

## ARCHITECTURES FOR PARALLEL COMPUTATION

1. Why Parallel Computation
2. Parallel Programs
3. A Classification of Computer Architectures
4. Performance of Parallel Architectures
5. The Interconnection Network
6. Array Processors
7. Multiprocessors
8. Multicomputers
9. Vector Processors
10. Multimedia Extensions to Microprocessors



## Why Parallel Computation?

- The need for high performance!

Two main factors contribute to high performance of modern processors:

1. Fast circuit technology
2. Architectural features:
  - large caches
  - multiple fast buses
  - pipelining
  - superscalar architectures (multiple funct. units)

### However

- Computers running with a single CPU, often are not able to meet performance needs in certain areas:
  - Fluid flow analysis and aerodynamics;
  - Simulation of large complex systems, for example in physics, economy, biology, technic;
  - Computer aided design;
  - Multimedia.
- Applications in the above domains are characterized by a very high amount of numerical computation and/or a high quantity of input data.



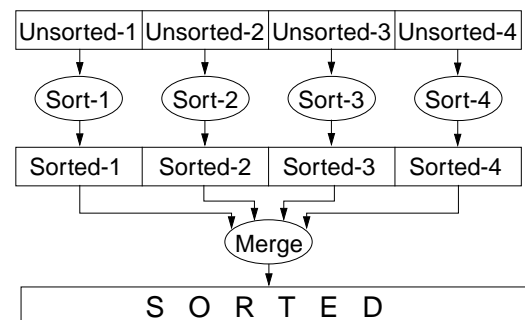
## A Solution: Parallel Computers

- One solution to the need for high performance: architectures in which *several CPUs* are running in order to solve *a certain application*.
- Such computers have been organized in very different ways. Some key features:
  - number and complexity of individual CPUs
  - availability of common (shared memory)
  - interconnection topology
  - performance of interconnection network
  - I/O devices
  - .....



## Parallel Programs

### 1. Parallel sorting



### Parallel Programs (cont'd)

A possible program for parallel sorting:

```

var t: array[1..1000] of integer;
- - - - -
procedure sort(i,j:integer);
  -sort elements between t[i] and t[j]-
end sort;
procedure merge;
  - - merge the four sub-arrays - -
end merge;

- - - - -
begin
  - - - - -
  cobegin
    sort(1,250) |
    sort(251,500) |
    sort(501,750) |
    sort(751,1000)
  coend;
  merge;
  - - - - -
end;

```



### Parallel Programs (cont'd)

2. Matrix addition:

$$\begin{array}{c|ccc}
 a_{11} & a_{21} & \cdots & a_{m1} \\
 a_{12} & a_{22} & \cdots & a_{m2} \\
 a_{13} & a_{23} & \cdots & a_{m3} \\
 \cdots & \cdots & \cdots & \cdots \\
 a_{1n} & a_{2n} & \cdots & a_{mn}
 \end{array}
 +
 \begin{array}{c|ccc}
 b_{11} & b_{21} & \cdots & b_{m1} \\
 b_{12} & b_{22} & \cdots & b_{m2} \\
 b_{13} & b_{23} & \cdots & b_{m3} \\
 \cdots & \cdots & \cdots & \cdots \\
 b_{1n} & b_{2n} & \cdots & b_{mn}
 \end{array}
 =
 \begin{array}{c|ccc}
 c_{11} & c_{21} & \cdots & c_{m1} \\
 c_{12} & c_{22} & \cdots & c_{m2} \\
 c_{13} & c_{23} & \cdots & c_{m3} \\
 \cdots & \cdots & \cdots & \cdots \\
 c_{1n} & c_{2n} & \cdots & c_{mn}
 \end{array}$$

```

var a: array[1..n,1..m] of integer;
    b: array[1..n,1..m] of integer;
    c: array[1..n,1..m] of integer;
    i:integer
- - - - -

begin
  - - - - -
  for i:=1 to n do
    for j:= 1 to m do
      c[i,j]:=a[i,j]+b[i,j];
    end for
  end for
  - - - - -
end;

