.
— The distance between two nodes is given by the number of bit positions at which the two nodes differ.



2.7.8 K-ring

Intuitively, the K-ring topology is a graph built using K rings where each ring goes over all the nodes in a diffenret order. The value K is called the dimension. More formally, the definition of the K-ring topology is the following :

- The size N is an integer > 0.
- The dimension K is an integer > 0.
- K different positive integers $(a_1, ..., a_K)$, prime with N and smaller than $\frac{N+1}{2}$.
- The nodes are numbered from 0 to $N-1$.
- Each node i is connected to the nodes $i + aj \mod N, j \in \{1, ..., K\}$

Conventions :

- With such a construction each a_j defins a ring on the N nodes, this ring will be identified as the $ring_j$ and a_j will be called the $step_j$.
- By renumbering the nodes it is always possible, for any K-rings to obtain a $ring_j$ with a $step_j$ equal to 1. By convention, we decide that this ring is numbered by 1. Consequently we will only consider K-rings with $a_1 = 1$.

2.8 Evaluating interconnection netwroks

- Diameter : distance between the farthest two nodes in the network.
- Bisection width : minimum number of wires you must cut to divide the network into two equal parts.
- Cost
  - number of links or switches is a meaningful measures of the cost.
  - a number of others factors, such as the ability to layout the network, the length of wires,...
- Scalability
  - the ability to evolve.
  - the ability to be realized for any number of nodes.

Network	Diameter	Bisection	Degree	Cost	Nodes	Links
Completely-connected	1	$\frac{p^2}{4}$	$p-1$	$\frac{p(p-1)}{2}$	p	$\frac{p(p-1)}{2}$
Star	2	1	1	$p-1$	p	$p-1$
Binary Tree	$2\log(\frac{p+1}{2})$	1	1	$p-1$	p	$p-1$
2-D Mesh	$p-1$	1	1	$p-1$	$p = ab$	$2ab-a-b$
2-D Torus	$2\sqrt{p}-1$	$\sqrt{p}$	2	$2(p-\sqrt{p})$	$p = ab$	$2p$
Hypercube	$\log p$	$\frac{p}{2}$	$\log p$	$\frac{(p\log p)}{2}$	$p = 2^n$	$\frac{p}{2}\log p$
K-D Torus	$d\lfloor\frac{k}{2}\rfloor$	$2k^{d-1}$	$2d$	dp		
K-Ring	$\approx \frac{N^2\cdot\sqrt{N}2N}{4}\sqrt{p}N$		$2k$		p	kp
Crossbar	1	p	1	p^2		
Omega Network	$\log p$	$\frac{p}{2}$	2	$\frac{p}{2}$		
Dynamic Tree	$2\log p$	1	2	$p-1$		

2.9 Communication cost

Overhead in parallel programs comes from :
- — idling
- — contention
- — communication

Communication cost depends on :
- — communication model
- — network topology
- — data handling and routing
- — associated software protocols
- — ...

The total time to transfer a message over a network comprises of the following :
- — Startup time t_s : time spent at sending and receiving nodes (executing the routing algorithm, programming routers, etc...).
- — Per-hop time t_h : function of number of hops and includes factors such as switch latencies, network delays, etc...
- — Per-word time t_w : time which includes all overhaeds that are determined by the length of the message. This includes bandwith of links, error checking and correction, etc...

The cost of communicating a message between two nodes / hops away using cut-through routing is given by $t_{comm} = t_s + lt_h + t_w m$. By the ways, lt_h is smaller than t_s and t_w so we generally have $t_{comm} = t_s + t_w m$.

For Shared Address Space Machines, the simplified cost model is still valid, but a number of other factors make accurate cost modeling more difficult :
- — memory layout determined by the system
- — finite cache size can result in cache thrashing
- — overheads associated with invalidate and update operations are difficult to quantify
- — spatial locality is difficult to model
- — prefetching can play a role in reducing the overhead associated with data access
- — false sharing and contention are difficult to model

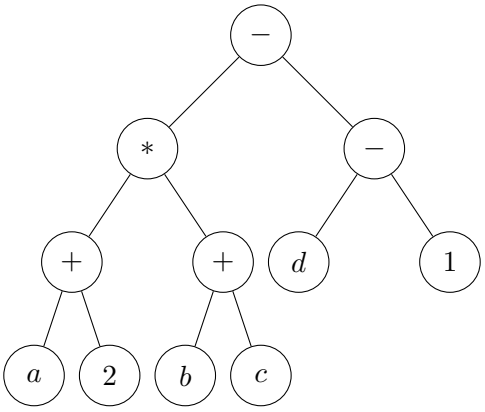
Chapitre 3

Models

There is three sources of parallelism :

- Control parallelism : do several things at once.
- Data parallelism : doing the same thing on multiple data.
- Flow parallelism : chain work.

