## 2.5 Classification of Parallel Computers

### 2.5.1 Granularity

In parallel computing, granularity means the amount of computation in relation to communication or synchronisation

Periods of computation are typically separated from periods of communication by synchronization events.

- ***fine level*** (same operations with different data)
  - vector processors
  - instruction level parallelism
  - **fine-grain parallelism**:
    - Relatively small amounts of computational work are done between communication events
    - Low computation to communication ratio
    - Facilitates load balancing

  - Implies high communication overhead and less opportunity for performance enhancement
  - If granularity is too fine it is possible that the overhead required for communications and synchronization between tasks takes longer than the computation.

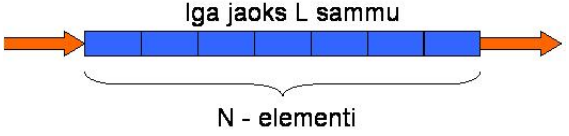
- ***operation level*** (different operations simultaneously)
- ***problem level*** (independent subtasks)
  - **coarse-grain parallelism**:
    - Relatively large amounts of computational work are done between communication/synchronization events
    - High computation to communication ratio
    - Implies more opportunity for performance increase
    - Harder to load balance efficiently

### 2.5.2 Hardware:

**Pipelining**

(was used in supercomputers, e.g. Cray-1)

In $N$ elements in pipeline and for $\forall$ element $L$ clock cycles $\Longrightarrow$ for calculation it would take $L+N$ cycles; without pipeline $L*N$ cycles




**Example of good code for pipelineing:**

```
do  i=1,k
   z(i)=x(i)+y(i)
end do
```

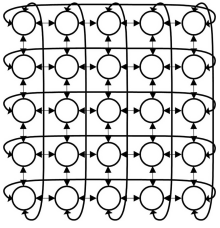
**Vector processors,**

fast vector operations (operations on arrays). Previous example good also for vector processor (vector addition) , but, e.g. recursion – hard to optimise for vector processors

Example: IntelMMX – simple vector processor.

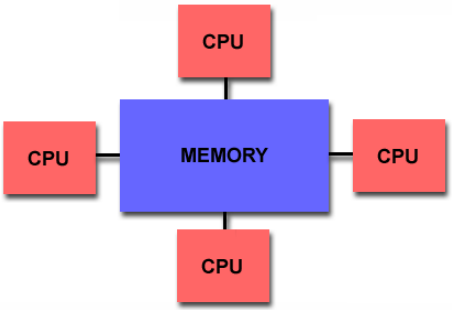
**Processor arrays**

Most often 2-dimensional arrays (For example: **MasPar MP2 - *massively parallel computer***)



MasPar-MP2: 128x128=16384 processors, each had 64Kbytes memory. each processor connected to its neighbours and on edges to the corresponding opposite edge nodes. Processors had mutual clock. Programming such a computer quite specific, special language, MPL, was used; need for thinkinking about communication between neighbours, or to all processor at once (which was slower).

### Shared memory computer




### Distributed systems

(e.g.: clusters) Most spread today.

One of the main questions on parallel hardware: do the processors share a mutual clock or not?

