Advanced Computer Architecture

Instructor

Dr. Dinh-Duc Anh-Vu

http://www.cse.hcmut.edu.vn/~anhvu

Chapter 7

PARALLEL PROCESSING

Lecture 7 – Parallel Processing

- Introduction and motivation
- Architecture classification
- Performance of Parallel Architectures
- Interconnection Network

Computer as a Sequential Machine

- Most computer programming languages require the programmer to specify algorithms as sequences of instructions.
- Processors execute programs by executing machine instructions in a sequence and one at a time.
- Each instruction is executed in a sequence of operations (fetch instruction, fetch operands, perform operation, store results)

Performance Improvement

- Reduction of instruction execution time
  - Increased clock frequency by fast circuit technology
  - Simplify instructions (RISC)
- Parallelism within processor
  - Pipelining
  - Parallel execution of instructions (ILP)
    - Superscalar
    - VLIW architectures
- Parallel processing

Why Parallel Processing ?

- Sequential computers often are not able to meet performance needs in certain applications:
  - Fluid flow analysis and aerodynamics;
  - Simulation of large complex systems;
    - in physics, economy, biology, etc.
    - seismology, meteorology, atomic, nuclear, plasma physics
  - Computer aided design;
  - Multimedia.
- Applications in the above domains are characterized by a very large amount of numerical computation and/or a high quantity of input data.
- In order to deliver sufficient performance for such applications, we can have many processors in a single computer.

Parallel Computer

- **Parallel computers** refer to architectures in which many CPUs are running in parallel to implement a certain application.
- Such computers can be organized in very different ways, depending on several key features:
  - number and complexity of individual CPUs;
  - availability of common (shared) memory;
  - interconnection topology;
  - performance of interconnection network;
  - I/O devices;
  - etc.

Parallel Programs

- In order to solve a problem using a parallel computer, one must decompose the problem into small sub-problems, which can be solved in parallel.
  - The results of sub-problems may have to be combined to get the final result of the main problem.
- Because of data dependency among the sub-problems, it is not easy to decompose a complex problem to generate a large degree of parallelism.
- Because of data dependency, the processors may have to communicate among each other.
- The time taken for communication is usually very high when compared with the processing time.
- The communication scheme must therefore be very well planned in order to get a good parallel algorithm.

Parallel Program Example (1)

- Matrix addition:

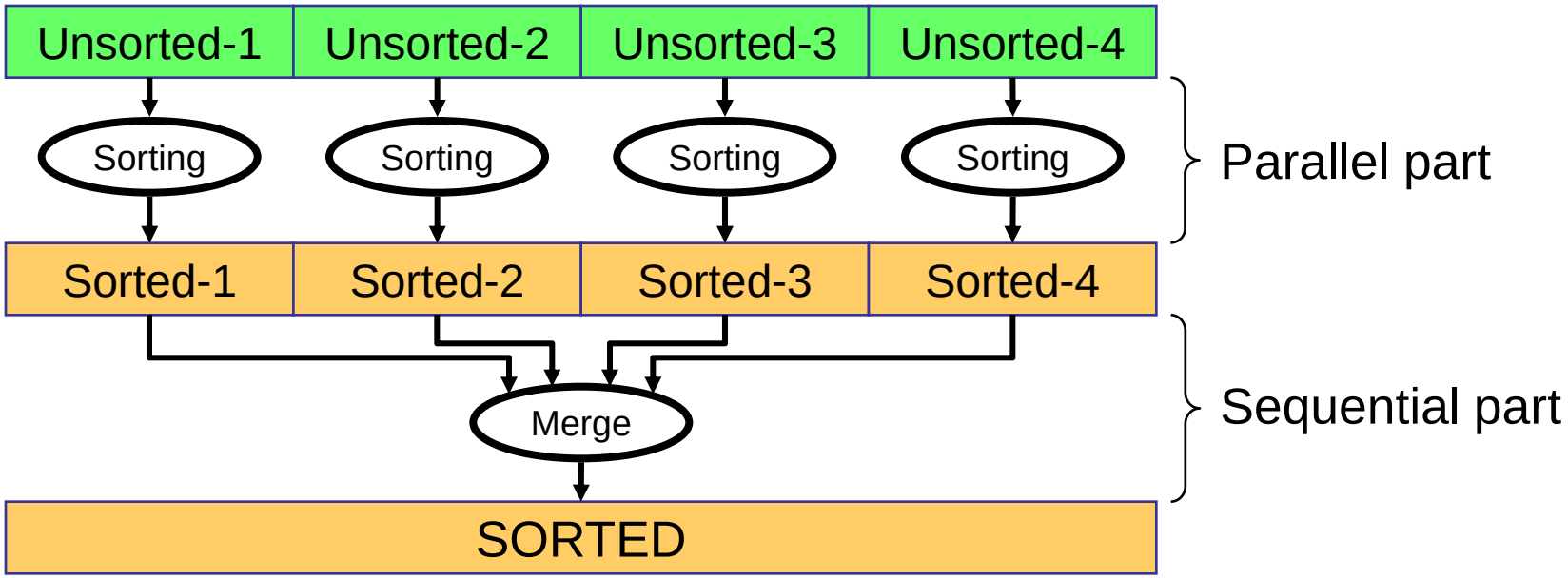
$$C = A + B = \begin{bmatrix} A_{11}+B_{11} & A_{12}+B_{12} & A_{13}+B_{13} & \cdots & A_{1M}+B_{1M} \\ A_{21}+B_{21} & A_{22}+B_{22} & A_{23}+B_{23} & \cdots & A_{2M}+B_{2M} \\ A_{31}+B_{31} & A_{32}+B_{32} & A_{33}+B_{33} & \cdots & A_{3M}+B_{3M} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ A_{N1}+B_{N1} & A_{N2}+B_{N2} & A_{N3}+B_{N3} & \cdots & A_{NM}+B_{NM} \end{bmatrix}$$

- Vector computation with vector of *m* elements:

```
for i:=1 to n do
    C[i,1:m]:=A[i,1:m] + B[i,1:m];
end for;
```

Parallel Program Example (2)

- Parallel sorting:




