Lesson 1: Parallel Computers

G Stefanescu — University of Bucharest

Parallel & Concurrent Programming
Fall, 2014

The demand for computational speed

- Computationally difficult problems: those that can not be solved in a reasonable amount of time by today's computers.

- Examples: modeling large DNA structures, global weather forecasting, modeling motion of astronomical bodies, etc.

- Detailed example (weather forecasting):
  —Suppose global atmosphere is divided into 5×10^8 cells (each of size 1 mile × 1 mile × 1 mile; the hight is up to 10 miles) and each calculation requires 200 floating point operations; hence one step requires 10^{11} floating point operations.
  —To forecast the whether over 10 days using 10-minute intervals, a computer operating at 100 Mflops takes:
  $6(iterations\ for\ an\ hour) \times 24(hours\ per\ day) \times 10(days) \times 10^{11} : 10^8(Mflops) \approx 1.5 \times 10^6(sec) \approx 17 days$ (unacceptable)

Parallel processing

To be successful, *parallel processing* needs to fulfill 3 conditions:

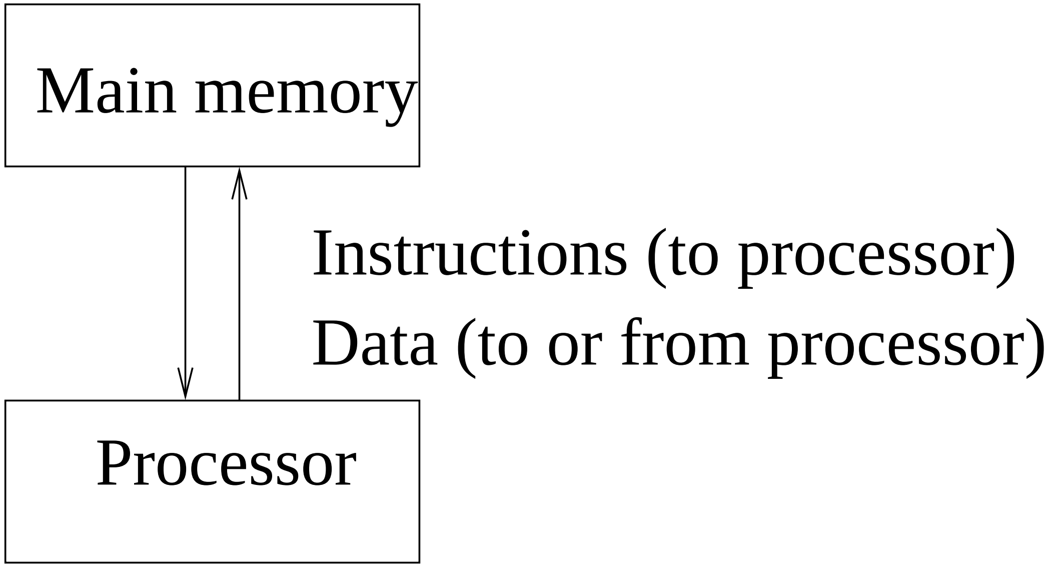
- *algorithm design*: the problem should be good enough to be decomposed in parts which may be solved in parallel
- *programming language*: the programming languages needs specific statements for implementing the parallel algorithm
- *parallel computer*: the computer [or network of computers] needs hardware capabilities for a real parallel running of the program

Our course focuses on the second issue; other SoC courses CS4231 and CS5221 cover in more detail the other aspects of parallel processing.

Parallel computers and programming

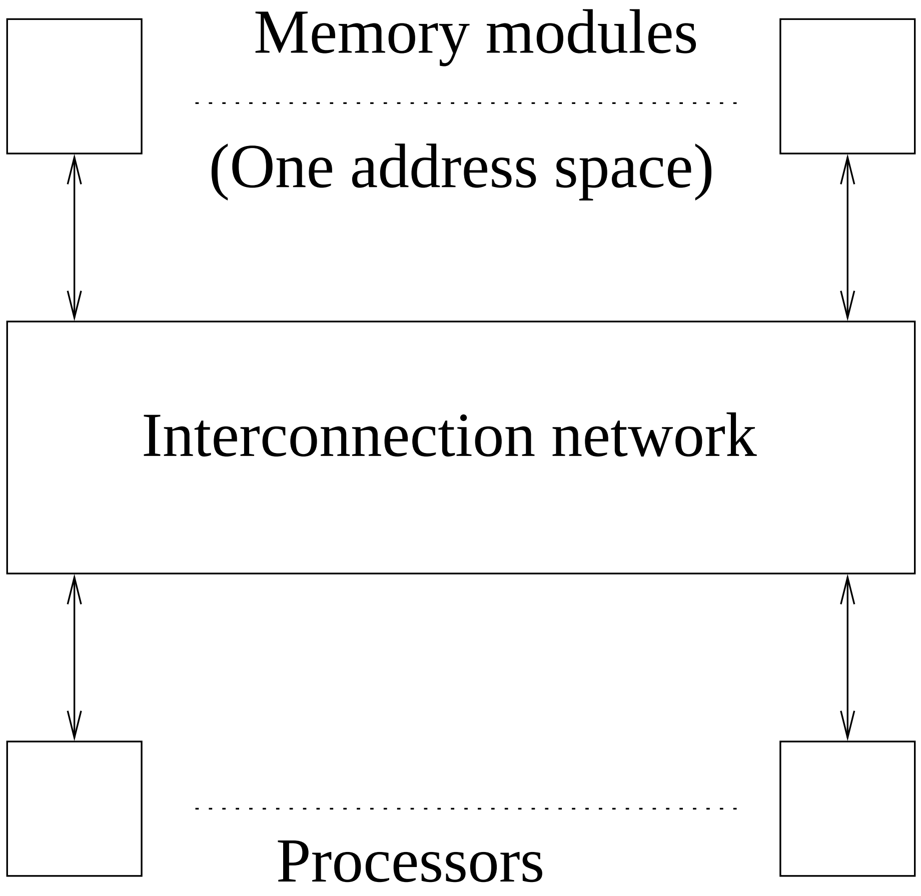
The key point is to use *multiple processors* operating together on a *single problem*. *[It is an old topics, see e.g., Gill, S 1958.]*

A conventional computer has a *processor* executing a program stored in a main *memory*.