```



### Parallel Programs (cont'd)

Matrix addition - parallel version:

```

var a: array[1..n,1..m] of integer;
    b: array[1..n,1..m] of integer;
    c: array[1..n,1..m] of integer;
    i:integer
- - - - -
procedure add_vector(n_ln:integer);
  var j:integer
begin
  for j:=1 to m do
    c[n_ln,j]:=a[n_ln,j]+b[n_ln,j];
  end for
end add_vector;

begin
  - - - - -
  cobegin for i:=1 to n do
    add_vector(i);
  coend;
  - - - - -
end;

```



### Parallel Programs (cont'd)

Matrix addition - vector computation version:

```

var a: array[1..n,1..m] of integer;
    b: array[1..n,1..m] of integer;
    c: array[1..n,1..m] of integer;
    i,j:integer
- - - - -

begin
  - - - - -
  for i:=1 to n do
    c[i,1:m]:=a[i,1:m]+b[i,1:m];
  end for;
  - - - - -
end;

Or even so:

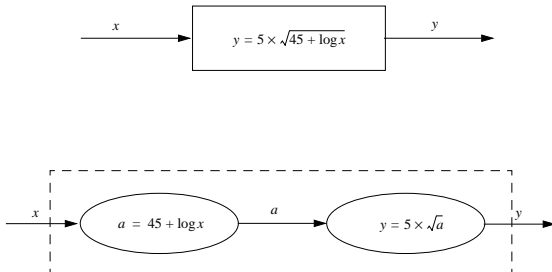
begin
  - - - - -
  c[1:n,1:m]:=a[1:n,1:m]+b[1:n,1:m];
  - - - - -
end;

```



## Parallel Programs (cont'd)

Pipeline model computation:



## Parallel Programs (cont'd)

A program for the previous computation:

```

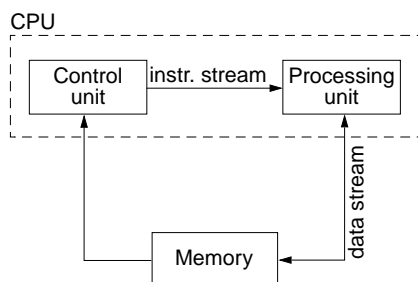
channel ch:real;
- - - - -
cobegin
  var x:real;
  while true do
    read(x);
    send(ch, 45+log(x));
  end while |
  var v:real;
  while true do
    receive(ch, v);
    write(5*sqrt(v));
  end while
coend;
- - - - -

```

## Flynn's Classification of Computer Architectures

- Flynn's classification is based on the nature of the instruction flow executed by the computer and that of the data flow on which the instructions operate.

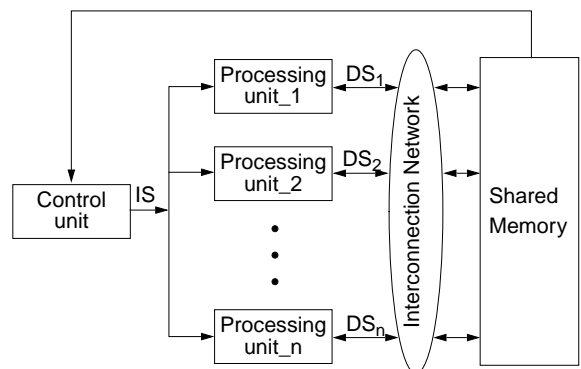
### 1. Single Instruction stream, Single Data stream (SISD)



## Flynn's Classification (cont'd)

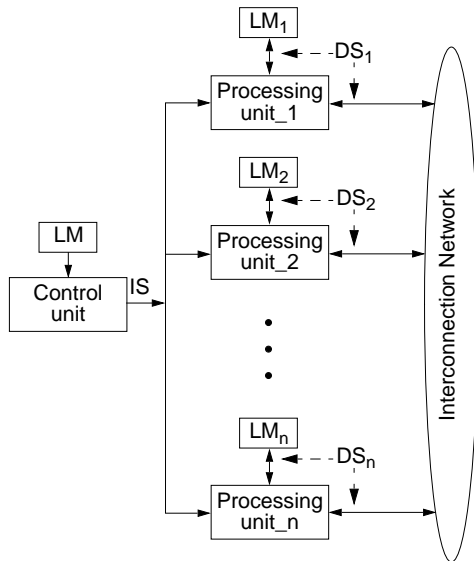
### 2. Single Instruction stream, Multiple Data stream (SIMD)