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

Sorting of 1000 integers

Lecture 7 – Parallel Processing

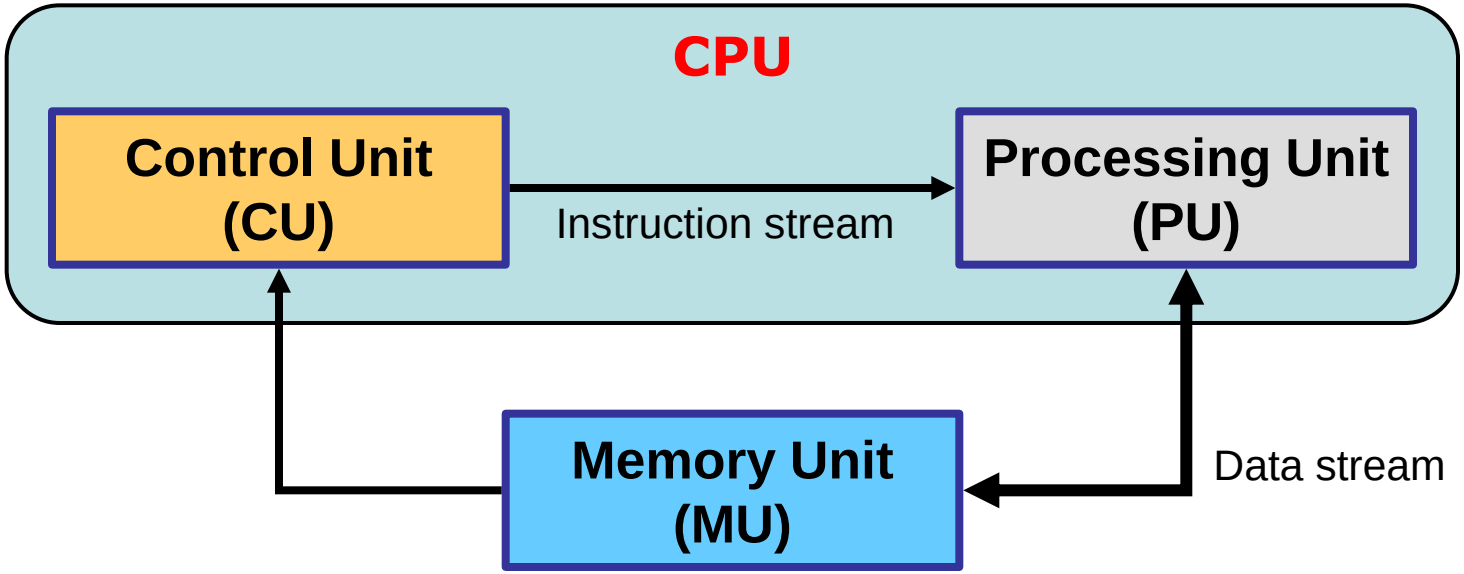
- Introduction and motivation
- Architecture classification
- Performance of Parallel Architectures
- Interconnection Network

Flynn's Classification of Architectures

- Flynn's classification (1966) is based on the nature of the instruction flow executed by the computer and that of the data flow on which the instructions operate.
- The multiplicity of instruction stream and data stream gives us four different classes:
  - Single instruction, single data stream – **SISD**
  - Single instruction, multiple data stream – **SIMD**
  - Multiple instruction, single data stream – **MISD**
  - Multiple instruction, multiple data stream – **MIMD**

Single Instruction, Single Data – SISD

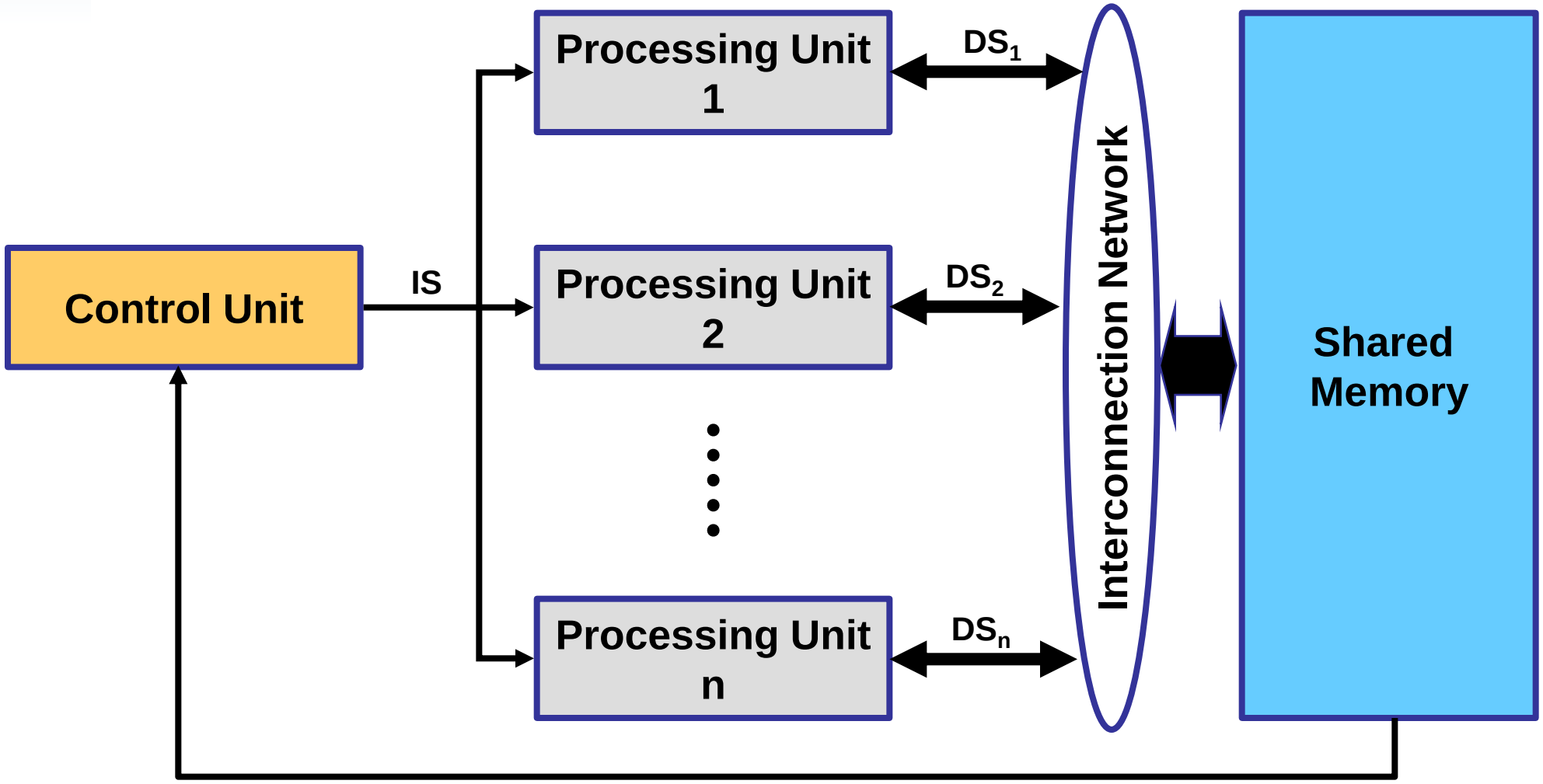
- Single processor
- Single instruction stream
- Data stored in single memory
- Uni-processor