(1) Shared memory multiprocessor system

In this model we have *multiple processors* and multiple *memory modules* such that *each processor can access any memory module* (hence we have a *shared-memory configuration*)



...and programming

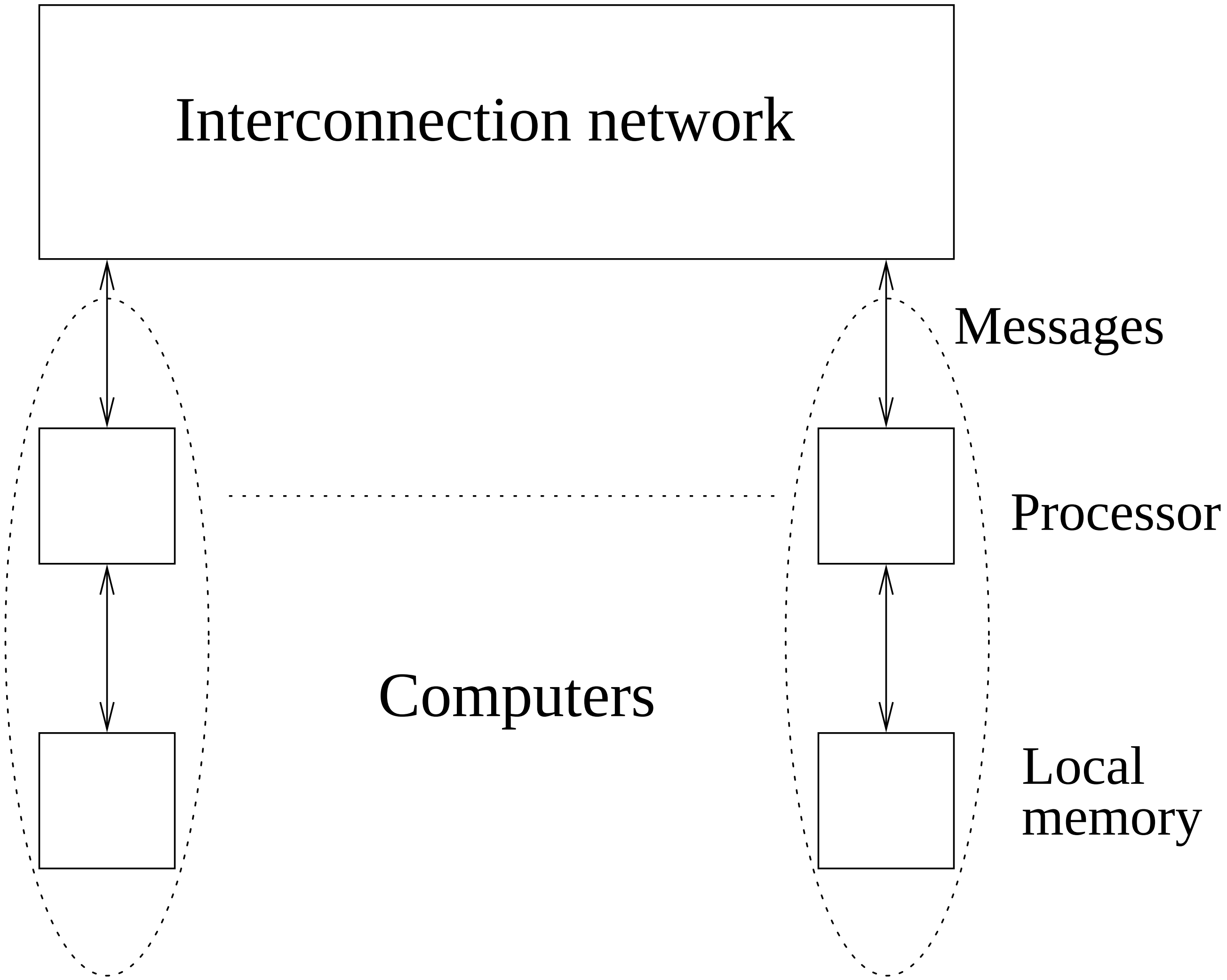
Programming in this case can be done in different ways:

- *Special (low level) parallel programming constructs*: The programs use constructs and statements that allow shared variables and parallel code sections to be declared. The *compiler* is responsible for producing the final executable code.

- *Threads*: One can use these high level constructs to describe the code for each processor and to access shared locations.

(2) Message-passing multicomputer (!?)

In this 2nd model we have complete computers connected through an *interconnection network*.



...and programming

- As before, one has to *divide the problem into parts* that are intended to be executed *simultaneously* to solve the problem. (Each part is solved by one of a set of *concurrent processes*.)
- Different from the previous case, the processes will communicated by *sending messages* only.

Notice: *In this case one has a sort of shared memory, too. But now this memory is* distributed*: Each processor has access to the full memory space using explicit message sending, only. (Sometimes, this access may be done in an automatic way hiding the fact the memory is actually distributed.)*

MIMD and SIMD classification

A useful classification of computers takes into account the *number of programs (instruction streams)* and the *number of input data* used for solving a problem.

- *SISD*: Single Instruction-stream Single Data-stream computers are just conventional computers (one program, one input datum);

- *MIMD*: Multiple Instruction-stream - Multiple Data-stream systems have multiple processors, separate programs and data for each processor. (*Both, shared memory and message passing multi-processors models just described are MIMD systems.*)

..MIMD and SIMD classification

- *SIMD*: Single Instruction-stream - Multiple Data-stream systems have multiple processors, multiple data again, but only one program which is broadcasted to each processor. (*Useful, as there are many important applications.*)
- *MISD*: Not so much used...

..MIMD sub-classification

Within the MIMD class (the main one we are concerned with), one may further identify 2 subclasses:

- *MPMD (Multiple Program Multiple Data)*: Each processor have its own program to execute (e.g., PVM programs).

- *SPMD (Single Program Multiple Data)*: In this case there is a single source program and each processor executes its personal copy of the program, independently. The source program can be constructed such that *parts of the program are executed by certain processors and not by others depending on the identity of the processor*. (This is mainly the case of MPI programs. Notice that SIMD and SPMD are different concepts, leading to different classes.)

