CM12002
Computer Systems Architectures

Parallel Architectures

Fabio Nemetz

Parallelism

Parallelism:

Carrying out multiple operations simultaneously.

Modern computers have several semi-autonomous subsystems:

- I/O control, direct memory access, graphics.
- Here, we're mainly concerned with parallelism in the main processor.

Parallel Architectures

The basic von Neumann machine is a *uniprocessor* architecture. Computers with multiple ALUs are *multiprocessors.*

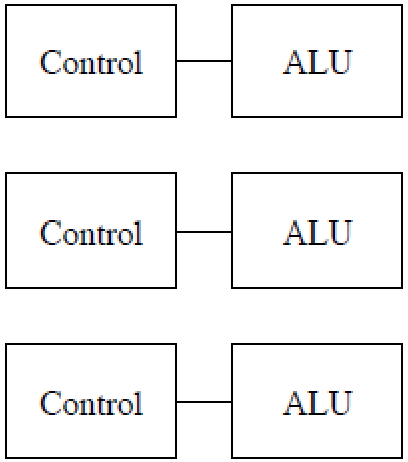
Multiprocessing is (potentially) fast, but harder to understand, control and predict: there is no single, dominant *parallel architecture*, but a set of alternatives, depending on whether each ALU has

- its own control unit (**task level**), and/or
- its own data storage (**data level**).

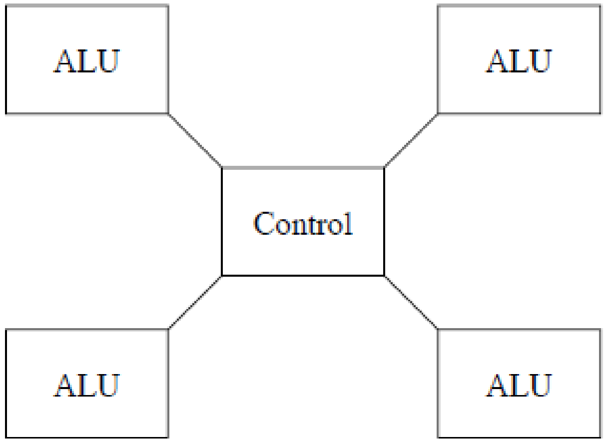
Task level parallelism

There are two possible *control architectures* for multiprocessors:

Multiple instruction streams
Each ALU is controlled by a separate control unit.



Single instruction stream
There is a single control unit issuing the same instruction to each ALU.



Data level parallelism

Independently of the control architecture the ALUs may either

- All operate on the same data stream, or
- Each operate on a different data stream
  (or a *vector*, or *array* of data.)

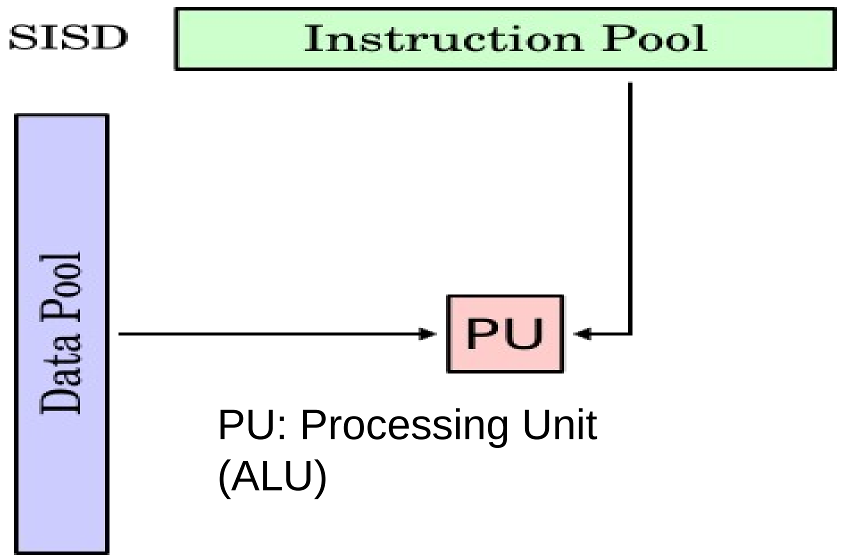
Flynn’s Taxonomy* (Task level x Data level)