SIMD with shared memory



### Flynn's Classification (cont'd)

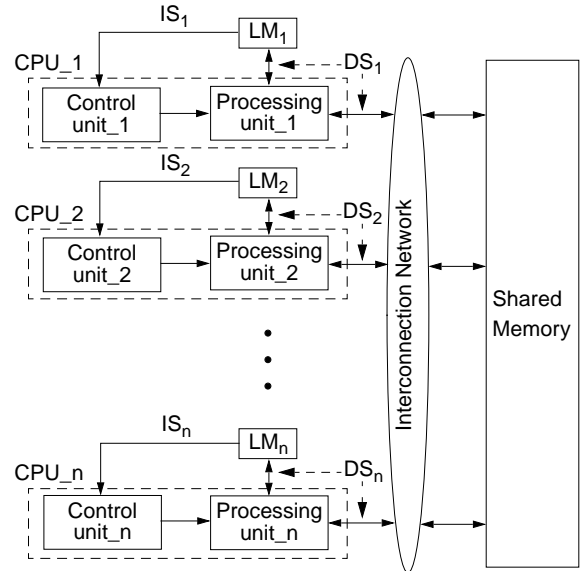
#### SIMD with no shared memory



### Flynn's Classification (cont'd)

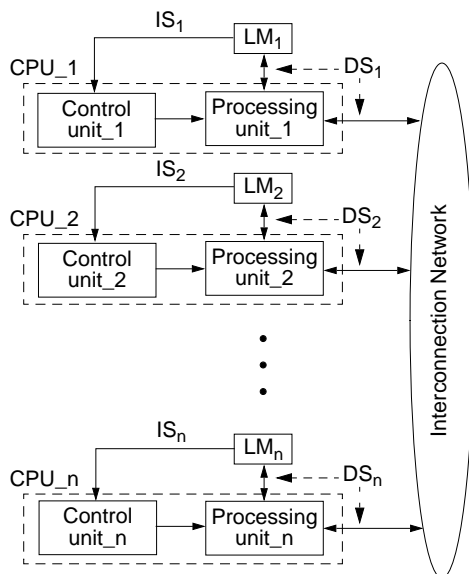
#### 3. Multiple Instruction stream, Multiple Data stream (MIMD)

#### MIMD with shared memory



### Flynn's Classification (cont'd)

#### MIMD with no shared memory



### Performance of Parallel Architectures

#### Important questions:

- How fast runs a parallel computer at its *maximal* potential?
- How fast execution can we expect from a parallel computer for a *concrete application*?
- How do we measure the performance of a parallel computer and the performance improvement we get by using such a computer?

## Performance Metrics

- **Peak rate:** the maximal computation rate that can be theoretically achieved when all modules are fully utilized.

The peak rate is of no practical significance for the user. It is mostly used by vendor companies for marketing of their computers.

- **Speedup:** measures the gain we get by using a certain parallel computer to run a given parallel program in order to solve a specific problem.

$$S = \frac{T_S}{T_P}$$

$T_S$ : execution time needed with the best sequential algorithm;

$T_P$ : execution time needed with the parallel algorithm.



## Performance Metrics (cont'd)

- **Efficiency:** this metric relates the speedup to the number of processors used; by this it provides a measure of the efficiency with which the processors are used.

$$E = \frac{S}{p}$$

S: speedup;

p: number of processors.

For the ideal situation, in theory:

$$S = \frac{T_S}{T_S/p} = p; \quad \text{which means} \quad E = 1$$

*Practically the ideal efficiency of 1 can not be achieved!*

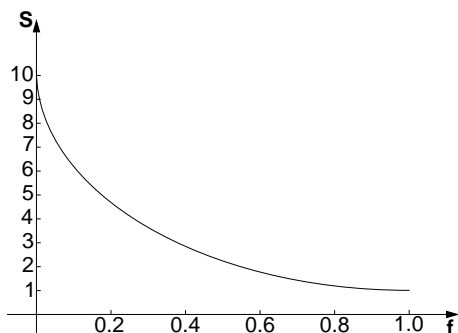


## Amdahl's Law

- Consider  $f$  to be the ratio of computations that, according to the algorithm, have to be executed sequentially ( $0 \leq f \leq 1$ );  $p$  is the number of processors;

$$T_P = f \times T_S + \frac{(1-f) \times T_S}{p}$$

$$S = \frac{T_S}{f \times T_S + (1-f) \times \frac{T_S}{p}} = \frac{1}{f + \frac{(1-f)}{p}}$$



## Amdahl's Law (cont'd)

**Amdahl's law:** even a little ratio of sequential computation imposes a certain limit to speedup; a higher speedup than  $1/f$  can not be achieved, regardless the number of processors.

$$E = \frac{S}{P} = \frac{1}{f \times (p-1) + 1}$$



To efficiently exploit a high number of processors,  $f$  must be small (the algorithm has to be highly parallel).