Network criteria

The following data may be used to assess the quality of a network:

- *Cost* - given by the number of links is the network (and the difficulty of the network construction, but this is not so easy to be captured as a number)
- *Bandwidth* - number of bits that can be transmitted in unit time (given as bits/sec)
- *Network latency* - the actual time to transfer a message through the network
- *Communication latency* - total time to send a message, including software and interface delays

..Network criteria

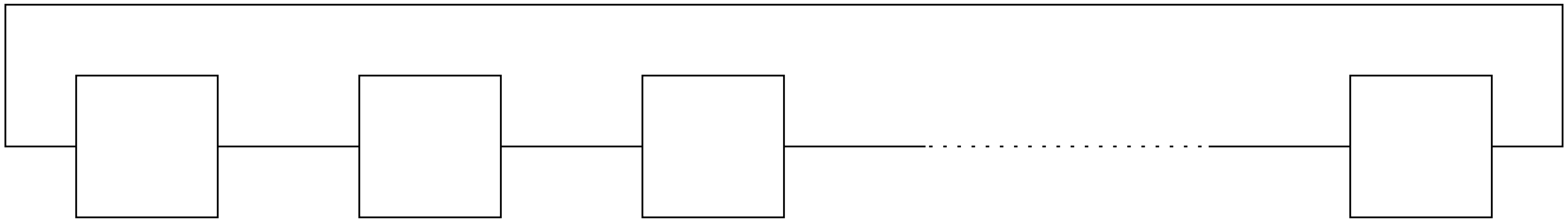
- *Message latency or startup time* - time to send a zero-length message (basically, the software and hardware overhead in sending messages)

- *Diameter* - the minimum number of links between two farthest nodes in the network (useful to determine the worst case delays)

- *Bisection width* - the number of links that must be cut to divide the network into two equal parts (useful to find lower bounds for sending messages in a parallel algorithm)

Examples of networks

Ring: A network of computers connected in a ring.



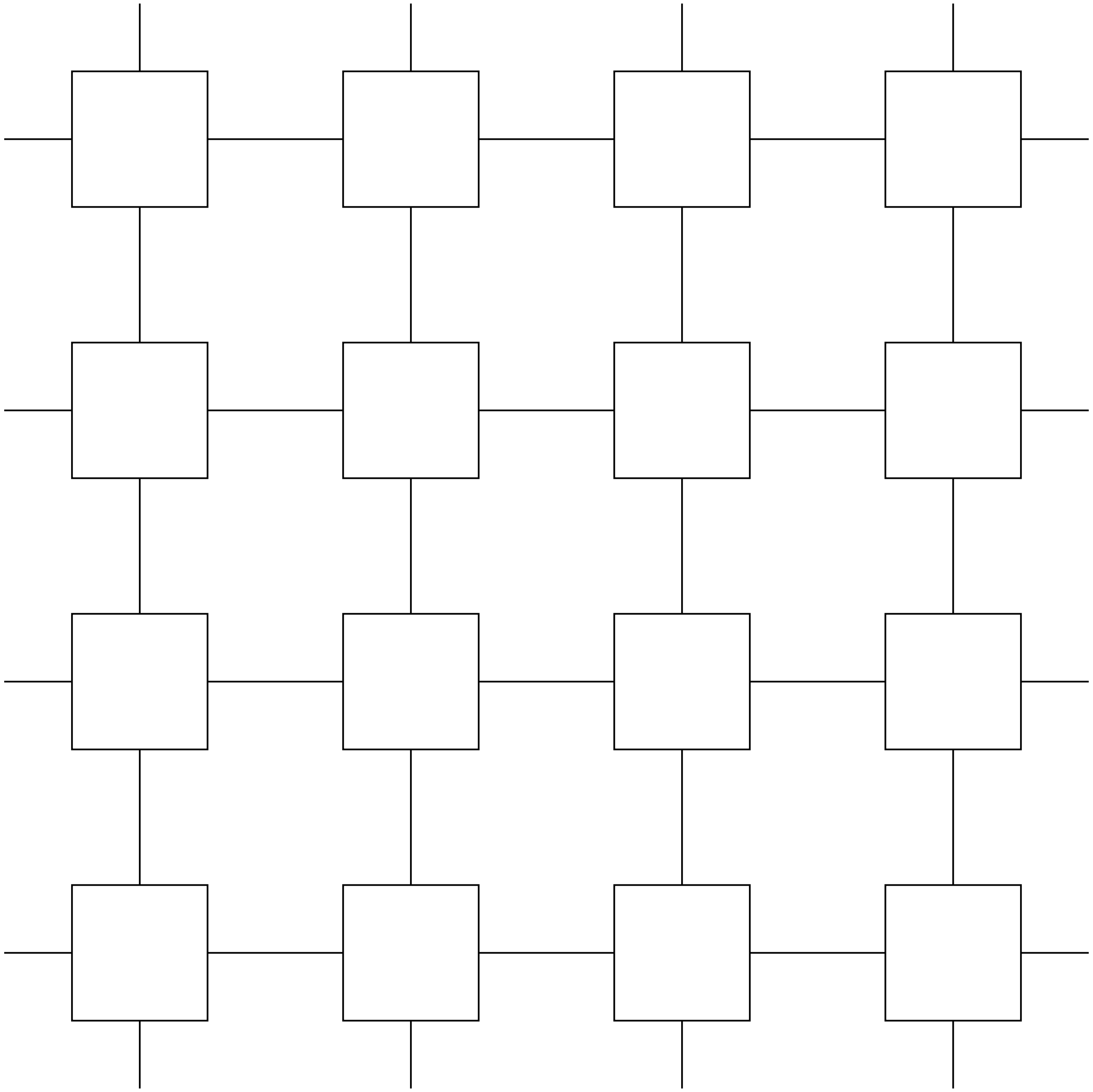
Example of measure: The diameter D is

- $n-1$ (n is the number of nodes) if the links are in one direction only;
- $\lfloor n/2 \rfloor$, if the links are bidirectional.

..Examples of networks

Mash: A two-dimensional array of computers.