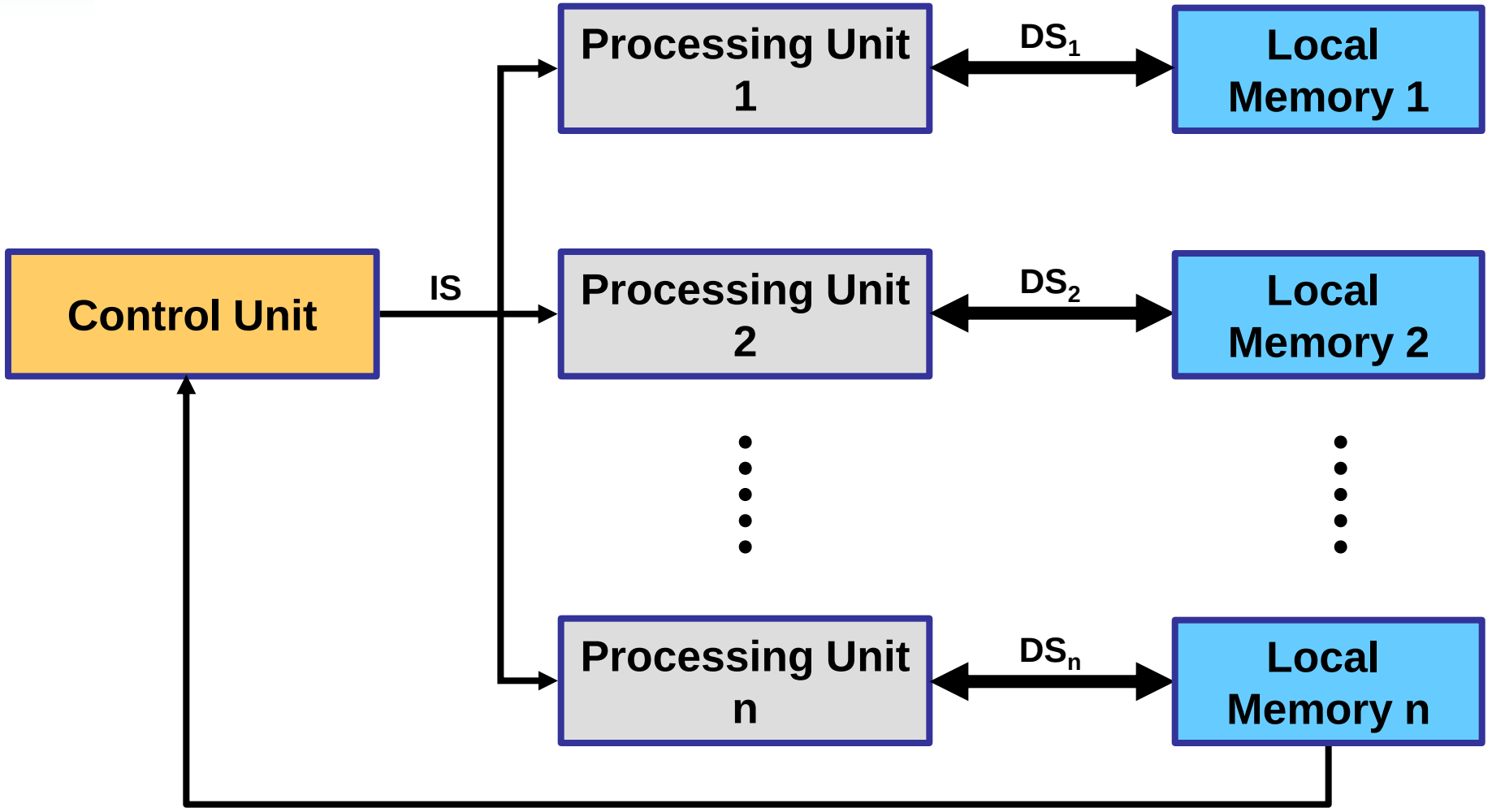
Single Instruction, Multiple Data – SIMD

- Single machine instruction stream
- Simultaneous execution on different set of data

SIMD with Shared Memory



SIMD with Distributed Memory



Single Instruction, Multiple Data – SIMD

- Single machine instruction stream
- Simultaneous execution on different set of data
- Large number of processing elements
- Lockstep synchronization among the process elements.
- The processing elements can
  - Have their respective private data memory; or
  - Share a common memory via a interconnection network.
- Array and vector processors are the most common SIMD machines