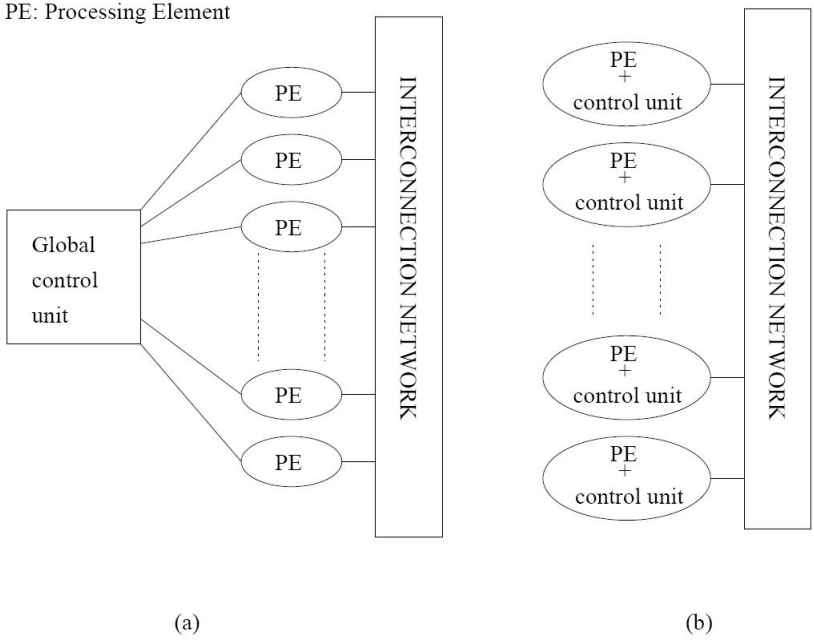



**Figure 2.3** A typical SIMD architecture (a) and a typical MIMD architecture (b).

## 2.6 Flynn's classification


**S** - Single

**M** - Multiple

**I** - Instruction

**Abbreviations:** **D** - Data

**For example: *Single Instruction Multiple Data stream***

=>:

**SISD** - single instruction single data stream, (e.g. simple PC)

**SIMD** - Same instructions applied to multiple data. (Example: MasPar)

**MISD** - same data used to perform multiple operations... Sometimes have been considered vector processors belonging here but most often said to be empty class

**MIMD** - Separate data and separate instructions. (Example: computer cluster)

### 2.6.1 SISD

- A serial (non-parallel) computer
- Single Instruction: Only one instruction stream is being acted on by the CPU during any one clock cycle
- Single Data: Only one data stream is being used as input during any one clock cycle
- Deterministic execution
- This is the oldest and even today, the most common type of computer
- Examples: older generation mainframes, minicomputers and workstations; most modern day PCs.

### 2.6.2 SIMD

- A type of parallel computer
- Single Instruction: All processing units execute the same instruction at any given clock cycle
- Multiple Data: Each processing unit can operate on a different data element
- Best suited for specialized problems characterized by a high degree of regularity, such as graphics/image processing.
- Synchronous (lockstep) and deterministic execution
- Two varieties: Processor Arrays and Vector Pipelines
- Examples:some early computers of this type:
    - Processor Arrays: Connection Machine CM-2, MasPar MP-1 & MP-2, ILLIAC IV

  - Vector Pipelines: IBM 9000, Cray X-MP, Y-MP & C90, Fujitsu VP, NEC SX-2, Hitachi S820, ETA10

- graphics cards
- Most modern computers, particularly those with graphics processor units (GPUs) employ SIMD instructions and execution units.
- possibility to switch off some processors (with mask arrays)

### 2.6.3 (MISD):

- A type of parallel computer
- Multiple Instruction: Each processing unit operates on the data independently via separate instruction streams.
- Single Data: A single data stream is fed into multiple processing units.
- Few actual examples of this class of parallel computer have ever existed. One is the experimental Carnegie-Mellon C.mmp computer (1971).
- Some conceivable uses might be:
  - multiple frequency filters operating on a single signal stream
  - multiple cryptography algorithms attempting to crack a single coded message.

