



Introduction: Part I



1. **Problem size exceeds desktop capabilities.**



1. **Problem size exceeds desktop capabilities.**
2. **Problem is inherently parallel (e.g. Monte-Carlo simulations).**



1. **Problem size exceeds desktop capabilities.**
2. **Problem is inherently parallel (e.g. Monte-Carlo simulations).**
3. **Modern architectures require parallel programming skills to be optimally exploited.**



Flynn's Taxonomy of Parallel Architectures

[FLYNN '72]

The basic definition of a parallel computer is very vague in order to cover a large class of systems. Important details that are not considered by the definition are:



The basic definition of a parallel computer is very vague in order to cover a large class of systems. Important details that are not considered by the definition are:

- How many processing elements?



The basic definition of a parallel computer is very vague in order to cover a large class of systems. Important details that are not considered by the definition are:

- How many processing elements?
- How complex are they?



The basic definition of a parallel computer is very vague in order to cover a large class of systems. Important details that are not considered by the definition are:

- How many processing elements?
- How complex are they?
- How are they connected?



The basic definition of a parallel computer is very vague in order to cover a large class of systems. Important details that are not considered by the definition are:

- How many processing elements?
- How complex are they?
- How are they connected?
- How is their cooperation coordinated?

The basic definition of a parallel computer is very vague in order to cover a large class of systems. Important details that are not considered by the definition are:

- How many processing elements?
- How complex are they?
- How are they connected?
- How is their cooperation coordinated?
- What kind of problems can be solved?

The basic classification allowing answers to most of these questions is known as **Flynn's taxonomy**. It distinguishes four categories of parallel computers.



Flynn's Taxonomy of Parallel Architectures

[FLYNN '72]

The **four categories** allowing basic answers to the questions on global process control, as well as the resulting data and control flow in the machine are



The **four categories** allowing basic answers to the questions on global process control, as well as the resulting data and control flow in the machine are

1. Single-Instruction, Single-Data (SISD)



The **four categories** allowing basic answers to the questions on global process control, as well as the resulting data and control flow in the machine are

1. Single-Instruction, Single-Data (SISD)
2. Multiple-Instruction, Single-Data (MISD)



The **four categories** allowing basic answers to the questions on global process control, as well as the resulting data and control flow in the machine are

1. Single-Instruction, Single-Data (SISD)
2. Multiple-Instruction, Single-Data (MISD)
3. Single-Instruction, Multiple-Data (SIMD)



The **four categories** allowing basic answers to the questions on global process control, as well as the resulting data and control flow in the machine are

1. Single-Instruction, Single-Data (SISD)
2. Multiple-Instruction, Single-Data (MISD)
3. Single-Instruction, Multiple-Data (SIMD)
4. Multiple-Instruction, Multiple-Data (MIMD)



Flynn's Taxonomy of Parallel Architectures

Single-Instruction, Single-Data (SISD)

The SISD model is characterized by

- a single processing element,



Flynn's Taxonomy of Parallel Architectures

Single-Instruction, Single-Data (SISD)

The SISD model is characterized by

- a single processing element,
- executing a single instruction,
- on a single piece of data,
- in each step of the execution.



The SISD model is characterized by

- a single processing element,
- executing a single instruction,
- on a single piece of data,
- in each step of the execution.

It is thus in fact the standard sequential computer implementing, e.g., the *von Neumann* model.



The SISD model is characterized by

- a single processing element,
- executing a single instruction,
- on a single piece of data,
- in each step of the execution.

It is thus in fact the standard sequential computer implementing, e.g., the *von Neumann* model.

Examples

- desktop computers until Intel[®] Pentium[®] 4 era,
- early netBooks on Intel[®] Atom[™] basis,
- pocket calculators,
- abacus,
- embedded circuits

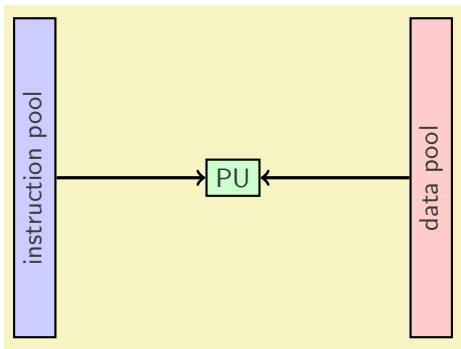


Figure: Single-Instruction, Single-Data (SISD) machine model



Flynn's Taxonomy of Parallel Architectures

Multiple-Instruction, Single-Data (MISD)

In contrast to the SISD model in the MISD architecture we have

- **multiple processing elements,**



In contrast to the SISD model in the MISD architecture we have

- **multiple processing elements,**
- **executing a separate instruction each,**
- **all on the same single piece of data,**
- **in each step of the execution.**



In contrast to the SISD model in the MISD architecture we have

- **multiple processing elements,**
- **executing a separate instruction each,**
- **all on the same single piece of data,**
- **in each step of the execution.**

The MISD model is usually not considered very useful in practice.