3.1 Control parallelism



The computation is seen as a evaluation tree where each sub-tree can be computed separly, but there is dependencies between computation. The charge balancing is in charge of programmers.

3.2 Data parallelism

The data parallelism is based on the observation that we often have to repeat some action on similar data and a lot of application use large amount of data, like array or matrix, on which they compute some data.

The idea is based on the divide-and-conquer paradigm in which we could sort the 4 part of an array and merge them at the end.

3.3 Flow parallelism

The idea comes from the fact that some applications may run along a line work. The typical use of parallel flow is when you have a function $f(x)$ which could be decomposed into several ones : $f(x) = f_1(f_2(f_3(...(f_p(x)...))))$. That is call a pipeline.

3.4 Speedup and efficiency

3.4.1 Speedup

The speedup is given by $S(p) = \frac{T_1}{T_p}$, where T_1 is the sequential time to solve p and T_p is the parallel time to solve p. Speedup could only be consider for solving the same problem and we must clearly specify what we exactly compare.

The ideal reference value is a linear function $S(p) = p$. Usualy $S(p) < p$) because of overhaed like communication, idling, contention,...

3.4.2 Efficiency

The efficiency $E(p)$ is the ration between the speedup and the number of processing element p : $E(p) = \frac{S(p)}{p}$. If $E(p) = 1$, this is optimal because we have $S(p) = p$.

3.4.3 Amdahl law

This law consider that the execution time T_1 of a sequential program can be split into two part : T_s and $T_{//}$. Where T_s is the sequential part of the program and $T_{//}$ the parallel ones. So we have $T_1 = T_s + T_{//}$. Using p processing element result of having $T_p \geq T_s + \frac{T_{//}}{p}$.

Now we take the speedup and we having the following equation :

$$S(p) \leq \frac{T_s + T_{//}}{T_s + \frac{T_{//}}{p}}$$

Then we compute the limit of p to ∞ :

$$\lim_{p \to \infty} S(p) = S(\infty) \geq \frac{T_s + T_{//}}{T_s} = \frac{T_1}{T_s} = {\frac{T_s}{T_1}}^{-1}$$

The value $\frac{T_s}{T_1}$ defines the proportion of the program which is always sequential.

Consequences : The Amdahl law expresses the fact that the upper bound of the speedup is inversely proportional to the percentage of the code which is sequential.

3.4.4 Gustafon law

This law explain that more the problem is big, better it parallelizes. Let's denote N the size of the problem, we have $T_s(N) = f(N) \cdot T_1(N)$. We could found the acceleration $S_p(N) = \frac{T_1(N)}{T_p(N)} \leq \frac{p}{p \cdot f(N) + 1 - f(N)}$.

3.5 Performance

There is two type of peformance :

- Time performance
  - Wall-clock time : from the first processor start to the last processor end. Problems : number of processor use on different machine.
  - Raw FLOP count. Problems : what good are FLOP counts when they don't solve a problem.
- Memory performace

The main problem with performance it's that is dependent of the computer we are using, this is why we prefer the notion of complexity : time and memory complexity.

3.5.1 Parallel complexity

The parallel time complexity of a parallel algorithm depends on :

- the input size N
- the number of processors p
- the communication parameters of the machine

An algorithm must, therefire, be analyzed in the context of the underlying platform.

3.6 PRAM

Parallel Random Access machine : consist of p precessors and a global memory of unbounded size that is uniformly accessible to all processors (UMA). Processors share a common clock but may execute different instructions in each cycle.

For memory access we use the following :

- Exclusive-read, Exclusive-write (EREW)
- Concurrent-read, Exclusive-write (CREW)
- Exclusive-read, Concurrent-write (ERCW)
- Concurrent-read, Concurrent-write (CRCW)

Concurrent-write :

- Common : write only if all value are identical.
- Arbitrary : write the data from a randomly selected processor.
- Priority : follow a predetermined priority order.
- Sum : write the sum of all data items.