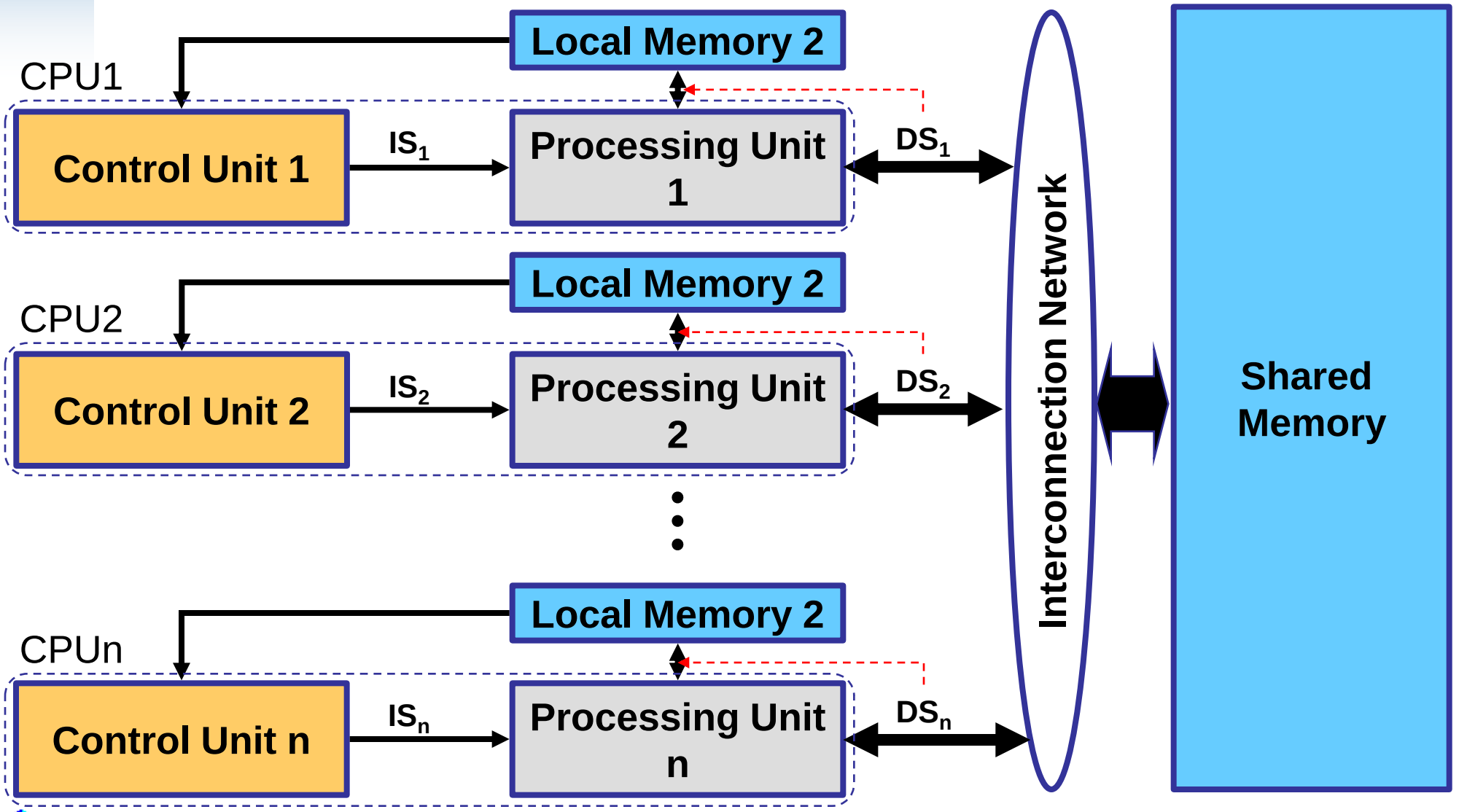
Multiple Instruction, Single Data – MISD

- Single sequence of data
- Transmitted to set of processors
- Each processor executes different instruction sequence
- **Never been implemented!**

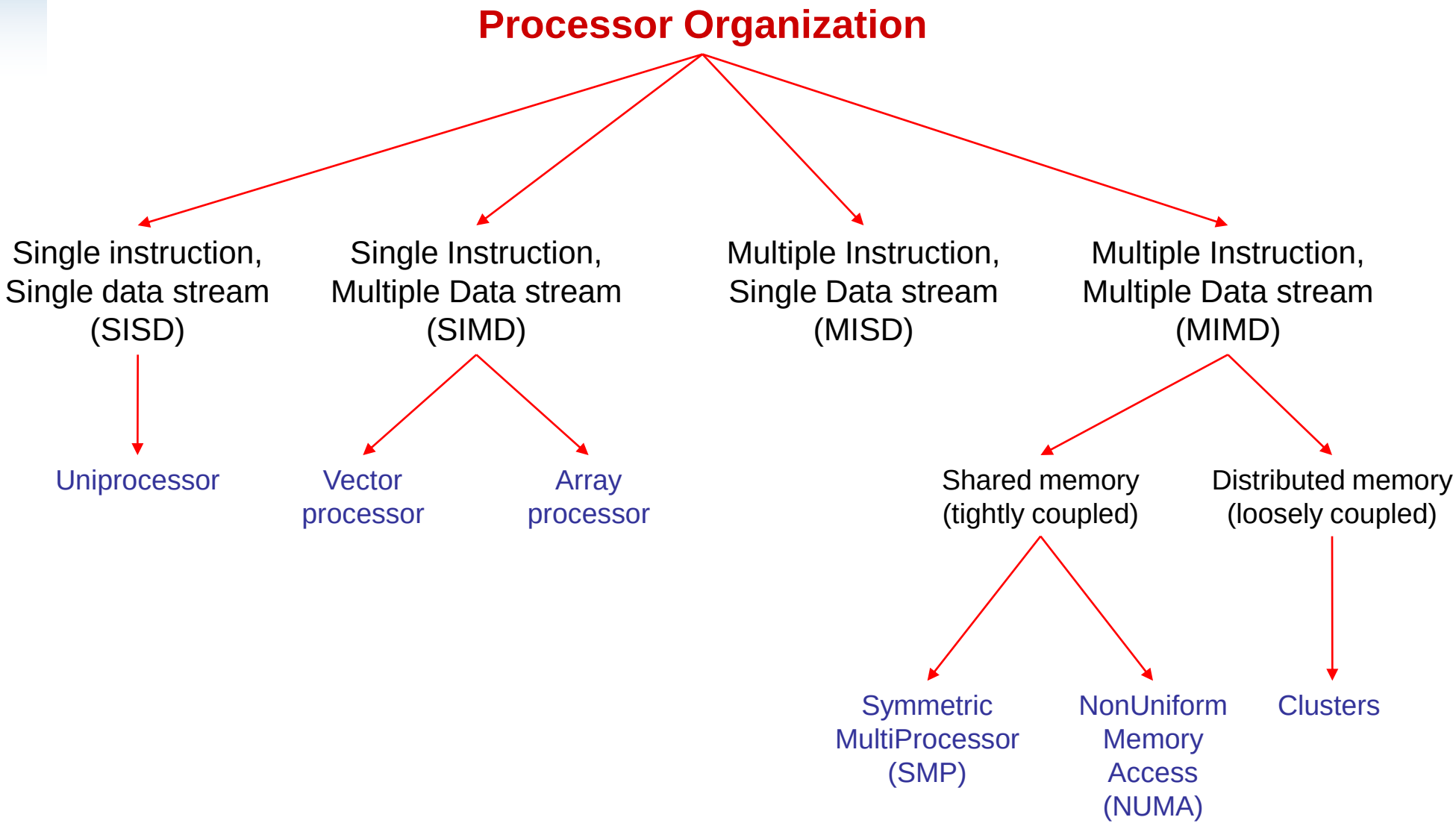
Multiple Instruction, Multiple Data – MIMD