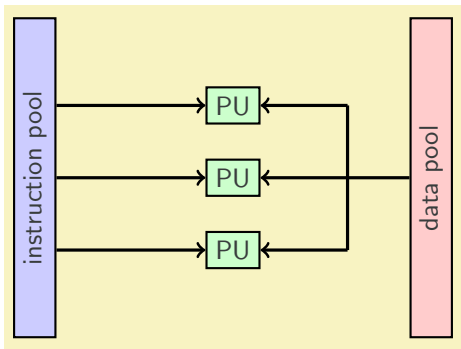


Figure: Multiple-Instruction, Single-Data (MISD) machine model



Here the characterization is

- **multiple processing elements,**
- **execute the same instruction,**
- **on a multiple pieces of data,**
- **in each step of the execution.**

This model is thus the ideal model for all kinds of vector operations

$$c = a + \alpha b.$$

Examples

- Graphics Processing Units,
- Vector Computers,
- SSE (Streaming SIMD Extension) registers of modern CPUs.



The attractiveness of the SIMD model for vector operations, i.e., linear algebra operations, comes at a cost.

Consider the simple conditional expression

```
if (b==0) c=a; else c=a/b;
```

The SIMD model requires the execution of both cases sequentially. First all processes for which the condition is true execute their assignment, then the other do the second assignment. Therefore, conditionals need to be avoided on SIMD architectures to guarantee maximum performance.

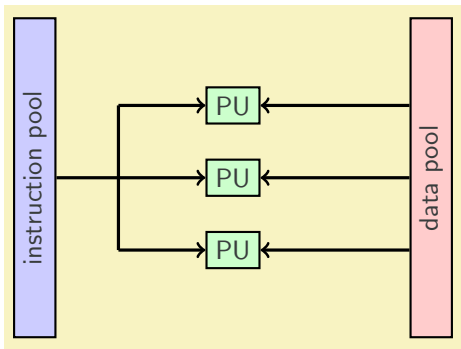


Figure: Single-Instruction, Multiple-Data (SIMD) machine model



MIMD allows

- **multiple processing elements,**
- **to execute a different instruction,**
- **on a separate piece of data,**
- **at each instance of time.**



MIMD allows

- **multiple processing elements,**
- **to execute a different instruction,**
- **on a separate piece of data,**
- **at each instance of time.**

Examples

- multicore and multi-processor desktop PCs,
- cluster systems.



Flynn's Taxonomy of Parallel Architectures

Multiple-Instruction, Multiple-Data (MIMD): Hardware classes

MIMD computer systems can be further divided into three class regarding their memory configuration:

distributed memory

Every processing element has a certain exclusive portion of the entire memory available in the system. Data needs to be exchanged via an interconnection network.

shared memory

All processing units in the system can access all data in the main memory.

hybrid

Certain groups of processing elements share a part of the entire data and instruction storage.



Single Program, Multiple Data (SPMD)

SPMD is a programming model for MIMD systems. “In SPMD multiple autonomous processors simultaneously execute the same program at independent points.”¹ This contrasts to the SIMD model where the execution points are not independent.

This is opposed to

Multiple Program, Multiple Data (MPMD)

A different programming model for MIMD systems, where multiple autonomous processing units execute different programs at the same time. Typically Master/Worker like management methods of parallel programs are associated with this class.