### Other Aspects which Limit the Speedup

- Beside the intrinsic sequentiality of some parts of an algorithm there are also other factors that limit the achievable speedup:
  - communication cost
  - load balancing of processors
  - costs of creating and scheduling processes
  - I/O operations
- There are many algorithms with a high degree of parallelism; for such algorithms the value of  $f$  is very small and can be ignored. These algorithms are suited for massively parallel systems; in such cases the other limiting factors, like the cost of communications, become critical.



### Efficiency and Communication Cost

Consider a highly parallel computation, so that  $f$  (the ratio of sequential computations) can be neglected.

We define  $f_c$ , the fractional communication overhead of a processor:

$T_{calc}$ : time that a processor executes computations;

$T_{comm}$ : time that a processor is idle because of communication;

$$f_c = \frac{T_{comm}}{T_{calc}}$$

$$T_p = \frac{T_S}{p} \times (1 + f_c)$$

$$S = \frac{T_S}{T_p} = \frac{p}{1 + f_c}$$

$$E = \frac{1}{1 + f_c} \approx 1 - f_c$$

- With algorithms that have a high degree of parallelism, massively parallel computers, consisting of large number of processors, can be efficiently used if  $f_c$  is small; this means that the time spent by a processor for communication has to be small compared to its effective time of computation.
- In order to keep  $f_c$  reasonably small, the size of processes can not go below a certain limit.



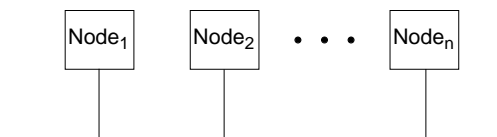
### The Interconnection Network

- The interconnection network (IN) is a key component of the architecture. It has a decisive influence on the overall performance and cost.
- The traffic in the IN consists of data transfer and transfer of commands and requests.
- The key parameters of the IN are
  - total bandwidth: transferred bits/second
  - cost



### The Interconnection Network (cont'd)

#### Single Bus

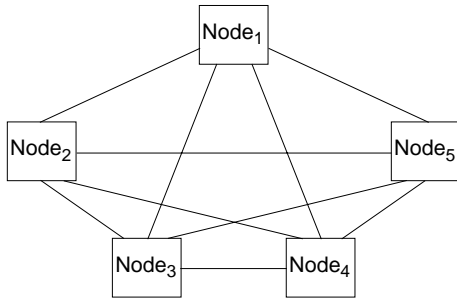


- Single bus networks are simple and cheap.
- One single communication is allowed at a time; the bandwidth is shared by all nodes.
- Performance is relatively poor.
- In order to keep a certain performance, the number of nodes is limited (16 - 20).



### The Interconnection Network (cont'd)

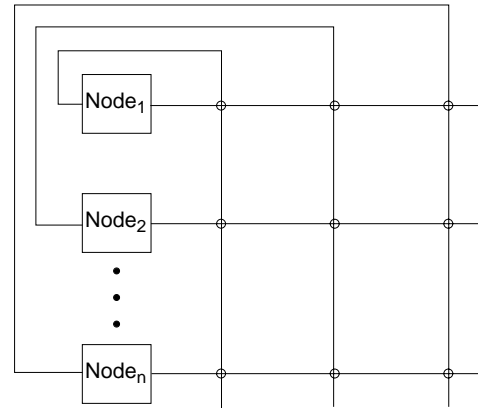
#### Completely connected network



- Each node is connected to every other one.
- Communications can be performed in parallel between any pair of nodes.
- Both performance and cost are high.
- Cost increases rapidly with number of nodes.