- Set of general-purpose processors
- Simultaneously execute different instruction sequences
- Different sets of data
- The MIMD class can be further divided:
  - Shared memory (tightly coupled):
    - Processors share memory and communicate via that shared memory
    - Symmetric multiprocessor (SMP)
      - Share single memory or pool
      - Shared bus to access memory
      - Memory access time to given area of memory is approximately the same for each processor
    - Non-uniform memory access (NUMA)
      - Access times to different regions of memroy may differ
  - Distributed memory (loosely coupled) = Clusters
    - Collection of independent uniprocessors or SMPs
    - Interconnected to form a cluster
    - Communication via fixed path or network connections

MIMD with Shared Memory



Taxonomy of Parallel Processor Architecture



Lecture 7 – Parallel Processing

- Introduction and motivation
- Architecture classification
- Performance of Parallel Architectures
- Interconnection Network

Performance of Parallel Architectures

Important questions:

- How fast does a parallel computer run at its maximal potential?
- How fast execution can we expect from a parallel computer for a given 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 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*:

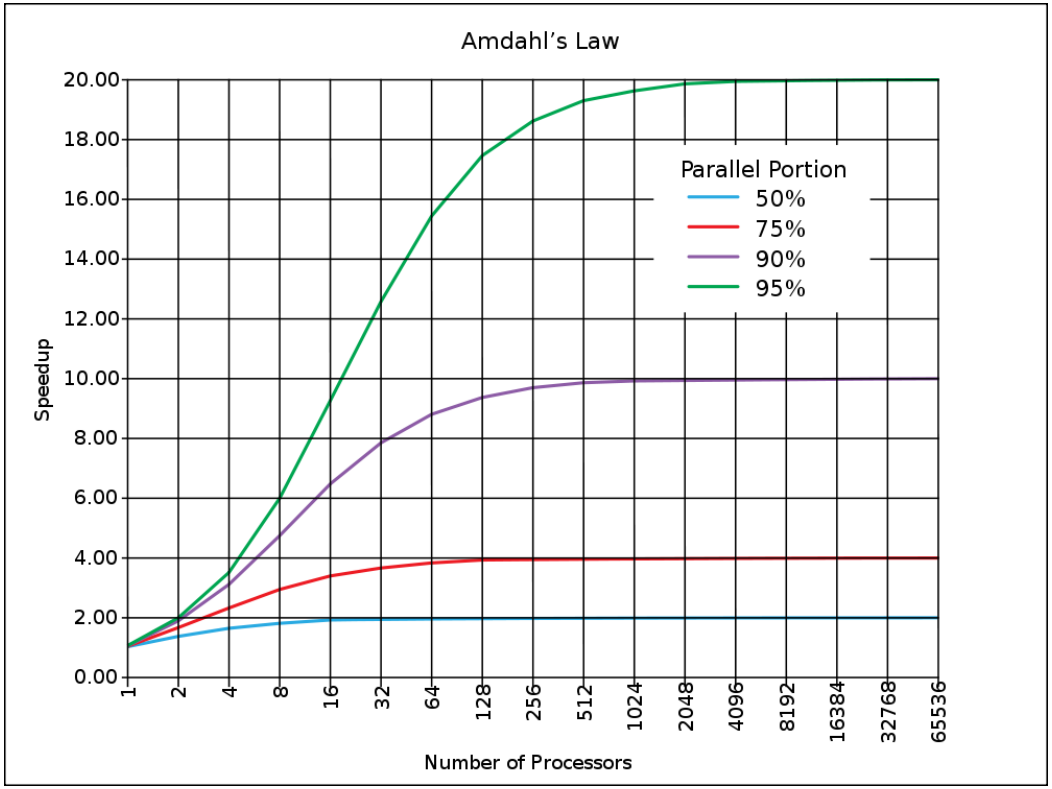
  S = *P*; which means *E* = 1.

Practically the ideal efficiency of 1 cannot be achieved!

Amdahl's Law

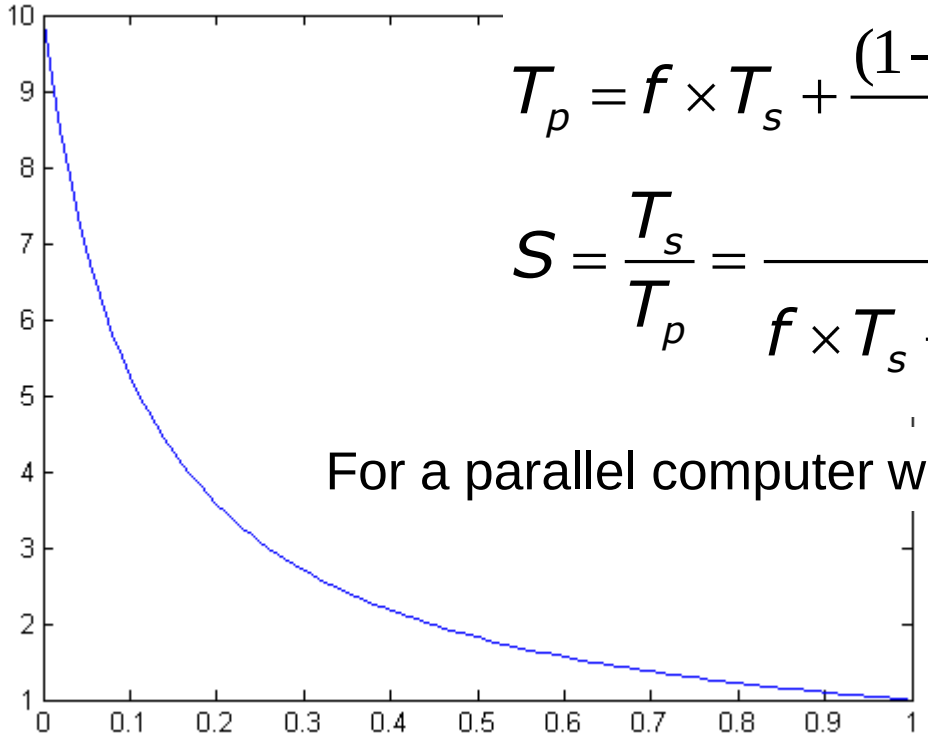
- Used to find the maximum expected improvement to an overall system when only part of the system is improved
- Often used in parallel computing to predict the theoretical maximum speedup using multiple processors
- The speedup of a program using multiple processors in parallel computing is limited by the time needed for the sequential fraction of the program.

