Hardware architecture and basics of parallel computing

Gian Franco Marras
g.marras@cineca.it
Dipartimento Supercalcolo, Applicazioni e Innovazione (SCAI)
CINECA - High Performance System Group
Casalecchio di Reno (BO) – Italy
www.cineca.it - www.cineca.it/en/index.htm

Outline

- Parallel Architectures & Parallel Programming
- Message Passing Interface
  - Point-point comunication
  - Collective comunication
- OpenMP
  - Directives
  - Runtime Library Routines
  - Environment Variables

Parallel Architectures and Parallel Programming

An overview

What is a Supercomputer?

A supercomputer is a computer that is at the frontline of current processing capacity, particulary speed of calculation.

Supercomputers are used for highly calculation-intensive tasks such as problems involving quantum physics, weather forecasting, climate research, molecular modeling, physical simulations and a particular class of problems, known as Grand Challenge problems.

What is Grand Challenge Problem?

A grand challenge is a fundamental problem in science or engineering, with broad applications, whose solution would be enabled by the application of high performance computing resources that could become available in the near future.

Which are Grand Challenges today?

- Prediction of weather and global change
- Material science and Superconductivity
- Structural biology
- Human genome
- Astronomy
- Turbolence
- Design of hypersonic aircraft
- Geophysics
- ...

Parallel computing

Parallel computing is a form of computation in which many calculations are carried out simultaneously, operating on the principle that large problems can often be divided into smaller ones, which are then solved concurrently.

Parallelism has been employed for many years, mainly in high-performance computing

High-performance computer

High-performance computing uses supercomputers and computer clusters to solve advanced computation problems.
A list of the most powerfull HPC can be found on the TOP500 list (www.top500.org)

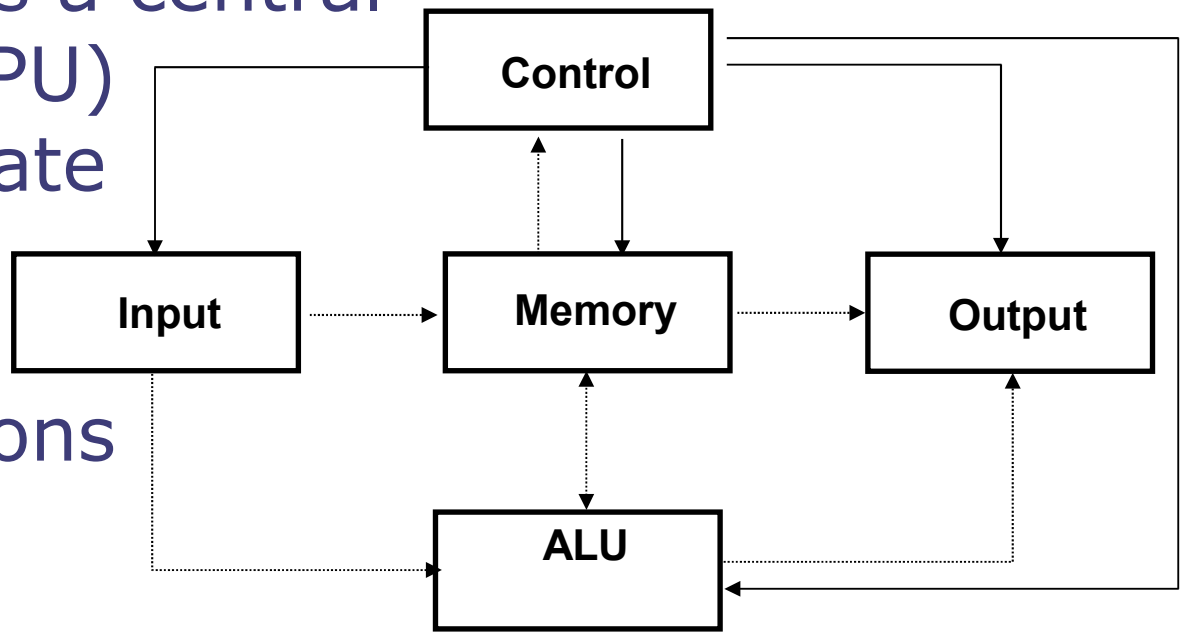
The TOP500 list ranks the world's 500 fastest high-performance computers, as measured by the High Performance Linpack (HPL) benchmark.