### The Interconnection Network (cont'd)

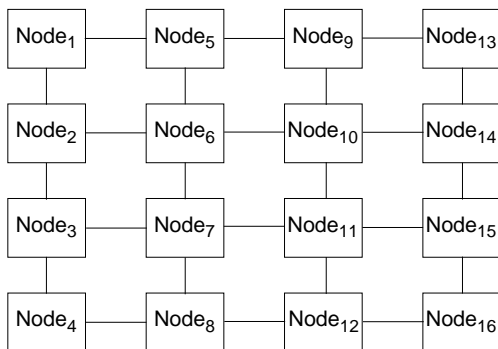
#### Crossbar network



- The crossbar is a dynamic network: the interconnection topology can be modified by positioning of switches.
- The crossbar switch is completely connected: any node can be directly connected to any other.
- Fewer interconnections are needed than for the static completely connected network; however, a large number of switches is needed.
- A large number of communications can be performed in parallel (one certain node can receive or send only one data at a time).

### The Interconnection Network (cont'd)

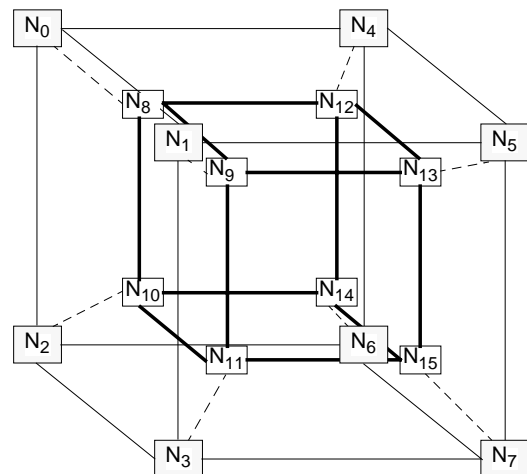
#### Mesh network



- Mesh networks are cheaper than completely connected ones and provide relatively good performance.
- In order to transmit an information between certain nodes, routing through intermediate nodes is needed (maximum  $2*(n-1)$  intermediates for an  $n*n$  mesh).
- It is possible to provide wraparound connections: between nodes 1 and 13, 2 and 14, etc.
- Three dimensional meshes have been also implemented.

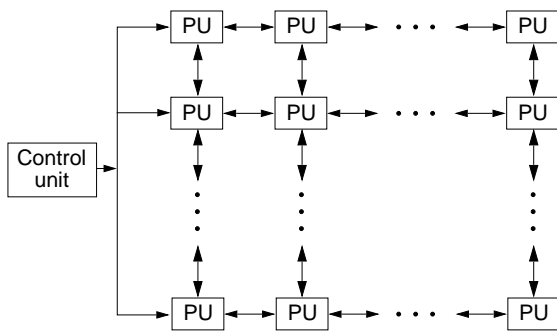
### The Interconnection Network (cont'd)

#### Hypercube network



- $2^n$  nodes are arranged in an  $n$ -dimensional cube. Each node is connected to  $n$  neighbours.
- In order to transmit an information between certain nodes, routing through intermediate nodes is needed (maximum  $n$  intermediates).

### SIMD Computers

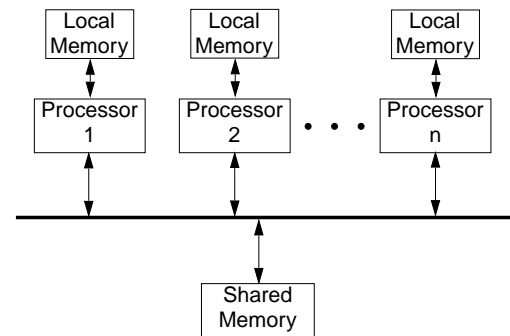


- SIMD computers are usually called *array processors*.
- PU's are usually very simple: an ALU which executes the instruction broadcast by the CU, a few registers, and some local memory.
- The first SIMD computer:
  - ILLIAC IV (1970s): 64 relatively powerful processors (mesh connection, see above).
- Contemporary commercial computer:
  - CM-2 (Connection Machine) by Thinking Machines Corporation: 65 536 very simple processors (connected as a hypercube).
- Array processors are highly specialized for numerical problems that can be expressed in matrix or vector format (see program on slide 8). Each PU computes one element of the result.