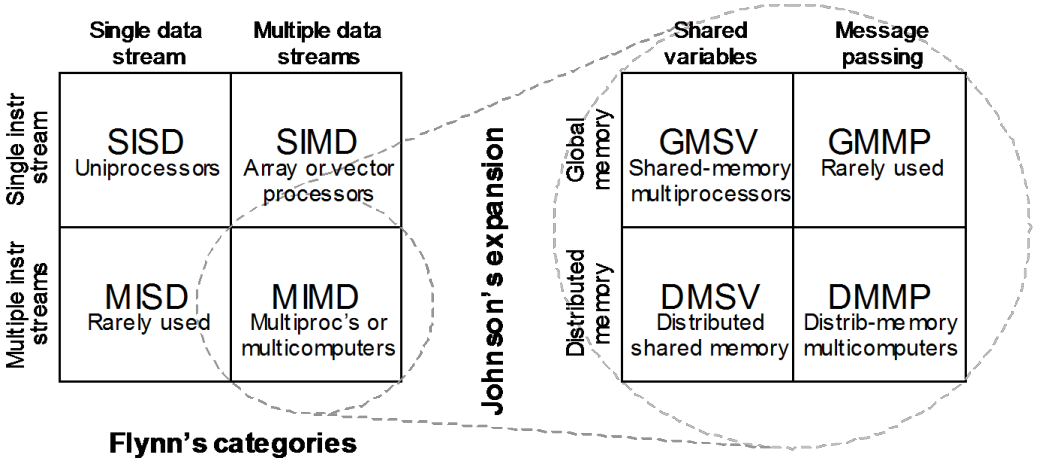
### 2.6.4 MIMD

- A type of parallel computer
- Multiple Instruction: Every processor may be executing a different instruction stream
- Multiple Data: Every processor may be working with a different data stream
- Execution can be synchronous or asynchronous, deterministic or non-deterministic
- Currently, the most common type of parallel computer - most modern super-computers fall into this category.
- Examples: most current supercomputers, networked parallel computer clusters and "grids", multi-processor SMP computers, multi-core PCs.
- Note: many MIMD architectures also include SIMD execution sub-components

### 2.6.5 Comparing SIMD with MIMD

- SIMD have less hardware units than MIMD (single instruction unit)
- Nevertheless, as SIMD computers are specially designed, they tend to be expensive and timeconsuming to develop
- not all applications suitable for SIMD
- Platforms supporting SPMD can be built from cheaper components

### 2.6.6 Flynn-Johnson classification

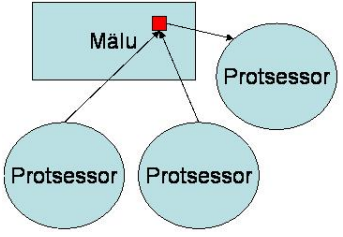



(picture by: Behrooz Parhami)

# 2.7 Type of memory access

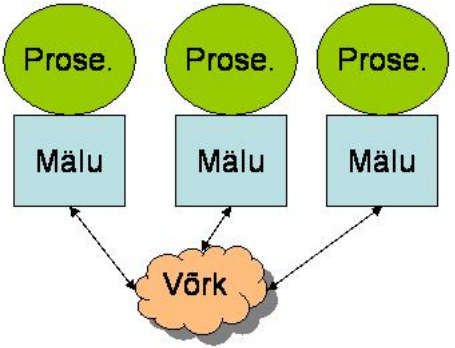
## 2.7.1 Shared Memory

- common shared memory
- Problem occurs when more than one process want to write to (or read from) the same memory address
- Shared memory programming models do deal with these situations