LINPACK benchmark

- LINPACK is a software library for performing numerical linear algebra on digital computers. It was written in Fortran.
- LINPACK use BLAS libraries for performing basic vector and matrix operations.
- The LINPACK Benchmarks measure how fast a computer solves a dense N by N system of linear equations **Ax = b**
- The result is reported in millions of floating point operations per second (MFLOP/s or FLOPS).

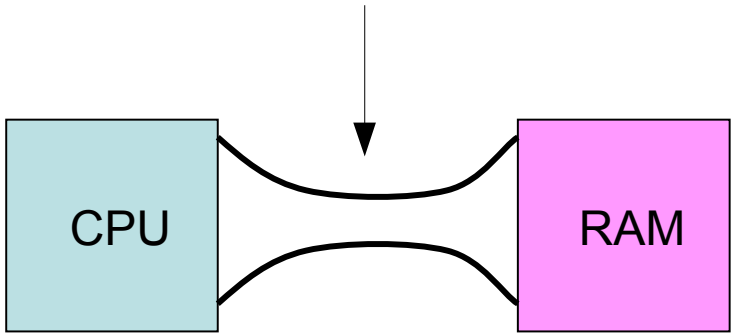
Computers & Von Neumann architecture

A computer is a programmable machine that receives input, stores and manipulates data and instructions, and provides output in a useful format. The von Neumann architecture is a design model for a stored-program digital computer that uses a central processing unit (CPU) and a single separate storage structure ("memory") to hold both instructions and data.



Von Neumann bottleneck

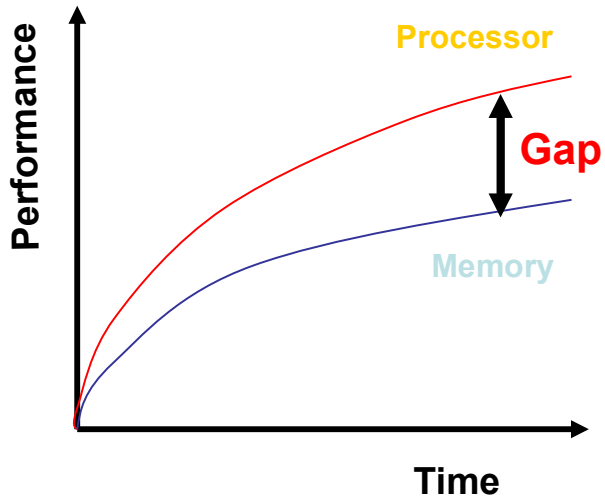
In most modern computers, throughput (data transfer rate) is much smaller than the rate at which the CPU can work.
The CPU is continuously forced to wait for needed data to be transferred to or from memory.
The performance problem can be alleviated providing one or more cache between the CPU and the main memory (Memory hierarchy).



Moore Law

The number of transistors that can be placed inexpensively on an integrated circuit has doubled approximately every 18 months. The trend has continued for more than half a century and is not expected to stop until 2015 or later.

Memory's performance has doubled every 6 years.



Cache

The system cache is responsible for a great deal of the system performance improvement of today's PCs.
The cache is a buffer of sorts between the very fast processor and the relatively slow memory that serves it.
The presence of the cache allows the processor to do its work while waiting for memory far less often than it otherwise would.

Principle of locality

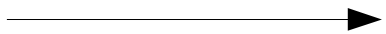
A hierarchical memory system is efficient if the mode of data access are predictable.

- **Temporal locality**: if at one point in time a particular memory location is referenced, then it is likely that the same location will be referenced again in the near future.
- **Spatial locality**: if a particular memory location is referenced at a particular time, then it is likely that nearby memory locations will be referenced in the near future.