### MULTIPROCESSORS

Shared memory MIMD computers are called **multiprocessors**:



- Some multiprocessors have no shared memory which is central to the system and equally accessible to all processors. All the memory is distributed as local memory to the processors. **However**, each processor has access to the local memory of any other processor  $\Rightarrow$  a global physical address space is available. This memory organization is called *distributed shared memory*.



### Multiprocessors (cont'd)

- Communication between processors is through the shared memory. One processor can change the value in a location and the other processors can read the new value.
- From the programmers point of view communication is realised by *shared variables*; these are variables which can be accessed by each of the parallel activities (processes):
  - table  $t$  in slide 5;
  - matrixes  $a$ ,  $b$ , and  $c$  in slide 7;
- With many fast processors memory contention can seriously degrade performance  $\Rightarrow$  multiprocessor architectures don't support a high number of processors.



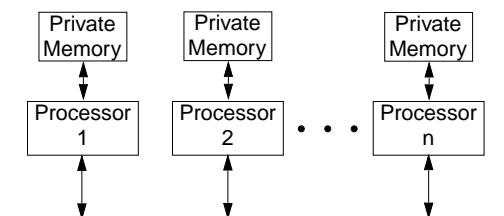
### Multiprocessors (cont'd)

- IBM System/370 (1970s): two IBM CPUs connected to shared memory.  
IBM System370/XA (1981): multiple CPUs can be connected to shared memory.  
IBM System/390 (1990s): similar features like S370/XA, with improved performance. Possibility to connect several multiprocessor systems together through fast fibre-optic connection.
- CRAY X-MP (mid 1980s): from one to four vector processors connected to shared memory (cycle time: 8.5 ns).  
CRAY Y-MP (1988): from one to 8 vector processors connected to shared memory; 3 times more powerful than CRAY X-MP (cycle time: 4 ns).  
C90 (early 1990s): further development of CRAY Y-MP; 16 vector processors.  
CRAY 3 (1993): maximum 16 vector processors (cycle time 2ns).
- Butterfly multiprocessor system, by BBN Advanced Computers (1985/87): maximum 256 Motorola 68020 processors, interconnected by a sophisticated dynamic switching network; distributed shared memory organization.  
BBN TC2000 (1990): improved version of the *Butterfly* using Motorola 88100 RISC processor.



## Multicomputers

MIMD computers with a distributed address space, so that each processor has its one private memory which is not visible to other processors, are called **multicomputers**:



## Multicomputers (cont'd)

- Communication between processors is only by *passing messages* over the interconnection network.
- From the programmers point of view this means that no shared variables are available (a variable can be accessed only by one single process). For communication between parallel activities (processes) the programmer uses *channels* and *send/receive operations* (see program in slide 10).
- There is no competition of the processors for the shared memory  $\Rightarrow$  the number of processors is not limited by memory contention.
- The speed of the interconnection network is an important parameter for the overall performance.



## Multicomputers (cont'd)

- **Intel iPSC/2** (1989): 128 CPUs of type 80386 interconnected by a 7-dimensional hypercube ( $2^7=128$ ).
- **Intel Paragon** (1991): over 2000 processors of type i860 (high performance RISC) interconnected by a two-dimensional mesh network.
- **KSR-1** by Kendal Square Research (1992): 1088 processors interconnected by a ring network.
- **nCUBE/2S** by nCUBE (1992): 8192 processors interconnected by a 10-dimensional hypercube.
- **Cray T3E MC512** (1995): 512 CPUs interconnected by a tree-dimensional mesh; each CPU is a DEC Alpha RISC.
- **Network of workstations:**  
A group of workstations connected through a Local Area Network (LAN), can be used together as a multicomputer for parallel computation. Performances usually will be lower than with specialized multicomputers, because of the communication speed over the LAN. However, this is a cheap solution.



## Vector Processors

- Vector processors include in their instruction set, beside scalar instructions, also instructions operating on vectors.
- Array processors (SIMD) computers (see slide 29) can operate on vectors by executing simultaneously the same instruction on pairs of vector elements; each instruction is executed by a separate processing element.
- Several computer architectures have implemented vector operations using the parallelism provided by pipelined functional units. Such architectures are called *vector processors*.