3.7 BSP

Bulk Synchronous Parallel abstract computer : composed of a set of pairs processors and local memory, a communication network a efficient synchronization mechanism.

BSP execution model is a sequence of supersteps, during a superstep each processor can :
- Do any local computation
- Send request to other processors
- Send data to other processors

At the end of each supersteps there is a synchronization phase between all processors where all communication are effectively realized.

3.8 Performance metrics

- Execution time : serial and parallel
- Speedup
- Efficiency
- Overhead $T_o = pT_p - T_s$

The time $T_{//}(A_N)$ is the parallel execution time of the algorithm A with a data of size N. It's define as the numbers of steps used by the PRAM machine to execute the algorithm.

The surface $H_{//}(A_N)$ used by the algorithm A with a data of size N is defined as the maximal number of processors necessary to the PRAM machine to execute the algorithm.

The work $W(A_N)$ of the algorithm A with a data size of N is defined by $W(A_N) = T_{//}(A_N) \cdot H_{//}(A_N)$.

3.9 Efficient and optimal parallel algorithms

A parallel algorithm A is said efficient iff :
- $O(T_{//}(A_N)) = \log^k(N)$
- $O(W(A_N)) = T_1 O(\log^k(N))$

In other words, the time complexity of the algorithm is polylogarithmic, this means that the time complexity is better that any sequential algorithm.

A parallel algorithm A is said optimal iff :
- $O(W(A_N)) = O(T_1(A_N))$
- $O(T_{//}(A_N)) = \log^k(N)$

A parallel algorithm is optimal if is time complexity is polylogarithmic and if it performs the same work than the sequential algorithm.

To run the algorithm we need $\frac{N}{2}$ processing elements, time complexity is $\log(N)$ and work complexity is $N\log(N)$. But the time complexity of the sequential algorithm (which is equal to the work complexity because $H = p = 1$) is $O(T_1) = N$. So the algorithm is efficient but not optimal.

Can we make this algorithm optimal ? : It is not optimal because the work complexity is bigger than the work complexity of the sequential algorithm. The problem comes from the surface which uses $\frac{N}{2}$ processors : $O(H_{//}(A_N)) = N$. So the solution is to optimaly use the good number of processor on local part.

Chapitre 4

Communication

Two types of communications :

- Point-to-Point : between two nodes (processors or processes)
  - Direct neighbors → without routing
  - Not directly connected → routing by intermediate nodes
- Global (group) communications : 2 or more nodes
  - broadcast, multicast, reduction, scatter, gather ...

4.1 Cost Model

4.1.1 Parameters

- t_L : Latency $[sec]$
- W : Bandwidth $[byte/sec]$, amount of bytes sent during one amount of time
- t_S : routing time on intermediate nodes $[sec]$
- m : message size $[byte]$

4.1.2 Formulas

Time $T[sec]$ needed to transfer a message with H intermediate nodes :

$T = t_L + m * t_W + H * t_S$

if $H = 0$ (pint-to-point) : $T = t_L + m * t_W$

t_S can often be neglected : $T = t_L + m * t_W$

4.2 Routing

Two ways :

- Store-and-Forward : locally stored on a node, then transmitted further
- Cut-Through : not entirely stored
  - 2 types : Circuit Switched and Worm-hole

— Both based on header to set switches in advance

4.3 Global communications

	Agregate	Reduce	Personalised
One-to-all		One-to-all broadcast	Scatter
All-to-one	Gather	All-to-one reduction	
All-to-all	All-to-all broadcast	All-to-all reduction All-Reduce	Total Exchange

FIGURE 4.1 – Types of global communications

4.3.1 One-to-All and All-to-One

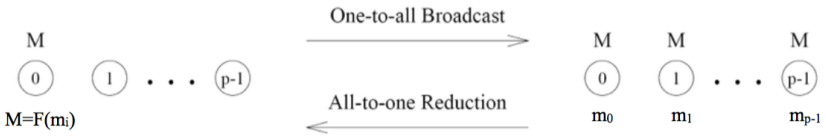


FIGURE 4.2 – One-to-All : ne processor has a piece of data (of size m) it needs to send to everyone

All-to-One :

— each processor has m units to broadcast
— results made available at target processor
— With or without reduction :
  — With : data items combined (reduce)
  — Without : m increases at each step (glued together, gather)