- For example, if 95% of the program can be parallelized, the theoretical maximum speedup using parallel computing would be 1/(1 – 95%) = 20× as shown in the diagram, no matter how many processors are used.



Amdahl's Law

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



$$T_p = f \times T_s + \frac{(1-f) \times T_s}{P}$$

$$S = \frac{T_s}{T_p} = \frac{T_s}{f \times T_s + \frac{(1-f) \times T_s}{P}} = \frac{1}{f + \frac{(1-f)}{P}}$$

For a parallel computer with 10 processing elements

Amdahl's Law (Cont'd)

- Amdahl's law says that even a little ratio of sequential computation imposes a limit to speedup.
  - A higher speedup than 1/*f* can not be achieved, regardless of the number of processors, since

$$S = \frac{1}{f + \frac{1-f}{P}}$$

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

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

Other Aspects that Limit the Speedup

- Beside the intrinsic sequentiality of 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.
  - The value of *f* is very small and can be ignored;
  - Suited for massively parallel systems; and
  - The other limiting factors, like the cost of communications, become critical.

Efficiency and Communication Cost

- Consider a highly parallel computation, so that *f* can be neglected.
- We define f_c the fractional communication overhead of a processor:
  - T_{calc} the time that a processor executes computations;
  - T_{comm} the time that a processor is idle because of communication;

$$f_c = \frac{T_{comm}}{T_{calc}} \qquad T_p = \frac{T_S}{P} \times (1 + f_c)$$

$$S = \frac{T_S}{T_p} \qquad E = \frac{1}{1 + f_c} \approx 1 - f_c \qquad (\text{if } f_c \text{ is small})$$

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

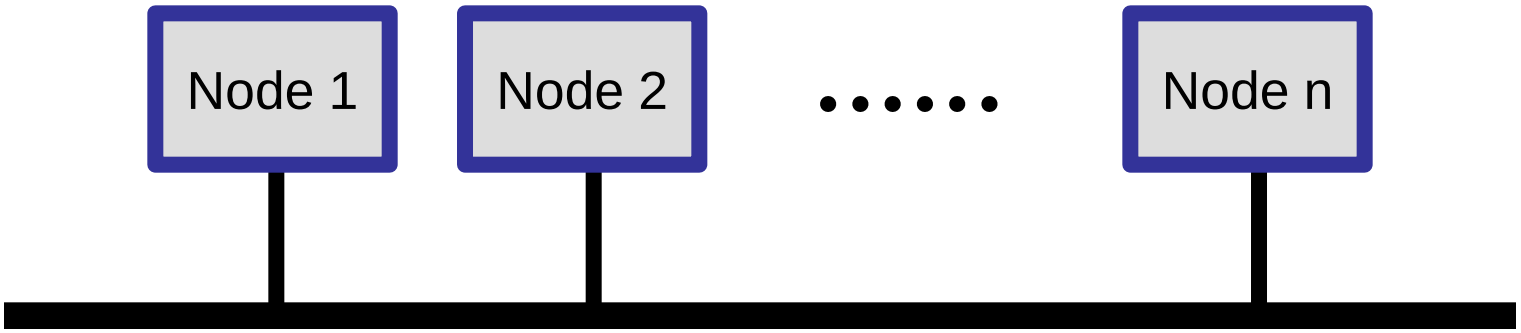
Lecture 7 – Parallel Processing

- Introduction and motivation
- Architecture classification
- Performance of Parallel Architectures
- Interconnection Network

Interconnection Network

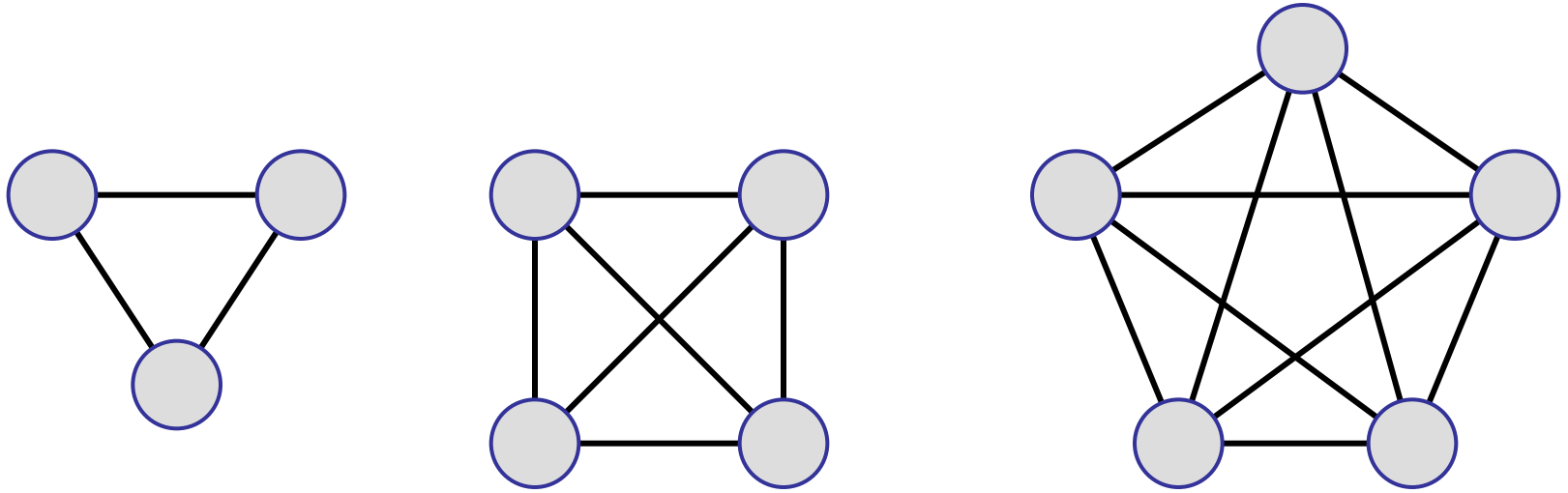
- The interconnection network (IN) is a key component of the architecture. It has a decisive influence on:
  - the overall performance; and
  - total cost of the architecture.
- The traffic in the IN consists of data transfer and transfer of commands and requests (control information).
- The key parameters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