¹Wikipedia: <http://en.wikipedia.org/wiki/SPMD>

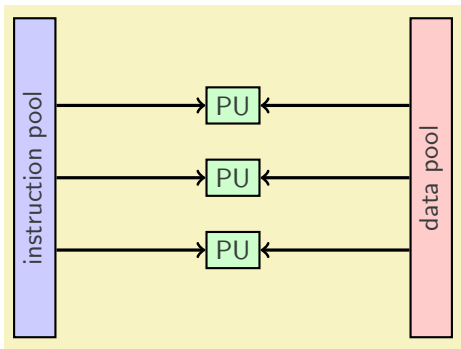


Figure: Multiple-Instruction, Multiple-Data (MIMD) machine model

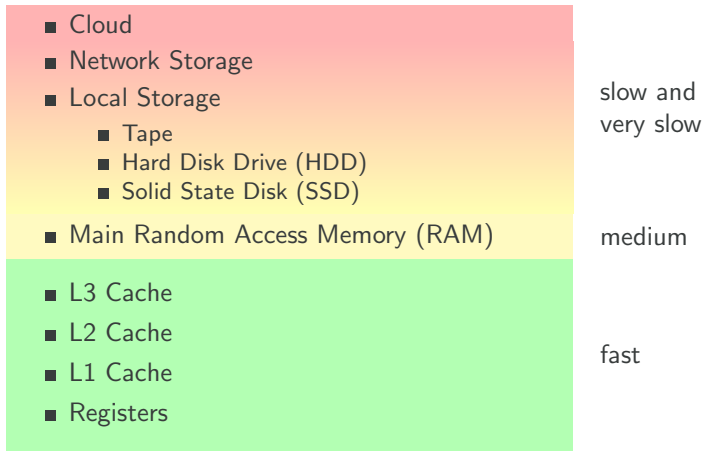


Figure: Basic memory hierarchy on a single processor system.