### 2.7.2 Distributed memory

- Networked processors with their private memory




### 2.7.3 hybrid memory models

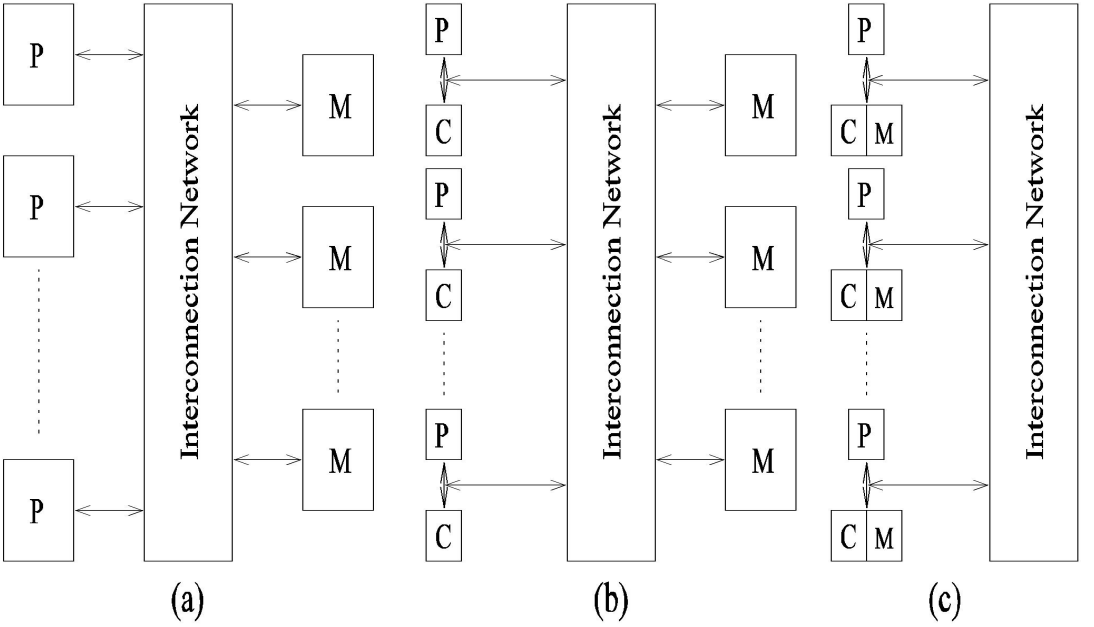
- E.g. *distributed shared memory,* **SGI Origin 2000**

# 2.8 Communication model of parallel computers

## 2.8.1 Communication through shared memory address space

- **UMA (*uniform memory access*)**
- **NUMA (*non-uniform memory access*)**
  - SGI Origin 2000
  - Sun Ultra HPC

**Comparing UMA and NUMA:**




C - Cache, P- Processor, M-Memory / (a) & (b) - UMA, (c) - NUMA

### 2.8.2 Communication through messages

- Using some messaging libraries like MPI, PVM.
- All processors are
  - independent
  - own private memory
  - have unique ID
- Communication is performed through exchanging messages

## 2.9 Other classifications

### 2.9.1 Algorithm realisation

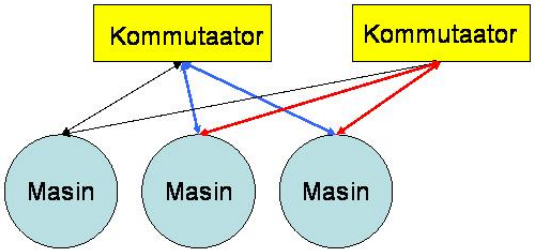
- using only hardware modules
- mixed modules (hardware and software)

### 2.9.2 Control type

1. synchronous
2. dataflow-driven
3. asynchronous

### 2.9.3 Network connection speed

- network bandwidth
  - can be increased e.g. through channel bonding

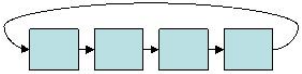



- network latency
  - the time from the source sending a message to the destination receiving it

More easy to increase bandwidth!

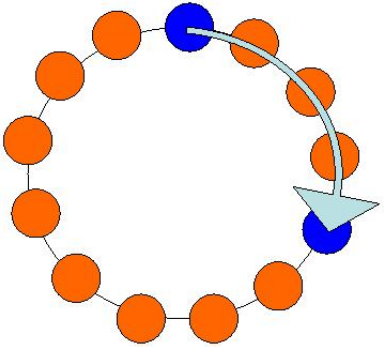
### 2.9.4 Network topology

- Bus-based networks
- Ring - one of the simplest topologies

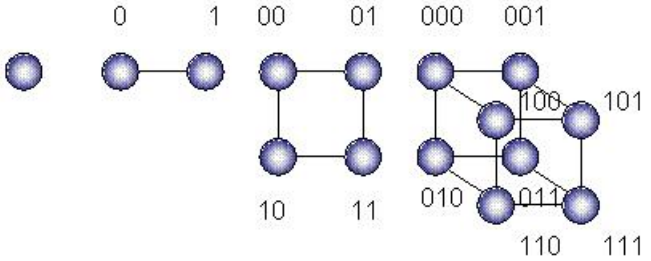


- Array topology
  - Example: cube (in case of 3D array)
- Hypercube
  - In ring the longest route between 2 processors is: $P/2$ – in hypercube – $\log P$