On Rings

— Naive : send $p-1$ messages from source to other $p-1$ nodes
  — source not is bottleneck
  — network under-utilized
  — require $p-1$ steps
— Recursive doubling : at each step the node owning the message sends the message to a selected node

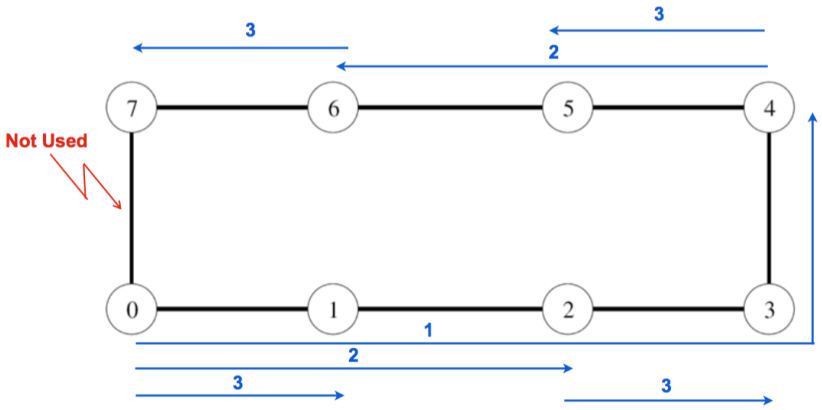


FIGURE 4.3 – One-to-All on ring. Number on arrow = step where transfer happens

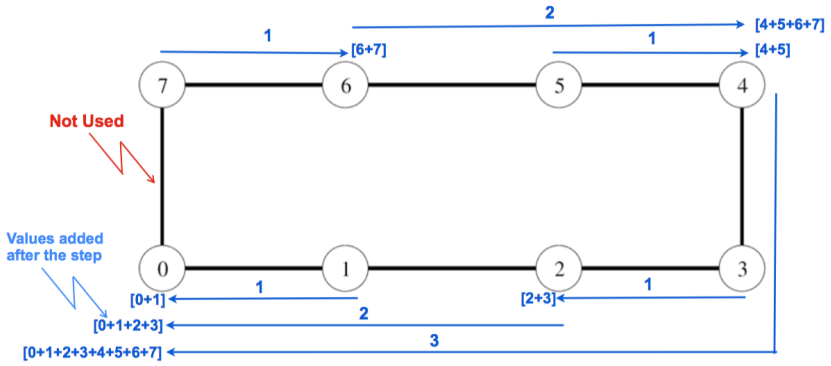


FIGURE 4.4 – All-to-One on Ring

Number of steps : $\log p = \log 2^N$

With reduction = Reduce (m constant)

- Transmission time for step i : $T_i = t_L + m * t_W + H_i * t_S$
- Total transmission time : $T = \log p * (t_L + m * t_W) + (p-1) * t_S)$

Without reduction = Gather (m increases at each step)

- Time for step i :
  - $T_i = t_L + m_i * t_W + H_i * t_s$
  - $m_i = 2^i * m$
- Total time : $T = \log p * t_L + m(p-1) * t_W + (p-1) * t_S$

On Mesh, Hypercube

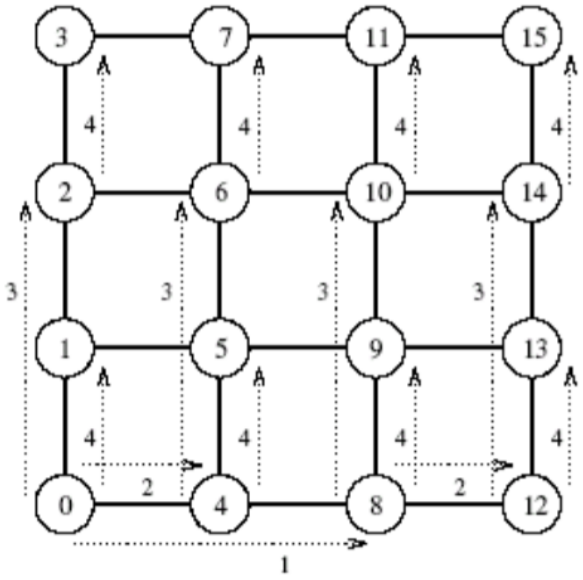


FIGURE 4.5 – One-to-All on mesh

Formulas :
- dD-mesh : $\log p * (t_L + m * t_W) + d * (\sqrt[d]{p} - 1) * t_S)$
- Hypercube $(d = \log p)$: $T = \log p * (t_L + m * t_W + t_S)$