	Single Data Stream	Multiple Data Streams
Single Instruction	SISD	SIMD
Multiple Instruction	MISD	MIMD

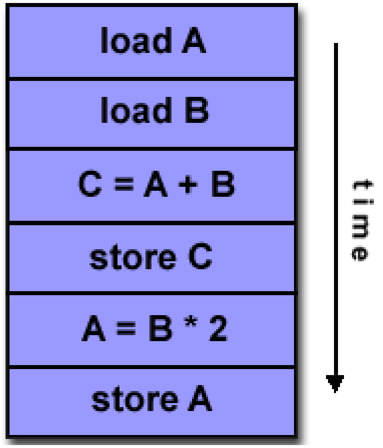
*Taxonomy: division into ordered groups or categories

Flynn, Michael J. (September 1972). "Some Computer Organizations and Their Effectiveness". *IEEE Trans. Comput.* **C–21** (9): 948–960

SISD: Single Instruction Single Data

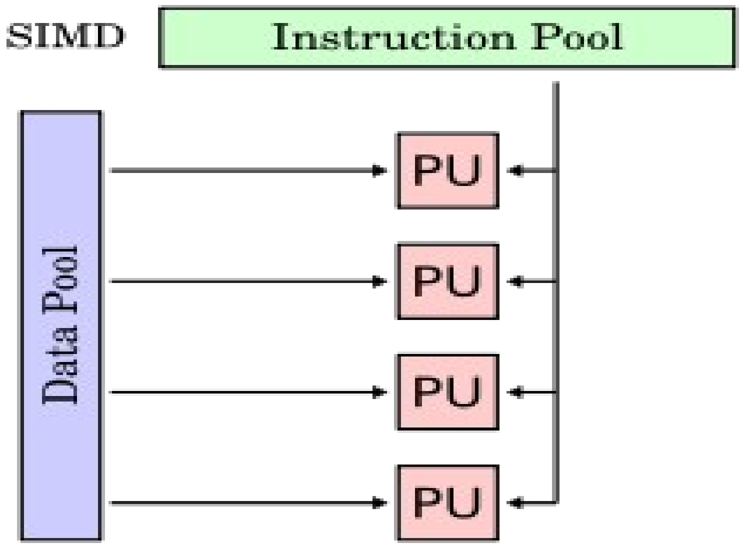


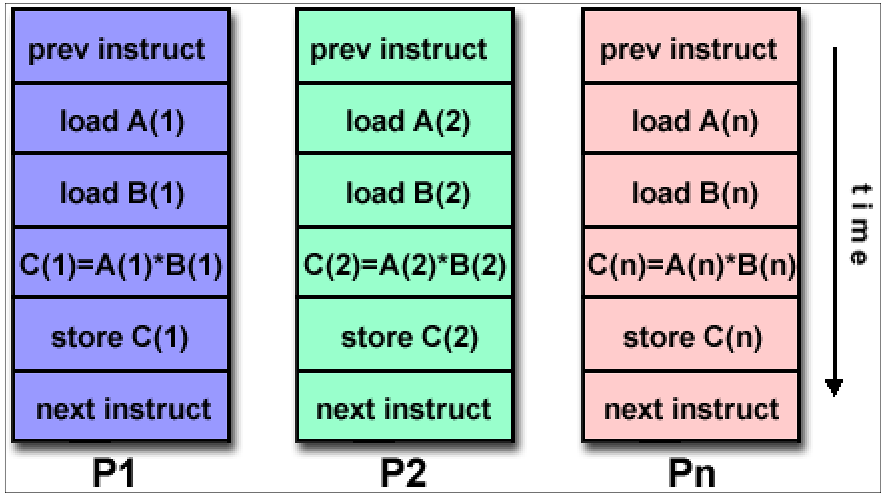
PU: Processing Unit
(ALU)



This is a uniprocessor architecture, such as von Neumann or Harvard architecture.