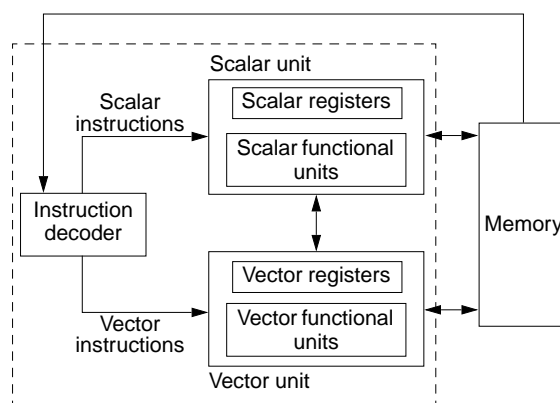


### Vector Processors (cont'd)

- Vector processors are not parallel processors; there are not several CPUs running in parallel. They are SISD processors which have implemented vector instructions executed on pipelined functional units.
- Vector computers usually have vector registers which can store each 64 up to 128 words.
- Vector instructions (see slide 40):
  - load vector from memory into vector register
  - store vector into memory
  - arithmetic and logic operations between vectors
  - operations between vectors and scalars
  - etc.
- From the programmers point of view this means that he is allowed to use operations on vectors in his programmes (see program in slide 8), and the compiler translates these instructions into vector instructions at machine level.



### Vector Processors (cont'd)



- Vector computers:
  - CDC Cyber 205
  - CRAY
  - IBM 3090 (an extension to the IBM System/370)
  - NEC SX
  - Fujitsu VP
  - HITACHI S8000



### The Vector Unit

- A vector unit typically consists of
  - pipelined functional units
  - vector registers
- Vector registers:
  - $n$  general purpose vector registers  $R_i$ ,  $0 \leq i \leq n-1$ ;
  - vector length register  $VL$ ; stores the length  $l$  ( $0 \leq l \leq s$ ), of the currently processed vector(s);  $s$  is the length of the vector registers  $R_i$ ;
  - mask register  $M$ ; stores a set of  $l$  bits, one for each element in a vector register, interpreted as boolean values; vector instructions can be executed in masked mode so that vector register elements corresponding to a false value in  $M$ , are ignored.



### Vector Instructions

#### LOAD-STORE instructions:

$R \leftarrow A(x1:x2:incr)$	load
$A(x1:x2:incr) \leftarrow R$	store
$R \leftarrow \text{MASKED}(A)$	masked load
$A \leftarrow \text{MASKED}(R)$	masked store
$R \leftarrow \text{INDIRECT}(A(X))$	indirect load
$A(X) \leftarrow \text{INDIRECT}(R)$	indirect store

#### Arithmetic - logic

$R \leftarrow R' \text{ b\_op } R''$
$R \leftarrow S \text{ b\_op } R'$
$R \leftarrow u\_op R'$
$M \leftarrow R \text{ rel\_op } R'$
$\text{WHERE}(M) R \leftarrow R' \text{ b\_op } R''$

#### chaining

$R2 \leftarrow R0 + R1$
$R3 \leftarrow R2 * R4$

execution of the vector multiplication has not to wait until the vector addition has terminated; as elements of the sum are generated by the addition pipeline they enter the multiplication pipeline; thus, addition and multiplication are performed (partially) in parallel.



### Vector Instructions (cont'd)

In a Pascal-like language with vector computation:

```
if T[1..50]>0 then
  T[1..50]:=T[1..50]+1;
```

A compiler for a vector computer generates something like:

```
R0 ← T(0:49:1)
VL ← 50
M ← R0 > 0
WHERE(M) R0 ← R0 + 1
```



### Multimedia Extensions to General Purpose Microprocessors

- Video and audio applications very often deal with large arrays of small data types (8 or 16 bits).
- Such applications exhibit a large potential of SIMD (vector) parallelism.



New generations of general purpose microprocessors have been equipped with special instructions to exploit this potential of parallelism.

- The specialised multimedia instructions perform vector computations on bytes, half-words, or words.



### Multimedia Extensions to General Purpose Microprocessors (cont'd)

Several vendors have extended the instruction set of their processors in order to improve performance with multimedia applications:

- MMX for Intel x86 family
- VIS for UltraSparc
- MDMX for MIPS
- MAX-2 for Hewlett-Packard PA-RISC

The Pentium line provides 57 MMX instructions. They treat data in a SIMD fashion (see textbook pg. 353).