4.3.2 All-to-All

2 types :

- All-to-All Broadcast : Each processor p_i sends to all others the same m_i-word message, different processors may broadcast different messages
- All-to-All Reduction : every node is the destination of an all-to-one reduction, each node starts with p messages

On a ring

- Half of the nodes first sends its message to one of its neighbors to initiate the all-to-all communication.
- In subsequent steps, each node forwards the [0,4,5,6,7] data he is receiving plus the data he is already owning to its neighbor
- The algorithm terminates in p steps

For reduction, inverted process. On receiving a message, a node must combine (for example add) it with its local copy of the value before forwarding the value message to the next neighbor.

Costs on ring :

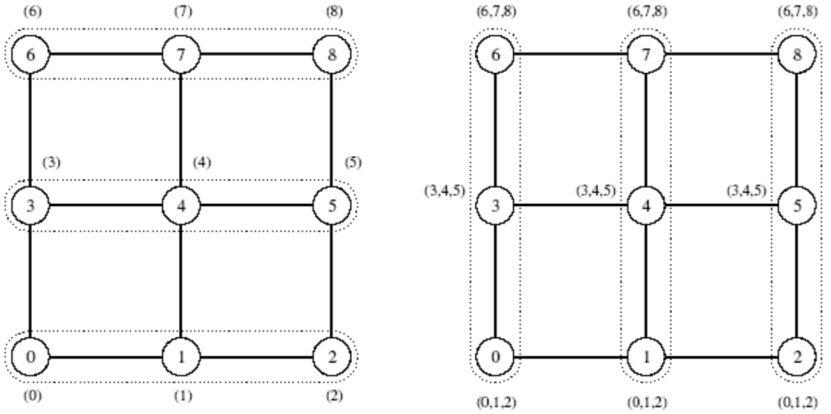
- With reduction : $T = p * (t_L + m * t_W)$
- Without reduction : $T = p * (t_L + t_W * m(2p - 1))$

Read and write at the same time :

— All of the nodes first sends its message to one of its neighbors to initiate the all-to-all communication.
— In subsequent steps, each node forwards the data he is receiving plus the data he is already owning to its neighbor the algorithm terminates in p-1 steps.
— With or without reduction at each step p messages of size m are sent and received.

Total time (with or without reduction) : $T = p * (t_L + m * t_W)$

On a mesh



(a) Initial data distribution (b) Data distribution after rowwise broadcast

FIGURE 4.6 – All-to-All broadcast on a mesh

Algorithm :

1. Phase 1
   — Each row of the mesh performs an all-to-all broadcast using the procedure for the ring
   — In this phase, all nodes collect $\sqrt{p}$ messages corresponding to the $\sqrt{p}$ nodes of their respective rows
2. Phase 2
   — Columnwise all-to-all broadcast

Torus is better for routing between the ends of columns/rows.

Cost for Torus :

— With reduction : $T = 2(\sqrt{p} * (t_L + m * t_W))$
— Without reduction :
   — Phase 1 : $\sqrt{p}(t_L + t_W * m(2\sqrt{p} - 1))$
   — Phase 2 : $\sqrt{p}(t_L + t_W * \sqrt{p} * m(2\sqrt{p} - 1))$
   — Total : $\sqrt{p} * (2t_L + m * (2p + \sqrt{p} - 1) * t_W)$

Cost for Torus with R/W in the same step :

— Without reduction :
   — First phase : $(\sqrt{p} - 1) * (t_L + m * t_W)$

— Second phase : $(\sqrt{p}-1)*(t_L+\sqrt{p}*m*t_W)$
— Total : $T=(\sqrt{p}-1)2t_L+m*t_W(p-1)$
— With reduction : $T=2(\sqrt{p}-1)*(t_L+m*t_W)$