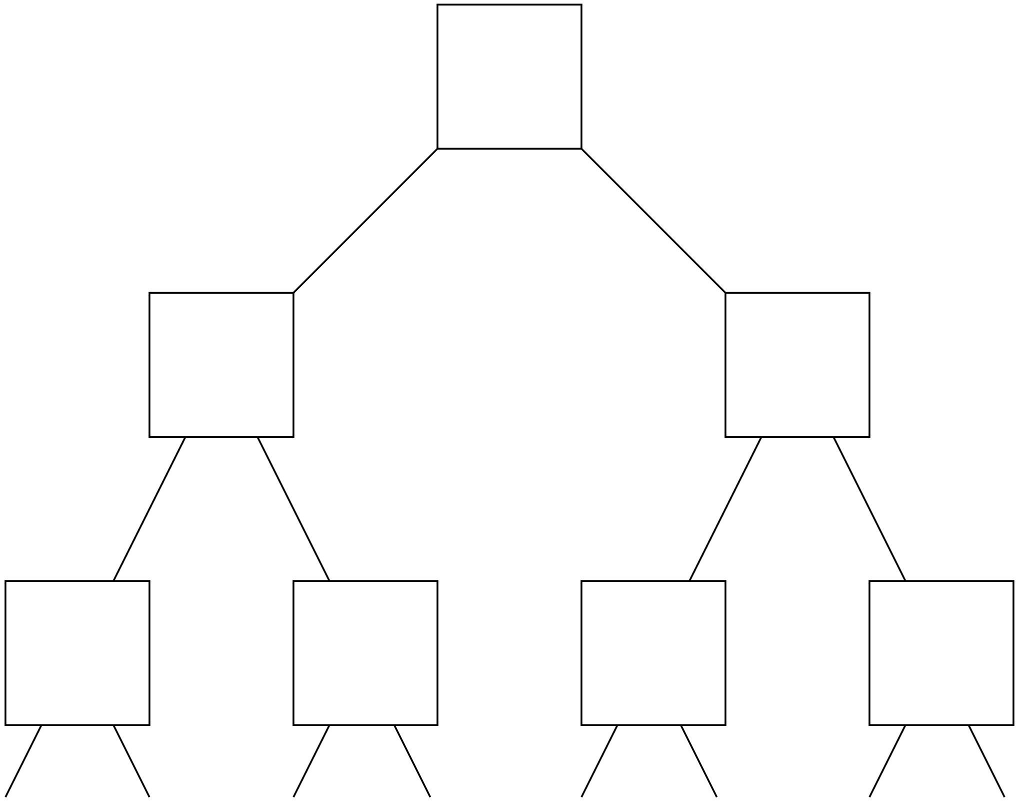
Example of measure: The diameter D is $2(\sqrt{n}-1)$ (the links are bidirectional)



..Examples of networks

Tree: A network of computers connected as a binary tree.

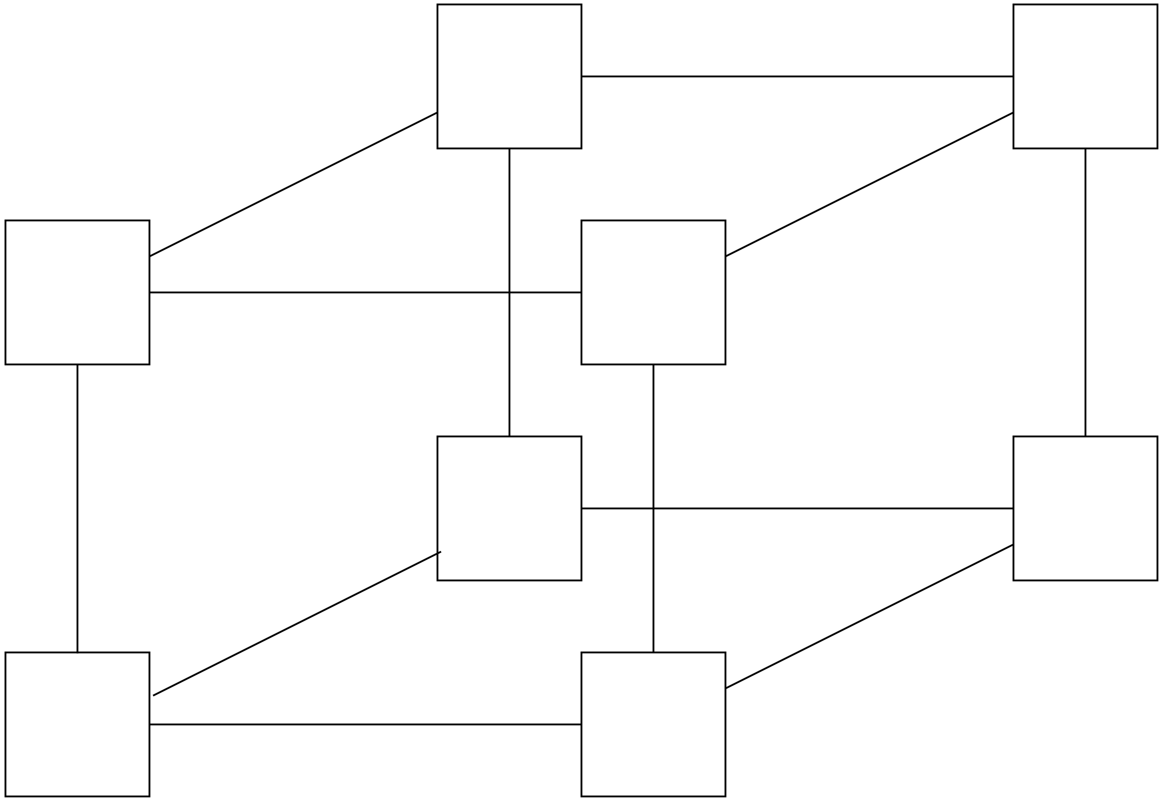
Example of measure: The diameter D is $2\,log_2(n+1)-2$ (the links are bidirectional)