Cache Memory

- **Temporal locality example:**

```
do i=1, n
  a(i) = b(i) + 1.0
enddo
do i=2, n
  c(i) = sqrt(a(i-1))
enddo
```



Loop Fusion

```
do i=1, n-1
  a(i) = b(i) + 1.0
  c(i+1) = sqrt(a(i))
enddo
a(n) = b(n) + 1.0
```

Cache Memory

- **Spatial locality example:**

```
do i=1, n
 do j=1, n
  a(i,j)=b(i,j)+ 1.0
 enddo
enddo
```

Loop Interchange

```
do j=1, n
 do i=1, n
  a(i,j) = b(i,j) + 1.0
 enddo
enddo
```

In Fortran the 2-dim arrays are stored by colomn.

Memory hierarchy

	Latency	Size
•Processor registers	1 CPU cycle	1 kbytes
•Level 1 (L1) cache	1~10 cycles	~128 kbytes
•Level 2 (L2) cache	2x to 10x L1	512 KB or more
•Level 3 (L3) cache	Higher latency	2048 KB or more
•Main memory	Hundreds of cycle	Multiple GB
•Disk storage	Millions of cycles	Multiple GB-TB

Modern Parallel Architectures

Basic architectural schemes:

Shared Memory

Distributed Memory

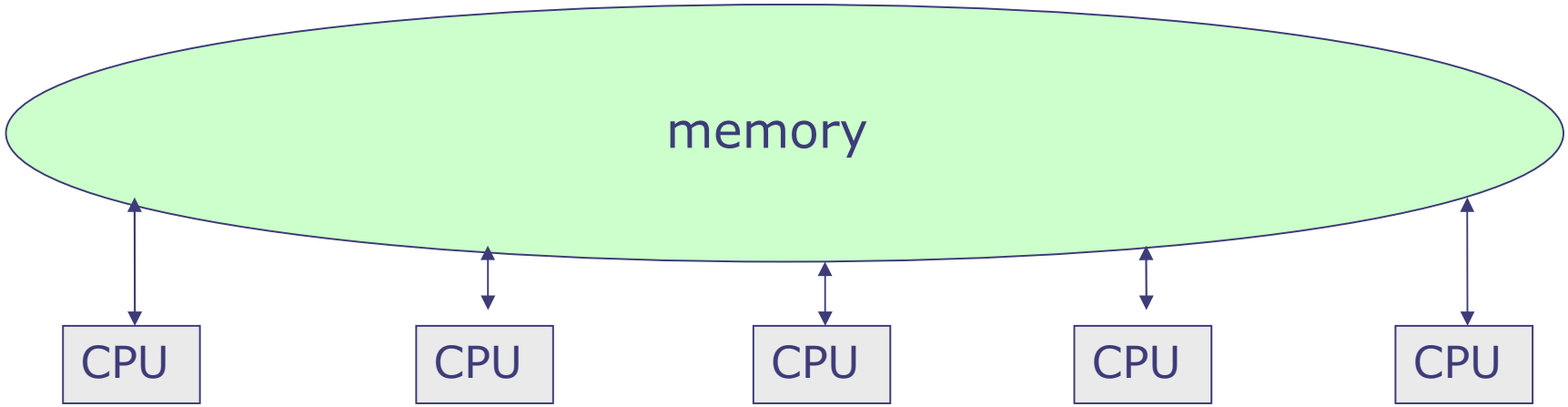
Nowadays most of the computer systems implements mixed architectures

Shared Memory System

Shared memory concept refers to a (typically) large block of RAM that can be accessed by several different CPUs at the same time in a multiple-processor environment.

A shared memory System offers a single memory space accesible by all processors.