4.3.3 Total Exchange

FIGURE 4.7 – Total exchange

On a ring

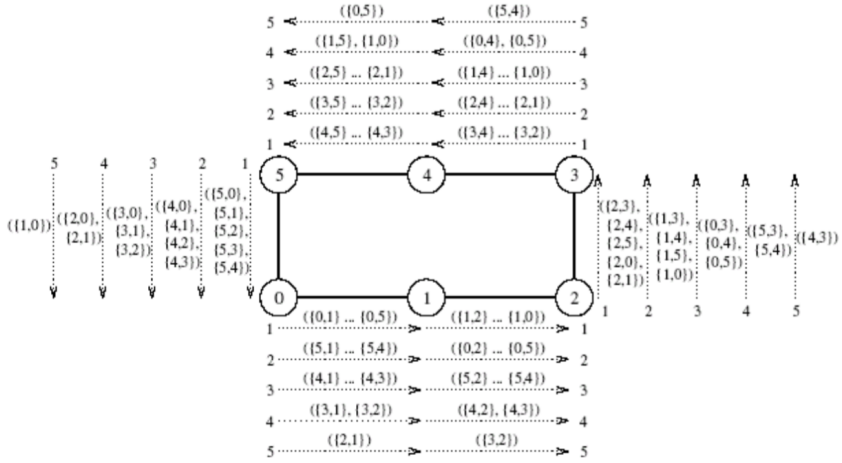


FIGURE 4.8 – Total exchange ring

Algorithm :

1. Each node sends all pieces of data as one consolidated message of size $m(p-1)$ to one of its neighbors
2. Each node extracts the information meant for it from the data received, and forwards the remaining pieces of size m each to the next node
3. The size of the consolidated message reduces by m at each step

Cost : $T=(p-1)*(t_L+t_W*m*p)$

On a mesh

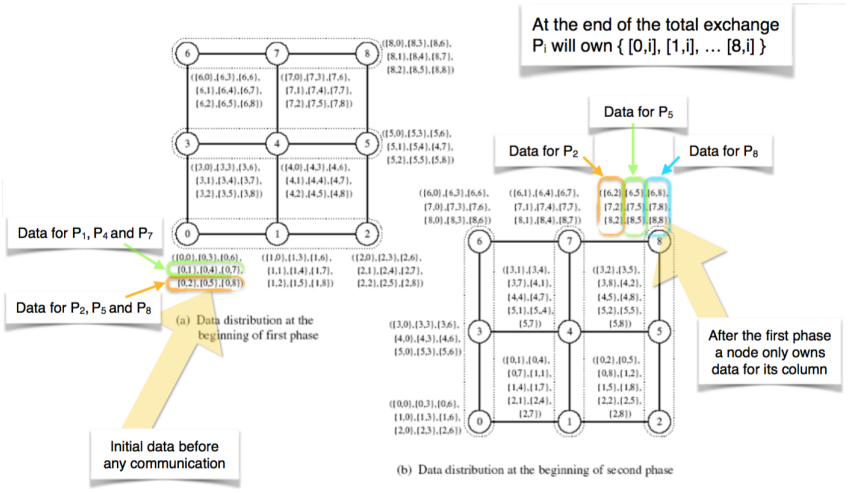


FIGURE 4.9 – Total exchange on a Mesh

Algorithm :

1. First step
   — All-to-All personalized communication is performed independently in each row
   — Each node of the row keeps all messages for which the destination is inside the column it belongs , it forwards the other ones
2. Second step
   — All-to-All personalized communication is performed independently in each column
   — Each node keeps the messages for which it is the destination, it forwards the other ones