..Examples of networks

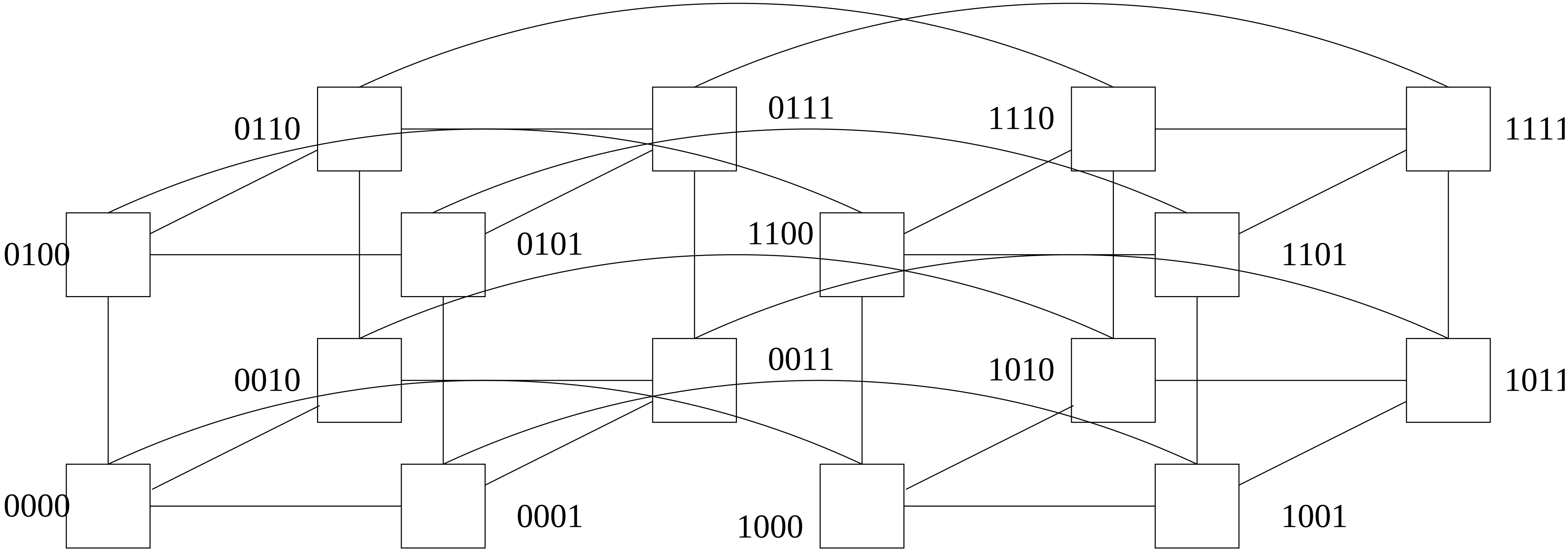
Hypercube: A network of computers connected as a three-dimensional hypercube.

Example of measure: The diameter D is $log_2 n$ (the links are bidirectional)