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 (around 16 - 20).

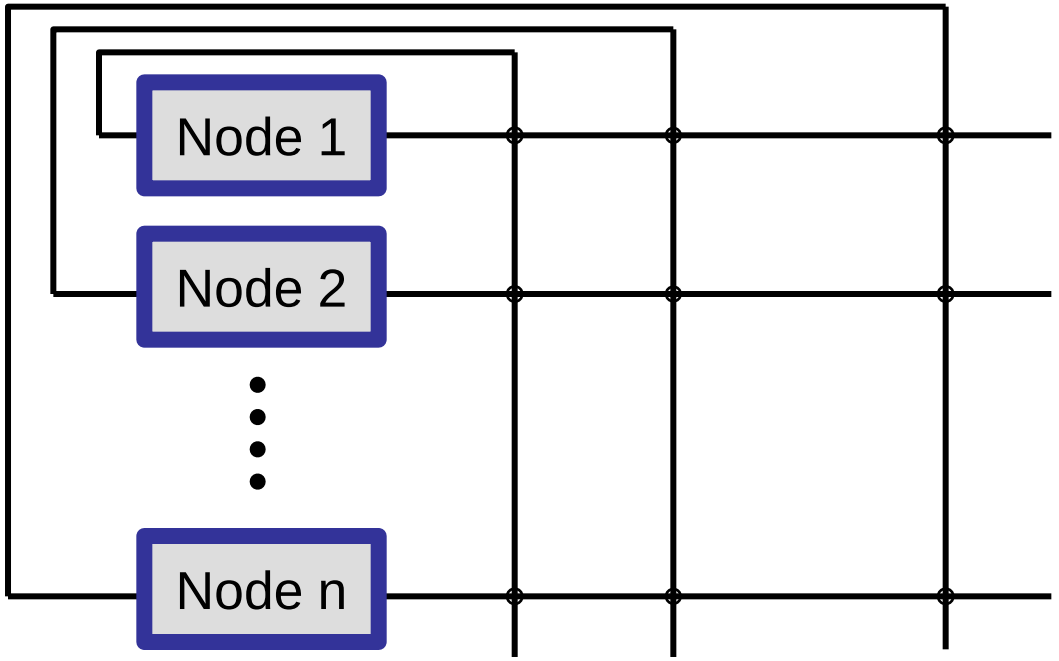
Completely Connected Network



N(N-1)/2 wires

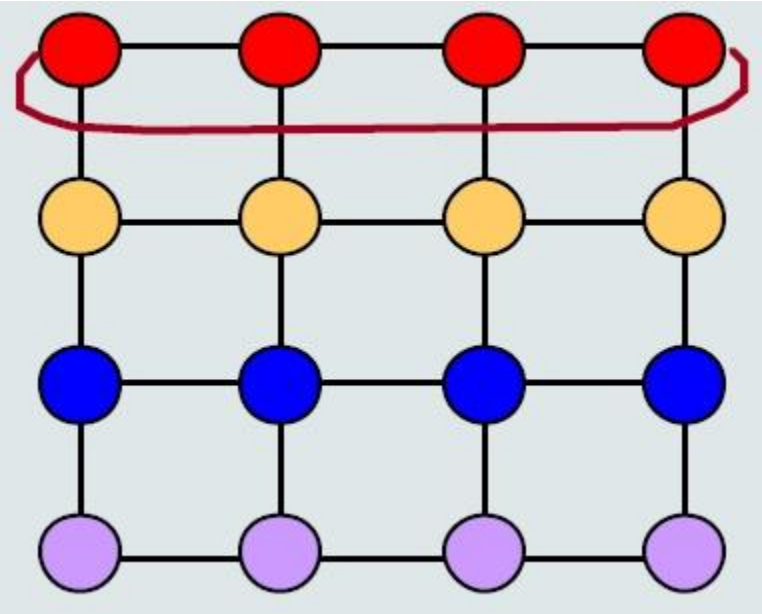
- Each node is connected to every other one.
- Communications can be performed in parallel between any pair of nodes.
- Both performance and cost are high.
- Cost increases rapidly with number of nodes.