Ring:



Hypercube:



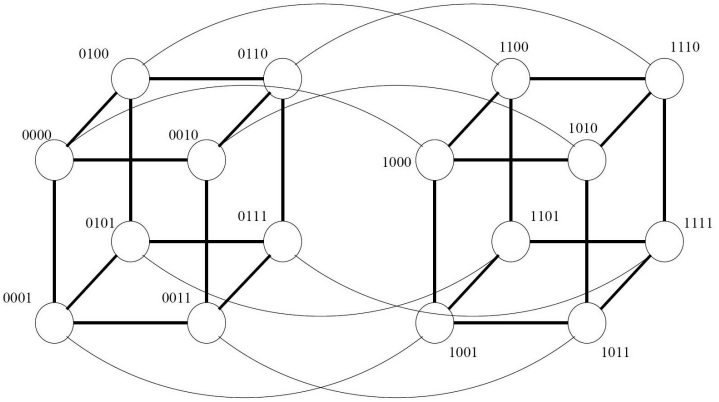
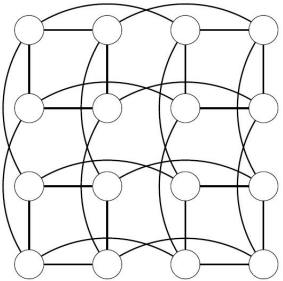


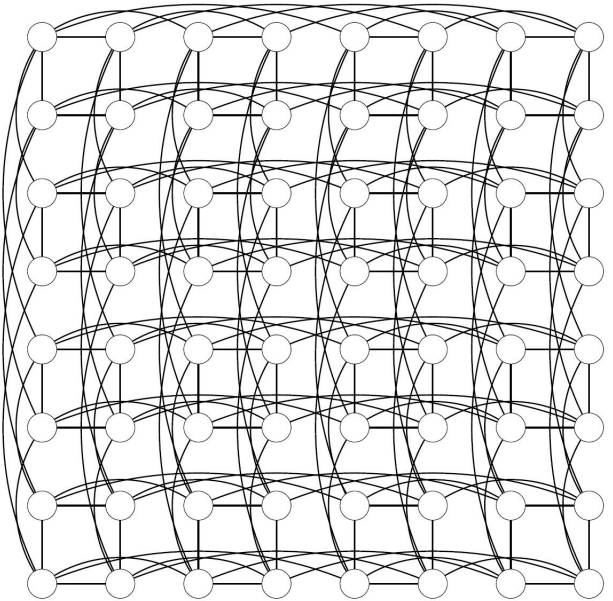

Figure. How to design a hypercube: Add similar structure and connect corresponding nodes (adding one 1 bit).

Problem: large number of connections per node

Easy to emulate Hypercube on e.g. MasPar array in $\log n$ time:

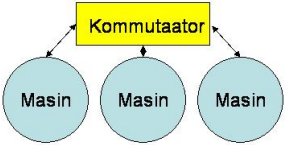


(a) $P = 16$



(b) $P = 32$

**Figure 2.33** Embedding a hypercube into a 2-D mesh.




- Star topology
  - Speed depends very much on switch properties *(e.g. latency, bandwidth, backplane frequency*) and ability to cope with arbitray communication patterns
- Clos-network

For example Myrinet. (quite popular around 2005)

- Cheaper but with higher latency: Gbit Ethernet., (10 Gbit Ethernet)
- Nowadays, most popular low-latency network type – Infiniband