..Examples of networks

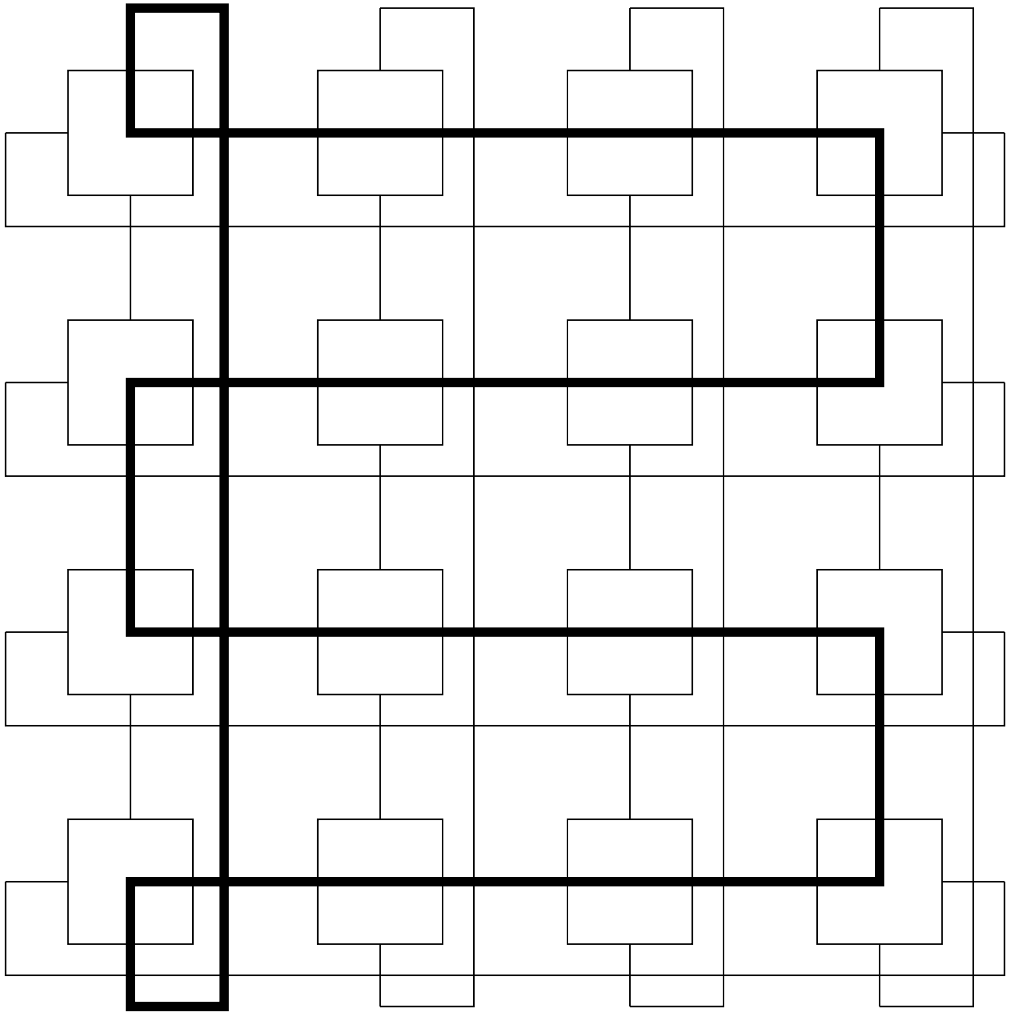
A four-dimensional hypercube



Embedding

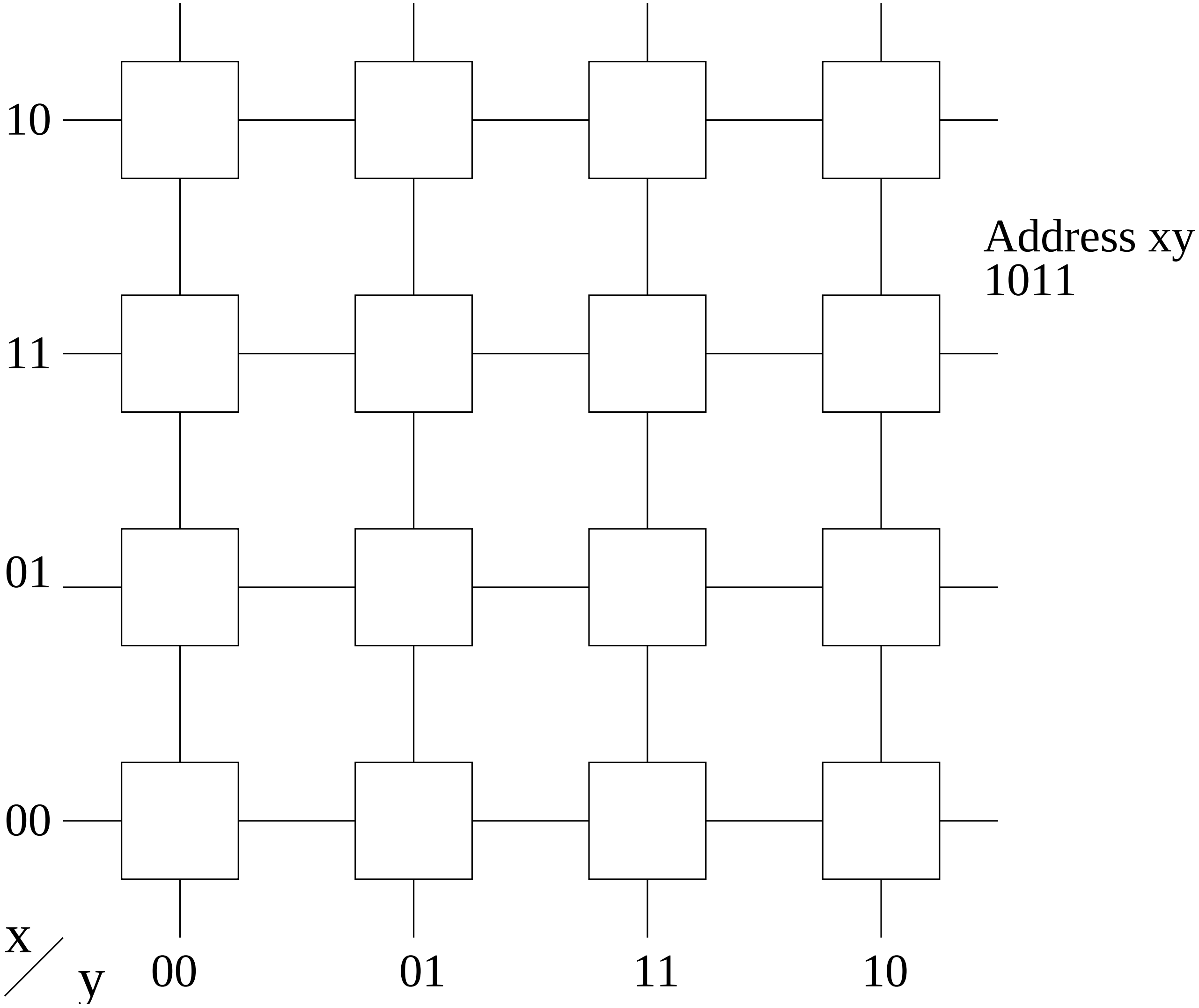
An *embedding* is an injective mapping of the nodes of one networks to the nodes of another network. It is a *perfect embedding* if the mapping is also surjective.

Example: Embedding a ring in a *torus*



..Embedding

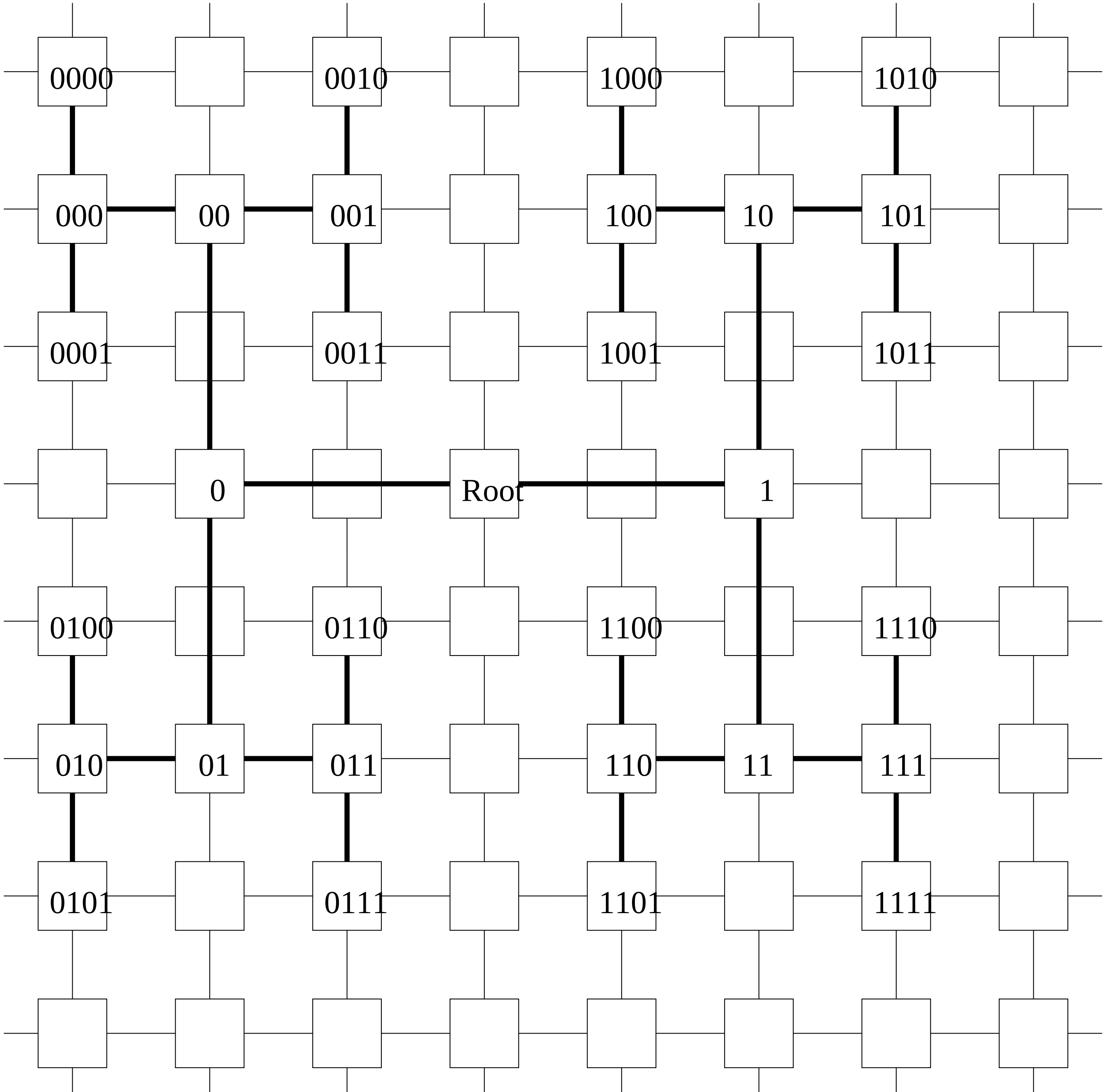
Example: Embedding a 2-dim mash (or torus) in a 4-dim hypercube



Notice: The embedding in the hypercube is based on one-bit change for each link.