### Multimedia Extensions to General Purpose Microprocessors (cont'd)

The basic idea: *subword execution*

- Use the entire width of a processor data path (32 or 64 bits), even when processing the small data types used in signal processing (8, 12, or 16 bits).



With word size 64 bits, the adders will be used to implement eight 8 bit additions in parallel

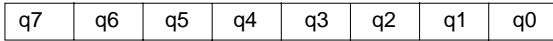
This is practically a kind of SIMD parallelism, at a reduced scale.



### Multimedia Extensions to General Purpose Microprocessors (cont'd)

Three packed data types are defined for parallel operations: packed byte, packed half word, packed word.

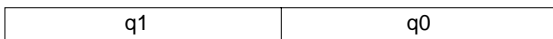
Packed byte



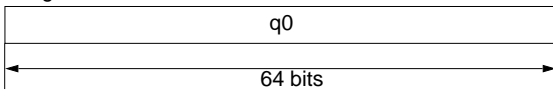
Packed half word



Packed word



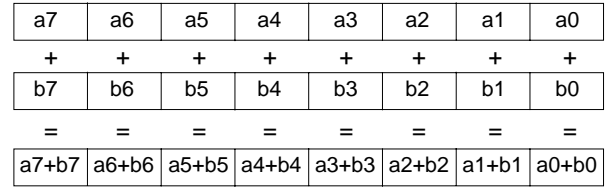
Long word



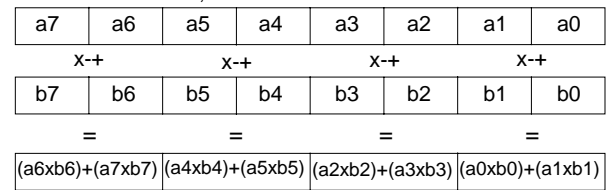
### Multimedia Extensions to General Purpose Microprocessors (cont'd)

- Examples of SIMD arithmetics with the MMX instruction set:

ADD R3 ← R1,R2



MPYADD R3 ← R1,R2



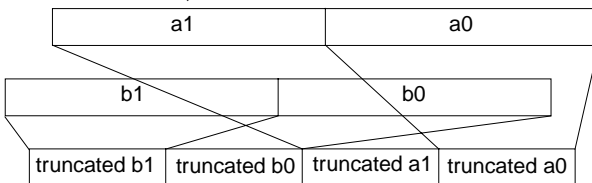
### Multimedia Extensions to General Purpose Microprocessors (cont'd)

How to get the data ready for computation?

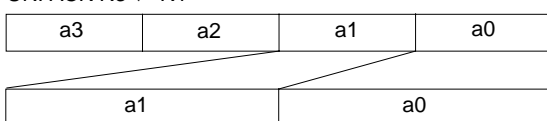
How to get the results back in the right format?

- Packing and Unpacking

PACK.W R3 ← R1,R2



UNPACK R3 ← R1



### Summary

- The growing need for high performance can not always be satisfied by computers running a single CPU.
- With Parallel computers, several CPUs are running in order to solve a given application.
- Parallel programs have to be available in order to use parallel computers.
- Computers can be classified based on the nature of the instruction flow executed and that of the data flow on which the instructions operate: SISD, SIMD, and MIMD architectures.
- The performance we effectively can get by using a parallel computer depends not only on the number of available processors but is limited by characteristics of the executed programs.
- The efficiency of using a parallel computer is influenced by features of the parallel program, like: degree of parallelism, intensity of inter-processor communication, etc.

### Summary (cont'd)

- A key component of a parallel architecture is the interconnection network.
- Array processors execute the same operation on a set of interconnected processing units. They are specialized for numerical problems expressed in matrix or vector formats.
- Multiprocessors are MIMD computers in which all CPUs have access to a common shared address space. The number of CPUs is limited.
- Multicomputers have a distributed address space. Communication between CPUs is only by message passing over the interconnection network. The number of interconnected CPUs can be high.
- Vector processors are SISD processors which include in their instruction set instructions operating on vectors. They are implemented using pipelined functional units.
- Multimedia applications exhibit a large potential of SIMD parallelism. The instruction set of modern general purpose microprocessors (Pentium, UltraSparc) has been extended to support SIMD-style parallelism with operations on short vectors.