Shared Memory



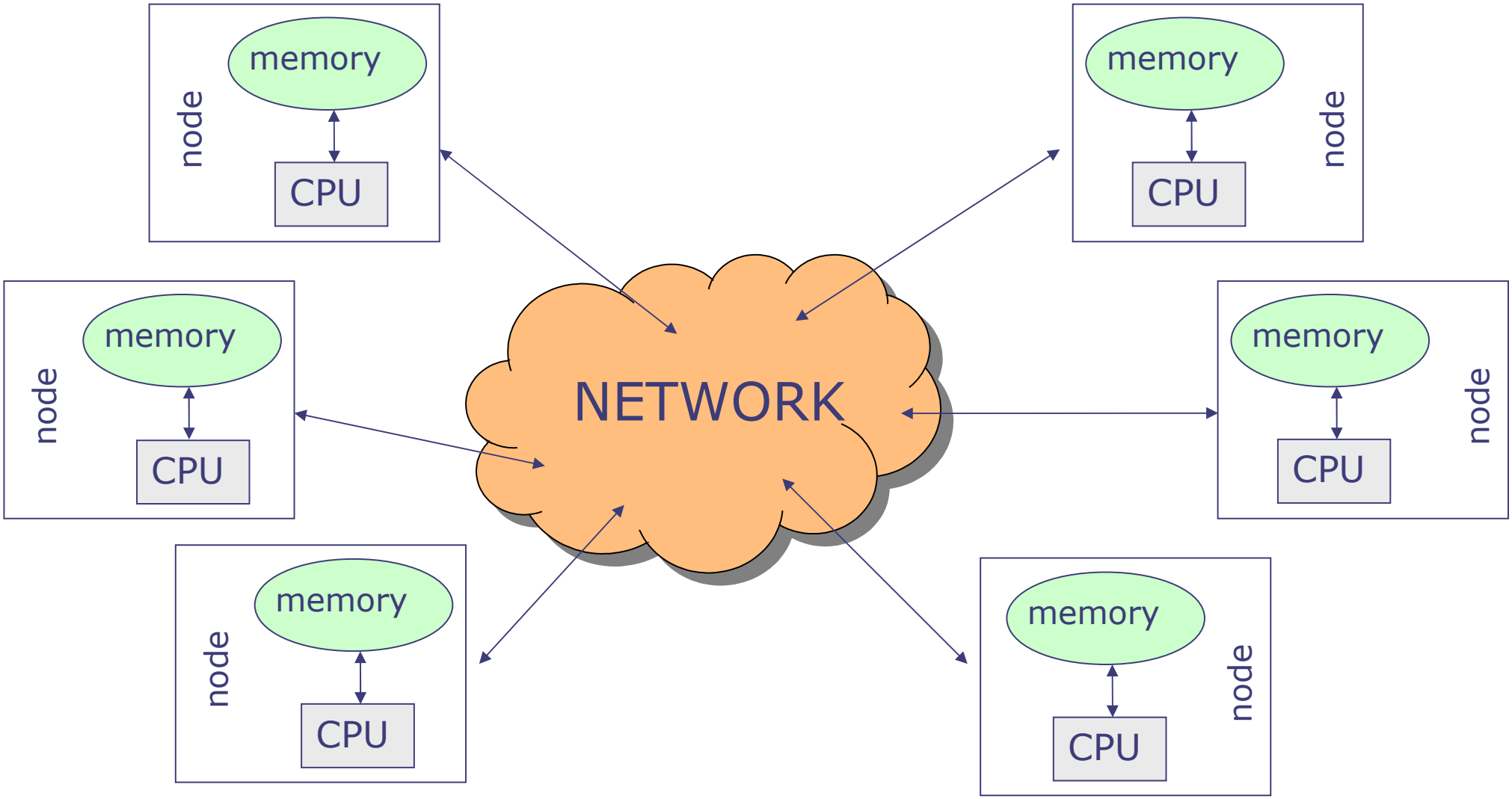
Distributed Memory System

Distributed memory concept refers to a multiple processor computer system where each processor has its own private memory.

Computational tasks can access only local data, and if any remote data is required, it must request them to remote processors.

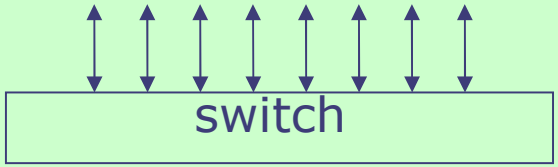
Computers are interconnected by communication channels that facilitate communications and allows to share data.