..Embedding

Example: Embedding a binary tree into a 2-dim mash



Dilation

- The *dilation of an embedding A to B* is the maximum number of links in the embedding network B corresponding to a link of the embedded network A.
- It is used to measure the quality of the embedding.
- The dilation of the embedding of a tree into a mash may be arbitrarily large [given by the hight of the tree].

Communication methods

Circuit switching:

- In this case one has to establish a connection path between the source and the destination and to maintain all the links in the path till the message is completely sent.
- Example: Conventional (simple) telephone systems.
- Main problem: It is not too efficient as it forces all the links in the path to be reserved till the complete transfer.

..Communication methods

Packet switching: A *store-and-forward* method may be used:

- Messages are divided into *packets* of information, each including source and destination addresses for routing.
- One puts a limit on the *maximum size* of a packet, say 1000 bytes; if a message is longer, then it is split into more packets.
- Each node has a *buffer* to hold the packets before transferring them onward to the next node.

The usual mail system is based on this method. It may be more efficient than circuit switching as links may be used by a greater number of messages. However, it has a significant latency since packets must first be stored in buffers even in the case when the outgoing link is available.

..Communication methods

Virtual cut-through: This is an improvement of the previous method aiming to eliminate storage latency:

- If the outgoing link is available, then the message is immediately passed forward without being stored in the nodal buffer.

If a complete path is available, then the message is passed immediately through to the destination. If no such path is available, then storage is needed for a successful transmission.

..Communication methods

Wormhole routing: The messages are divided into smaller units called *flits (flow control digits)*. Only the head of the message is initially transmitted from source node to next node when connecting link is available. Subsequent flits of message are transmitted when links become available.

A *request/acknowledge system* is used to *pull flits along*. It uses an (extra) R/A wire between the sending node and the receiving node:

- R/A is reset to 0 by receiving node when it is ready to receive a flit (i.e., when its flit buffer is empty)
- R/A is set to 1 by sending node when the sending node is about to send a flit

Sending node must wait for R/A to become 0 before setting it to 1 and sending the flit. It knows that the data has been received when receiving node resets R/A to 0.

Deadlock

A tricky problem in network routing is the *deadlock*: it may occur when *packets cannot be forwarded because they are blocked by other packets waiting to be forwarded and these packets are blocked in a similar way such that actually* none of the packets can move.

Example:

- node 1 wishes to send to 3 via node 2;
- node 2 wishes to send to 4 via node 3;
- node 3 wishes to send to 1 via node 4;
- node 4 wishes to send to 2 via node 1;

A famous example was introduced by Dijkstra: *5 philosophers* may think or eat spaghetti using a circular table with 5 plates and 5 forks. Eating spaghetti requires two forks (!?). A deadlock appears when all philosophers are present and all held the right-hand side fork.

..Deadlock

A general solution to deadlock is to use *virtual channels*:

- multiple virtual channels are associated with a physical channel and
- they are time-multiplexed onto the physical channel.

Networked computers as a multicomputer

A *cluster of workstations* or a *network of workstations* offers a very attractive alternative to expensive supercomputers and parallel computer systems for high-performance computing. The key advantages are:

- high performance workstations and PCs available at low cost
- the latest processors can be easily incorporated into the system
- one may use existing (or slightly modified) software

Parallel programming tools for clusters

- *PVM (Parallel Virtual Machine)* - developed in late 1980s and very popular
- *MPI (Message-Passing Interface)* - standard defined in 1990s

Real networks of workstations

Ethernet

- common communication network for workstations
- it consists of a single wire to which all computer are attached
- an ethernet frame consists of : preamble (64b), destination address (48b), source address (48b), type (16b), data (variable length), frame check sequence (32b)

Main problems: collisions and significant message latency.

..Real networks of workstations

Ring structures:

- a collection of workstations connected in a ring. Example: Token rings

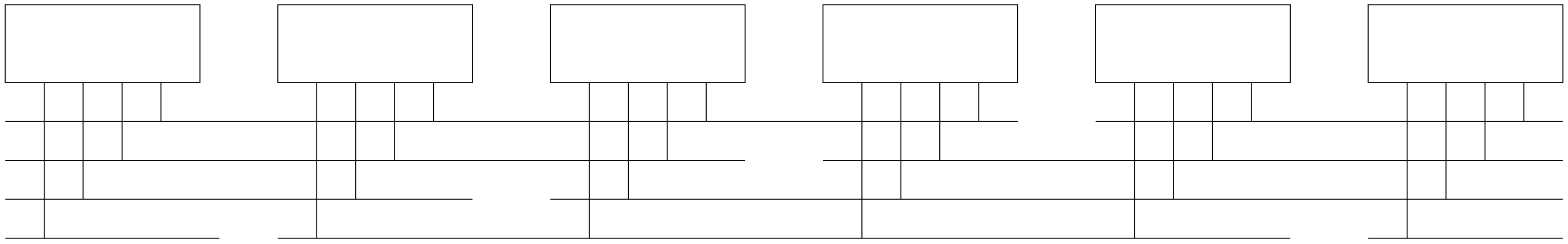
Point-to-point communication:

- this provides the highest intercommunication bandwidth;
- various point-to-point communications can be created using hubs and switches
- examples: HIPPI (High Performance Parallel Interface), Fast (100 MHz) and Gigabit Ethernet, and fiber optics.

..Real networks of workstations

Overlapping connectivity networks:

- there are regions of connectivity and they may overlap
- an example for Ethernet is



Performance analysis

Speedup factor:

- Denote by t_s the execution time on a single processor and by $t_p(n)$ the execution time on a n processor system. The speedup factor $S(n)$ is
$$S(n) = \frac{t_s}{t_p(n)}$$
Notice: The sequential and parallel parallel algorithms may be (and usually are) different.