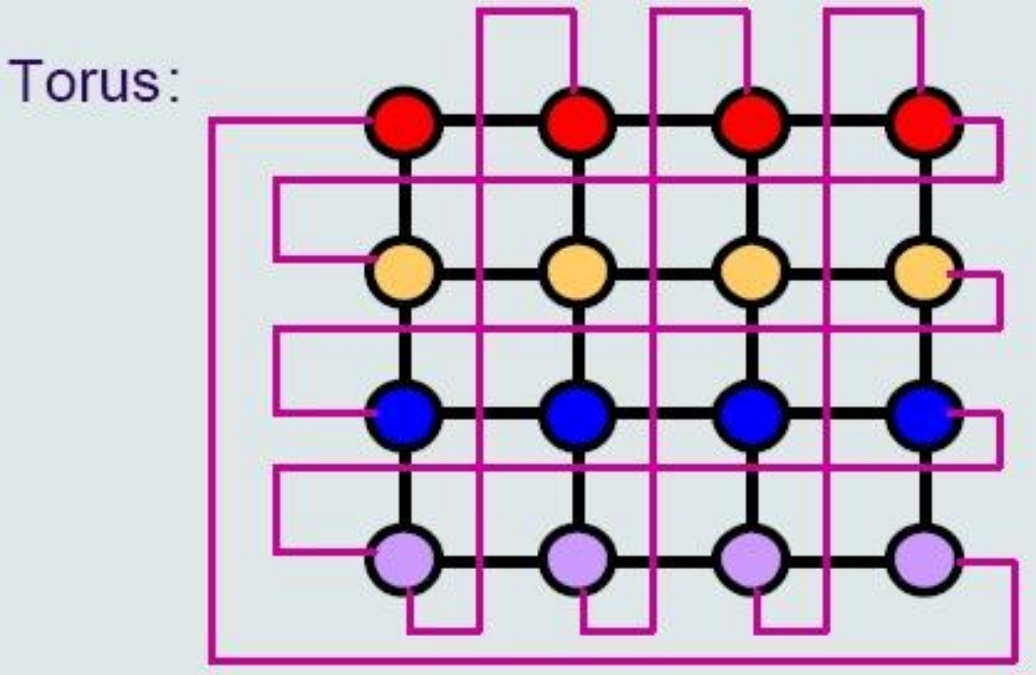
Crossbar Network



- A dynamic network: the interconnection topology can be modified by configuratiing the switches.
- It 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 (even though one node can receive or send only one data at a time).

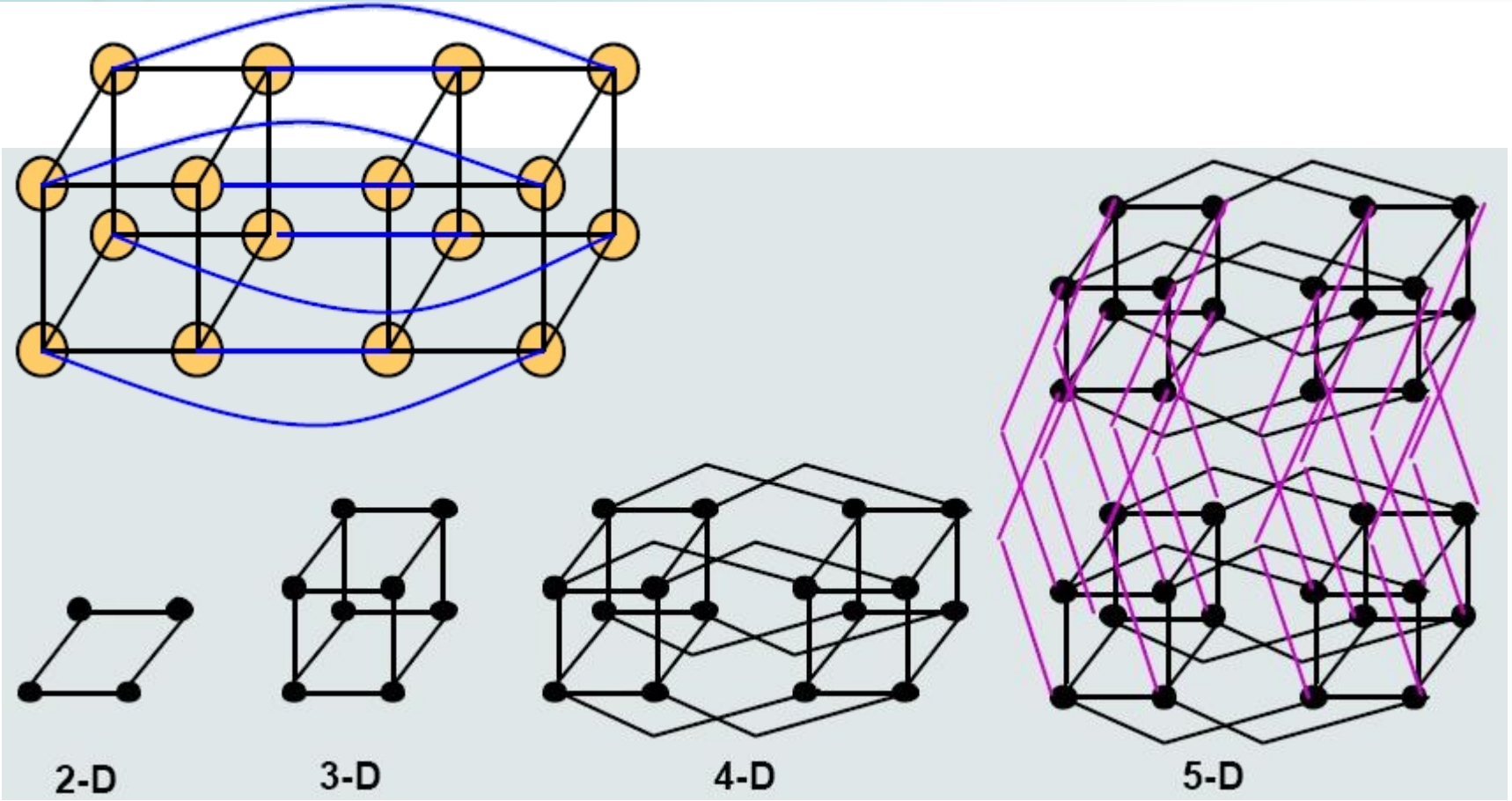
Mesh Network





- Cheaper than completely connected networks, while giving relatively good performance.
- In order to transmit data between two nodes, routing through intermediate nodes is needed (maximum 2*(n-1) intermediates for an n*n mesh).
- It is possible to provide wraparound connections:
  - Torus.
- Three dimensional meshes have been also implemented.

Hypercube Network



- 2^n nodes are arranged in an *n*-dimensional cube. Each node is connected to *n* neighbors.
- In order to transmit data between two nodes, routing through intermediate nodes is needed (maximum *n* intermediates).

Summary

- The growing need for high performance can not always be satisfied by computers with a single CPU.
- With parallel computers, several CPUs are running concurrently in order to solve a given application.
- Parallel programs have to be available in order to make use of the parallel computers.
- Computers can be classified based on the nature of the instruction flow and that of the data flow on which the instructions operate.
- A key component of a parallel architecture is also the interconnection network.
- The performance we can get with a parallel computer depends not only on the number of available processors but is limited by characteristics of the executed programs.