Distributed Memory

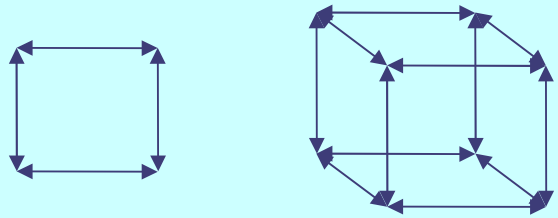


Most Common Networks

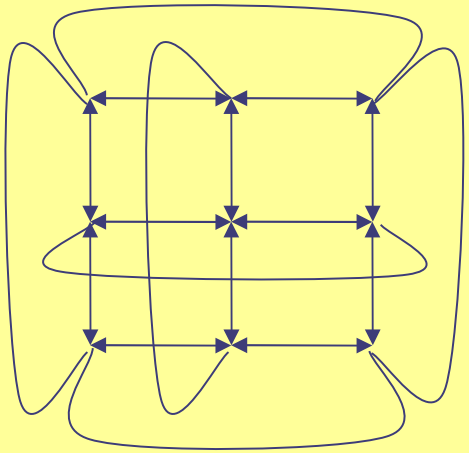
switched



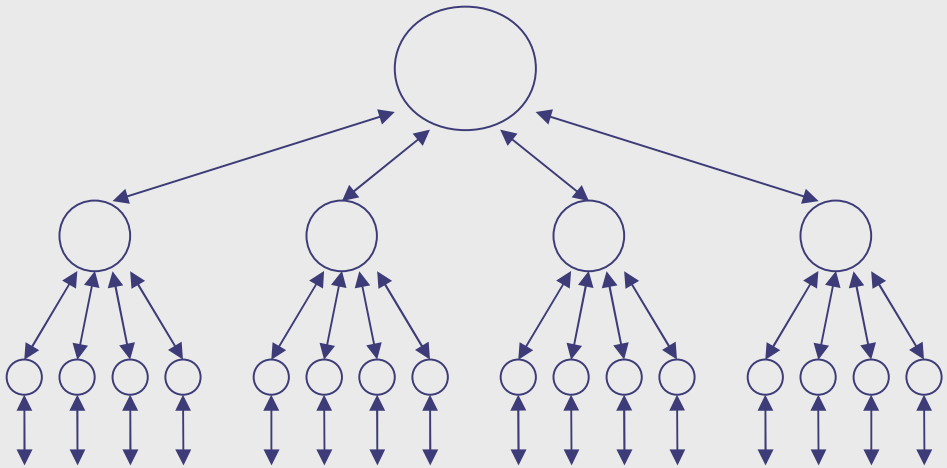
Cube, hypercube, n-cube



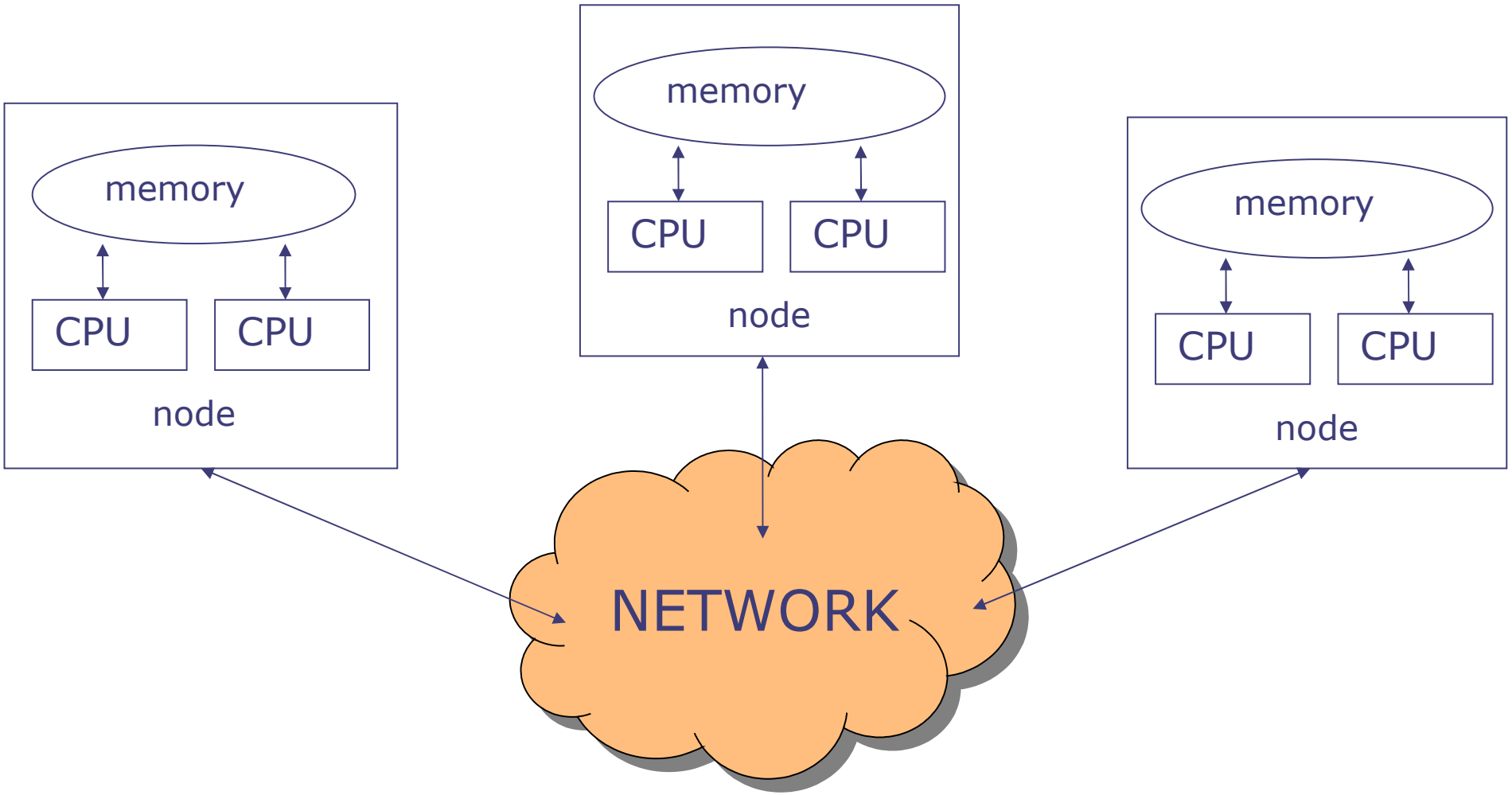
Torus in 1,2,...,N Dim



Fat Tree



Distribute Memory Architectures



Logical Machine Organization

The logical organization, seen by the programmer, could be different from the hardware architecture.

Its quite easy to logically partition a Shared Memory computer to reproduce a Distributed memory Computers.

The opposite is not true.