- A different approach is to use the number of *computational steps*: if cs_s is the number of computational steps on a single processor and $cs_p(n)$ is the number of parallel computational steps on a n processor system, then $S_{cs}(n)$ is
$$S_{cs}(n) = \frac{cs_s}{cs_p(n)}$$

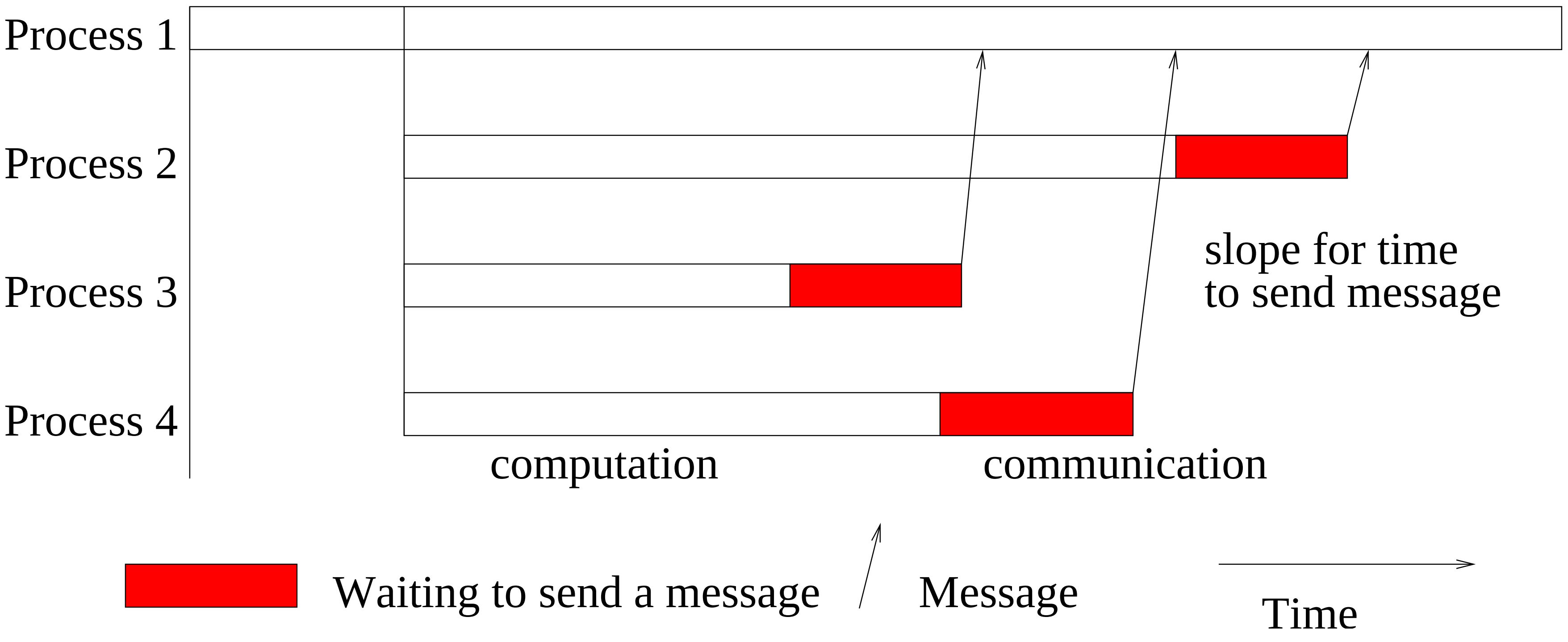
..Performance analysis

Superlinear speedup:

- the maximum speedup $S(n)$ is n, i.e., a *linear speedup*.
- sometimes there are cases when $S(n) > n$, known as "superlinear speedup" (e.g., in some search algorithms), but usually this is due to the use of an suboptimal sequential algorithm, or the use of some feature of the architecture that favors the parallel algorithm, or the use of extra memory in the multiprocessor system

..Performance analysis

Space-time diagram of a message passing program



An useful measure is *communication overhead*

$$\text{Communication overhead} = \frac{\text{Computation time}}{\text{Communication time}}$$

..Performance analysis

Amdahl's law: It allows to compute the maximum speedup for a problem having a serial factor f which can not be parallelized

- the speedup is

$$S(n) = \frac{t_s}{f t_s + \frac{(1-f)t_s}{n}} = \frac{n}{1+(n-1)f}$$

With an infinite number of processors the maximum speedup is limited to $\frac{1}{f}$. E.g., if 5% is serial, then the maximum speedup is 20, irrespective of the number of processors.

Notice: Take it with some care. It was used in 1960s to promote single processor systems.

..Performance analysis

Efficiency: it gives the fraction of time the processors are being used on computation.

$$E = \frac{\text{Execution time using one processor}}{\text{Execution time using a multiprocessor} \times \text{Number of processors}}$$

In symbols,

$$E(n) = \frac{t_s}{t_p(n) \times n} = \frac{S(n)}{n}$$

Notice: Usually it is given as a percentage, hence multiply the above result by 100.

..Performance analysis

Cost:

$$\text{Cost} = \text{Execution time} \times \text{Number of processors used}$$

- In the sequential case the cost is $C_s = t_s \times 1 = t_s$
- in the parallel case, $C_p(n) = t_p(n) \times n = \frac{nt_s}{S(n)} = \frac{t_s}{E(n)}$

Cost-optimal parallel algorithm: one in which the cost to solve the problem on a multiprocessor is proportional to the cost of solving it on a single processor system. (In other words, $E(n)$ is constant.)

..Performance analysis

Scalability: used to capture the situation when the increasing of the size of the system leads to a proportional increasing of the result.

- *hardware scalability* - larger hardware leads to proportional better performance
- *algorithmic scalability* - the parallel algorithm is such that increasing the input data leads to a bounded increase of computational steps

..Performance analysis

Gustafson's law: rather than assuming the problem size is fixed, we now assume that the parallel time is fixed and the serial part of the problem does not increase along with the problem size

- *Scaled speedup factor (or Gustafson's law)*

$$S_s(n) = \frac{s+np}{s+p} = s+np = n+(1-n)s$$

(here $0 < s, p < 1$ and $s+p = 1$)

Notice: For 5% and 20 processors this gives a better result than Amdahl's law (10.05 vs. 10.26) due to different assumptions.