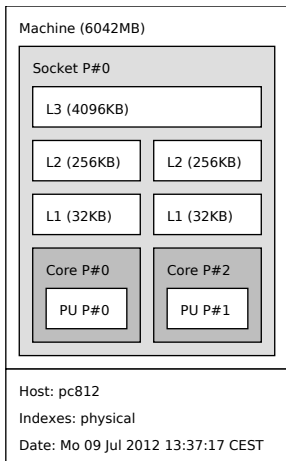


Figure: A sample dual core Xeon[®] setup

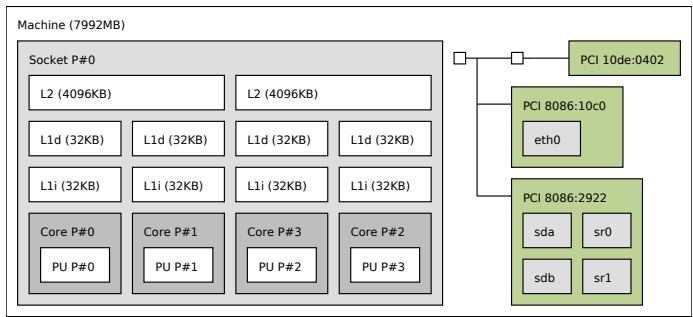


Figure: A sample Core™2 Quad setup

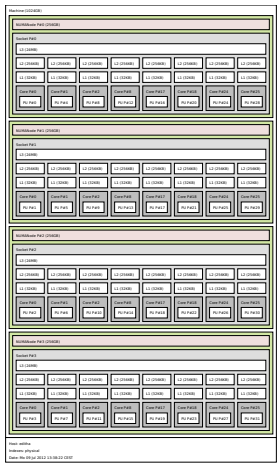


Figure: A four processor octa-core Xeon[®] system

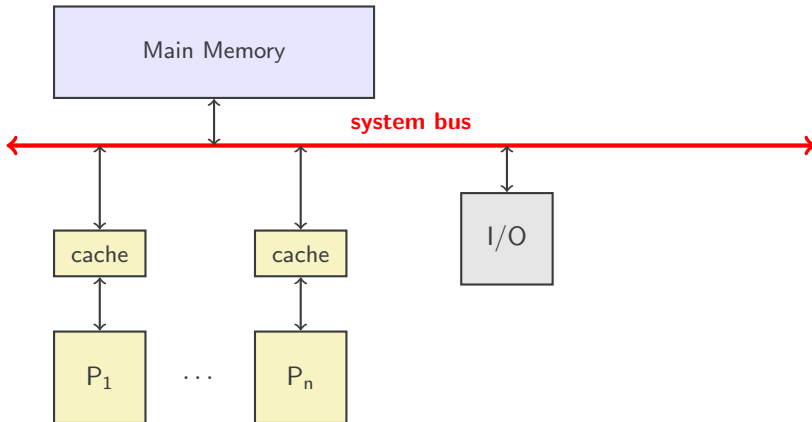


Figure: Schematic of a general parallel system

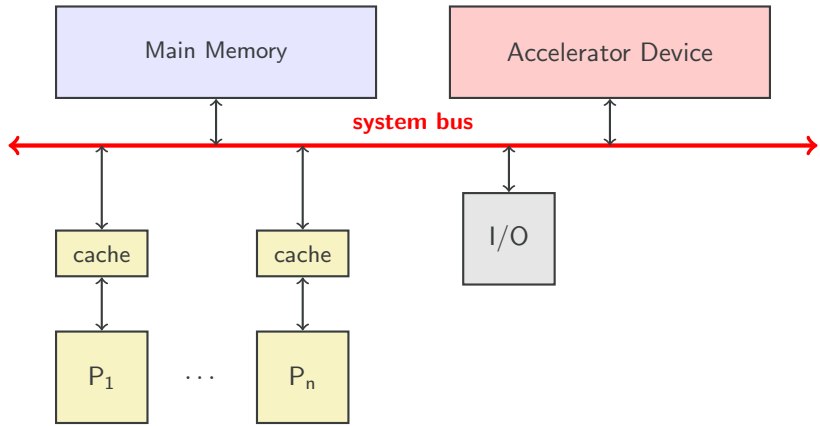


Figure: Schematic of a general parallel system

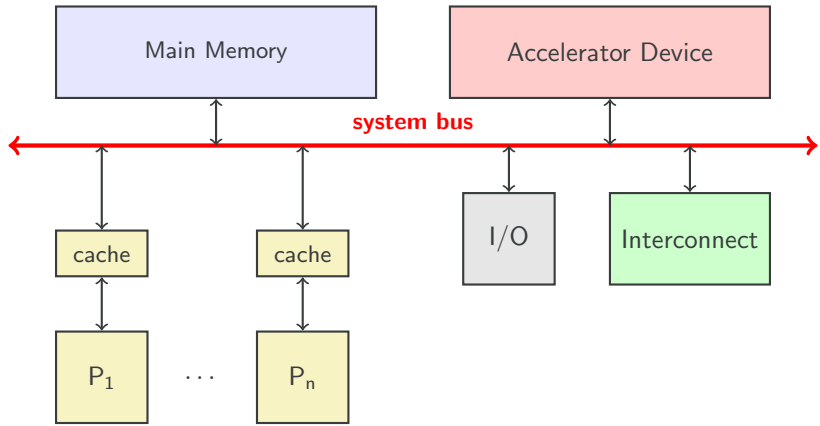


Figure: Schematic of a general parallel system

The **Interconnect** in the last figure stands for any kind of Communication grid.
This can be implemented either as

- **local hardware interconnect,**

or in the form of

- **network interconnect.**

In classical supercomputers the first was mainly used, whereas in today's cluster based systems often the network solution is used in the one form or the other.

MyriNet

- shipping since 2005
- transfer rates up to 10Gbit/s
- lost significance (2005 28.2% TOP500 down to 0.8% in 2011)

Infiniband

- transfer rates up to 300Gbit/s
- most relevant implementation driven by OpenFabrics Alliance², \rightsquigarrow usable on Linux, BSD, Windows systems.
- features remote direct memory access (RDMA) \rightsquigarrow reduced CPU overhead
- can also be used for TCP/IP communication

Omni-Path



- transfer rates up to 100Gbit/s
- introduced by Intel[®] in 2015/2016
- rising significance
- Intel[®]'s approach to address cluster of $> 10\,000$ nodes

²<http://www.openfabrics.org/>



1. **Linear array:**

nodes aligned on a string each being connected to at most two neighbors.



1. **Linear array:**

nodes aligned on a string each being connected to at most two neighbors.

2. **Ring:**

nodes are aligned in a ring each being connected to exactly two neighbors



1. **Linear array:**

nodes aligned on a string each being connected to at most two neighbors.

2. **Ring:**

nodes are aligned in a ring each being connected to exactly two neighbors

3. **Complete graph:**

every node is connected to all other nodes



1. **Linear array:**

nodes aligned on a string each being connected to at most two neighbors.

2. **Ring:**

nodes are aligned in a ring each being connected to exactly two neighbors

3. **Complete graph:**

every node is connected to all other nodes

4. **Mesh and Torus:**

Every node is connected to a number of neighbours (2-4 in 2d mesh, 4 in 2d torus).



1. **Linear array:**

nodes aligned on a string each being connected to at most two neighbors.

2. **Ring:**

nodes are aligned in a ring each being connected to exactly two neighbors

3. **Complete graph:**

every node is connected to all other nodes

4. **Mesh and Torus:**

Every node is connected to a number of neighbours (2-4 in 2d mesh, 4 in 2d torus).

5. **k-dimensional cube / hypercube:**

Recursive construction of a well connected network of 2^k nodes each connected to k neighbors. Line for two, square for four, cube for eight.



1. **Linear array:**

nodes aligned on a string each being connected to at most two neighbors.

2. **Ring:**

nodes are aligned in a ring each being connected to exactly two neighbors

3. **Complete graph:**

every node is connected to all other nodes

4. **Mesh and Torus:**

Every node is connected to a number of neighbours (2-4 in 2d mesh, 4 in 2d torus).

5. **k-dimensional cube / hypercube:**

Recursive construction of a well connected network of 2^k nodes each connected to k neighbors. Line for two, square for four, cube for eight.

6. **Tree:**

Nodes are arranged in groups, groups of groups and so forth until only one large group is left, which represents the root of the tree.