Parallel Programming Paradigms

The two architectures determine two basic scheme for parallel programming

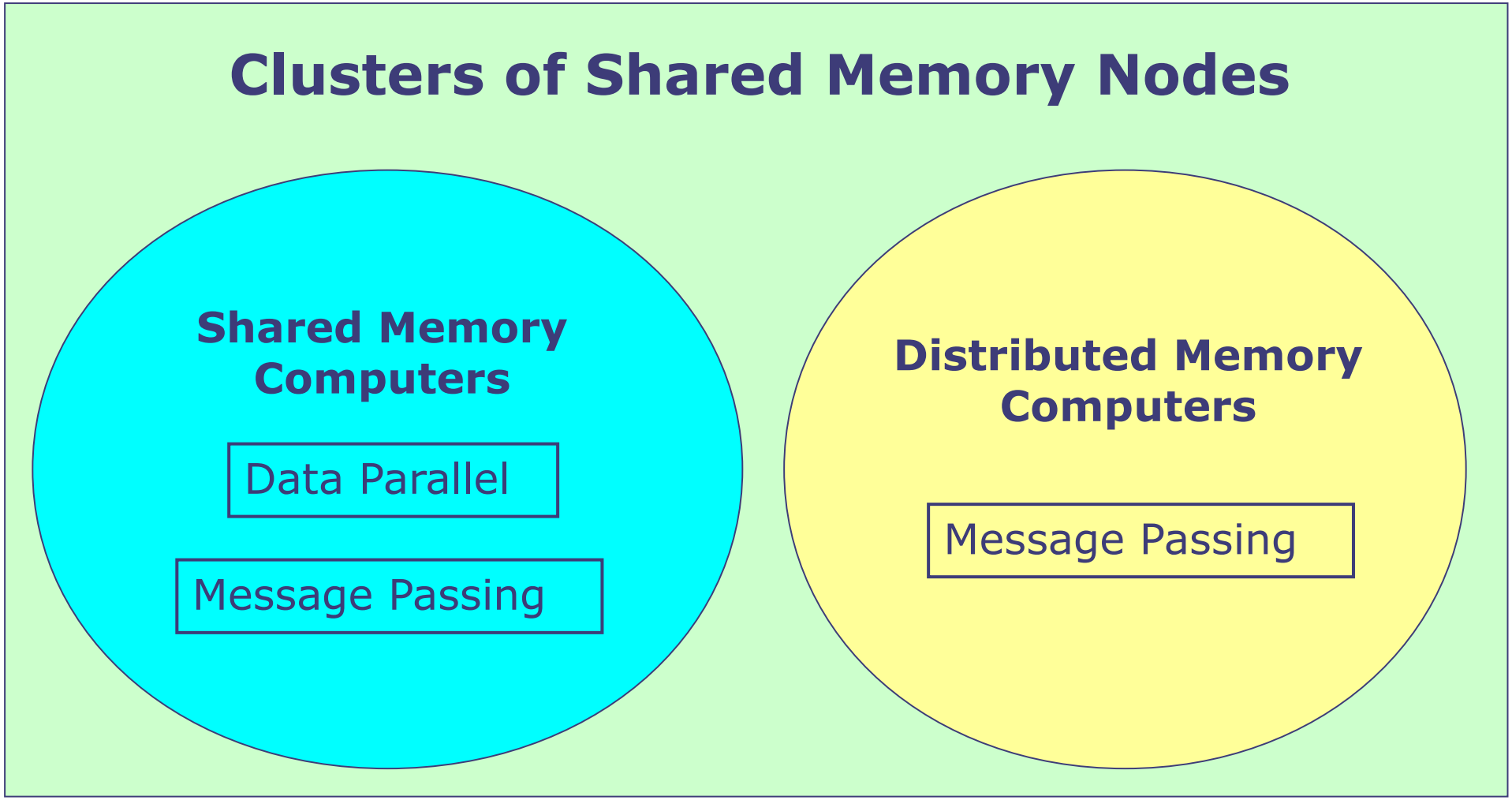
Message Passing (distributed memory)
all processes could **directly** access only their **local memory**; Sharing of data takes by explicitly sending and receiving data between processes.

Data Parallel (shared memory)
all processes (usually threads) could **directly** access the **whole memory**;

Parallel Programming Paradigms, cont.

Programming Environments	
Message Passing	**Data Parallel**
Standard compilers	Ad hoc compilers
Communication Libraries	Source code Directive
Ad hoc commands to run the program	Standard Unix shell to run the program
Standards: **MPI**	Standards: **OpenMP**

Architectures vs. Paradigms



Parallel programming Models for applications designing

…two basic models

Domain decomposition

Data are divided into pieces of approximately the same size and mapped to different processors. Each processors work only on its local data. The resulting code has a single flow.

Functional decomposition

The problem is decompose into a large number of smaller tasks and then the tasks are assigned to processors as they become available, Client-Server / Master-Slave paradigm.

Important Designing Clues

When writing a parallel code, regardless of the architecture, programming model and paradigm, be always aware of

- **Load Balancing**
- **Minimizing Communication**
- **Overlapping Communication and Computation**

Load Balancing

Equally divide the work among the available resource: processors, memory, network bandwidth, I/O, ...

This is usually a simple task for the problem decomposition model

It is a difficult task for the functional decomposition model

Minimizing Communication

When possible reduce the communication events:

Group lots of small communications into large one.

Eliminate synchronizations as much as possible. Each synchronization level off the performance to that of the slowest process.

Overlap Communication and Computation

When possible code your program in such a way that processes continue to do useful work while communicating.

This is usually a non trivial task and is afforded in the very last phase of parallelization.

This is possible with asynchronous communication.

If you succeed, you have done. Benefits are enormous.