Cost : $T = (2t_S + t_W mp)(\sqrt{p} - 1)$

Chapitre 5

Sorting algorithms

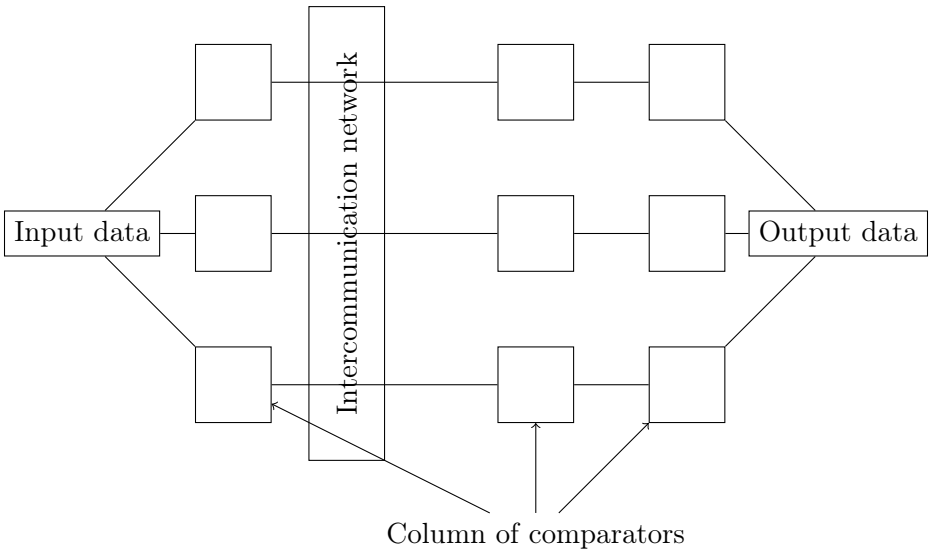
5.1 Constant time sorting algorithm

The constant time sorting algorithm is not efficient because is work complexity = $O(N^2) = O(N \log^{k+1}(N))$. This algorithm is not optimal either. Is surface complexity is $O(H_{//}(A_N)) = O(N^2)$.

5.2 Sorting network

A Sorting network is a network of comparators designed specifically for sorting. A comparator is a device with 2 inputs and 2 outputs $x_{out} = min(x_{in}, y_{in})$ and $y_{out} = max(x_{in}, y_{in})$ or the inverse (increasing comparator for first and decreasing comparator for the second).

Every sorting network is made up of a series of columns and each column contains a number of comparators connected in parallel.



A network with n inputs performing a max operation is denoted by $\ominus BM[n]$

5.3 Bitonic sort

First, we define what is a bitonic sequence. A bitonic sequence has two tones : increasing then decreasing, or vice versa and any cyclic rotation of a bitonic sequence is also bitonic.

$\langle 1, 2, 3, 7, 6, 0\rangle$ and $\langle 9, 2, 1, 0, 4, 8\rangle$ are bitonic sequence.

The bitonic sort is the following :
- Build a single bitonic sequence from the given unsorted sequence
- Rearrange the bitonic sequence into a sorted one

A bitonic sorting network sorts n elements in $O(\log^2 n)$ time.

To sort a bitonic sequence :
- Let $s = \langle a_0, a_1, ..., a_{n-1}\rangle$ a bitonic sequence.
- Consider the two subset of $s = \{s_1, s_2\}$
  - $s_1 = \langle min(a_0, a_{n/2}), min(a_1, a_{n/2+1}), ...\rangle$.
  - $s_2 = \langle max(a_0, a_{n/2}), max(a_1, a_{n/2+1}), ...\rangle$.
- s_1, s_2 are bitonic sequence too and each element of s_1 is less or equals to each element of s_2.
- Apply the procedure recursively too fully sort s.

The sorting network which use the bitonic algorithm contains $\log n$ columns and each columns contains $\frac{n}{2}$ comparators. In this case we perform a *min* operation.

Now we need to transform any sequence into a bitonic ones :
- A sequence of length 2 is always bitonic.
- A bitonic sequence of length 4 by sorting the first two elements using $\oplus BM[2]$ and next two element using $\ominus BM[2]$.
- This process can be reapeted to generate larger bitonic sequences.

The network of this algorithm contains $\frac{n}{2}$ with a $\oplus BM[2]$ on even place of the horizontal. This algorithm could also be mapped to meshes by performing compare-exchange operation between two wire only ig their labels differ in exactly one bit. The complexity for the mesh is $T_P = O(\log^2 n) + O(\sqrt{n})$,this is not the optimal cost.

When each processor hsa more than one value, the compare-exchange operation is replaced by the compare-split operation :

- assume each processor have $\frac{n}{p}$ elements.
- after the operation, the smaller $\frac{n}{p}$ are at processor P_i and the larger at P_j, where $i < j$.
- the complexity of a compare-split is $O(\frac{n}{p})$.
- the communication time is $t_s + t_w \frac{n}{p}$ (assuming they are neighbour).
- when n increase, t_s becomes insignificant.
- the global complexity is $O(\frac{n}{p})$.

1. each process sens its block of size $\frac{n}{p}$ to the other process
2. each process merges the received block with its own block
3. each process retains only the appropriate half of the merged block (process P_i retains the smaller elements and P_j the larger)

Quicksort : $O(n \log n)$.

Mergesort : $O(n \log n)$, but on a PRAM $O(n)$ with $O(T_{//}(A_N)) = O(N^2)$, neither efficient or optimal : communication overhead.