Scalability

It's refered to the capability of a system to increase performance under an increased load when resources are added.

A system whose performance improves after adding hardware, proportionally to the capacity added, is said to be a *scalable system*.

Measures:

- **Speed Up**
- **Efficiency**
- **Amdhal's Law**

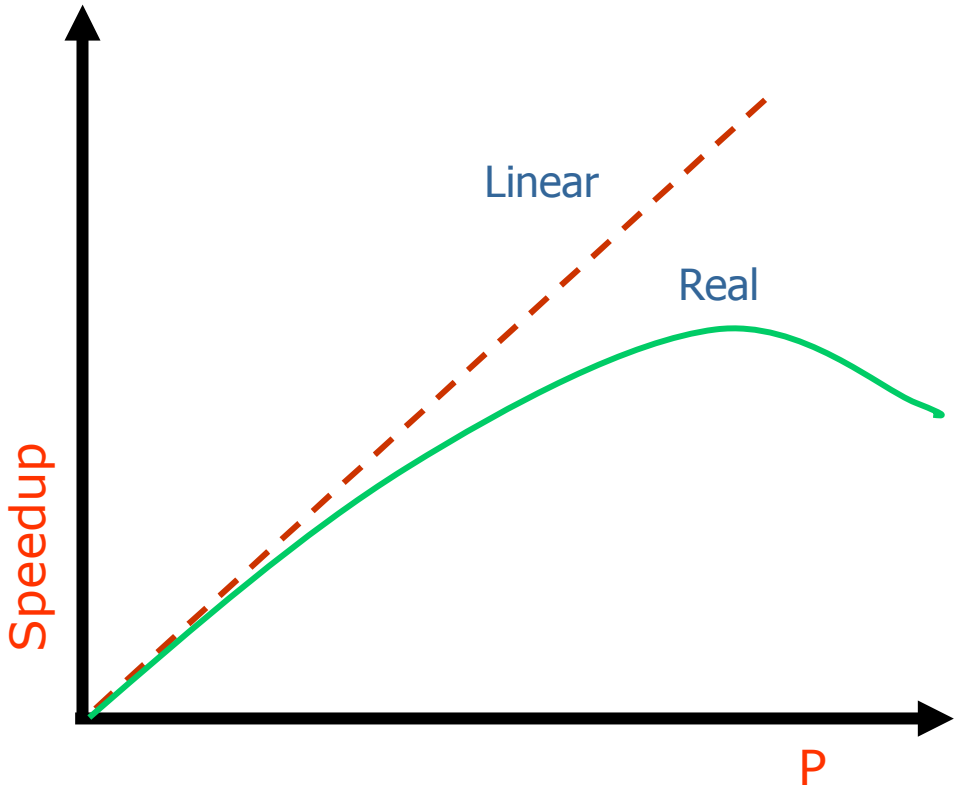
Speedup

Refers to how much a parallel algorithm is faster than a corresponding sequential algorithm.

$$S_p = T_s / T_p$$

where:

- **p** is the number of processors
- T_s is the execution time of the sequential algorithm
- T_p is the execution time of the parallel algorithm with p processors



Linear speedup is obtained when Sp = p.

Efficiency

$$E_p = S_p / P = T_s / PT_p$$

It is a value, typically between 0 and 1.
It estimates how well-utilized the processors are in solving the problem, compared to how much effort is wasted in communication and synchronization.

Amdhal's Law

It is used to predict the theoretical maximum speedup using multiple processors.

$$T_p(W,p)=T_s(W)\,f_s + (T_s(W)\,f_p / p)$$

where:

- **p** is the number of processors;
- T_s is the execution time of the sequential algorithm;
- T_p is the execution time of the parallel algorithm with p processors;
- f_s is the part that cannot be made parallel;
- $f_p = 1-f_s$ is the fraction of time that be made parallel.

Speedup $= 1 / ((1 - f_p) + f_p / p)$

Amdhal's Law

Amdhal's Law can be usefull for analize the scalability of a parallel system.

Maximun speedup <= $p / (1 + f_s (p - 1))$

Es: $f_s = 0.05$ ---> Max_speedup = 20 ~ $1/f_s$