SIMD: Single Instruction Multiple Data





SIMD: Single Instruction Multiple Data

Involves the application of a basic operation to a large dataset (vectors and matrices)

E.g.: Intel introduced MMX extensions to the x86 architecture in 1997; also modern graphics cards (GPUs)

Examples:

- Supercomputer modeling of physical systems, e.g. weather.
- Signal processing (sound, vision)
- Graphics/audio applications:
  - adjusting the contrast in a digital image
  - adjusting the volume of digital audio.

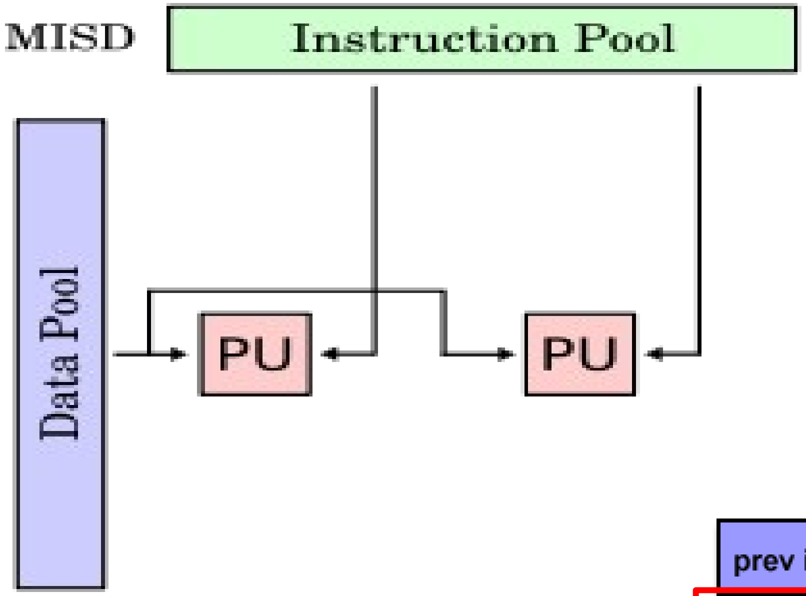
a	67	43	12	56	75	14	7	10
b	3	89	20	43	23	77	95	36
c								

SISD:	SIMD:
for (i=0; i<8; i++) a[i] = b[i] + c[i];	a = b + c; // vector addition

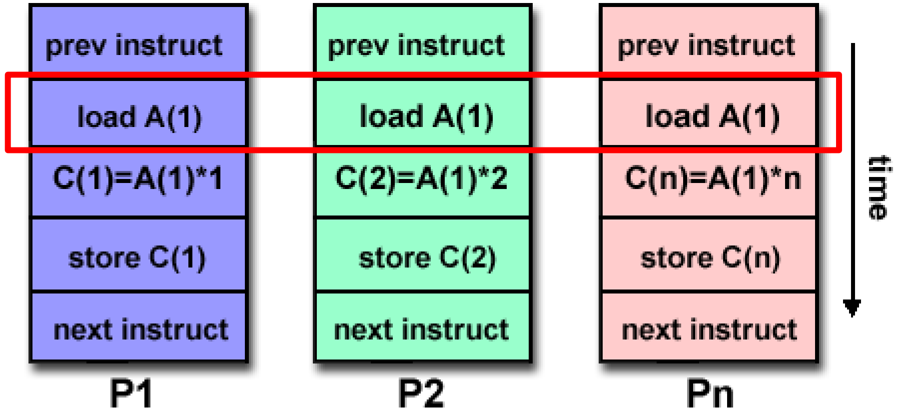
SIMD: Single Instruction Multiple Data

Example in video games Xbox 360: Xenon CPU (XCPU) with SIMD architecture

MISD: Multiple Instruction Single Data



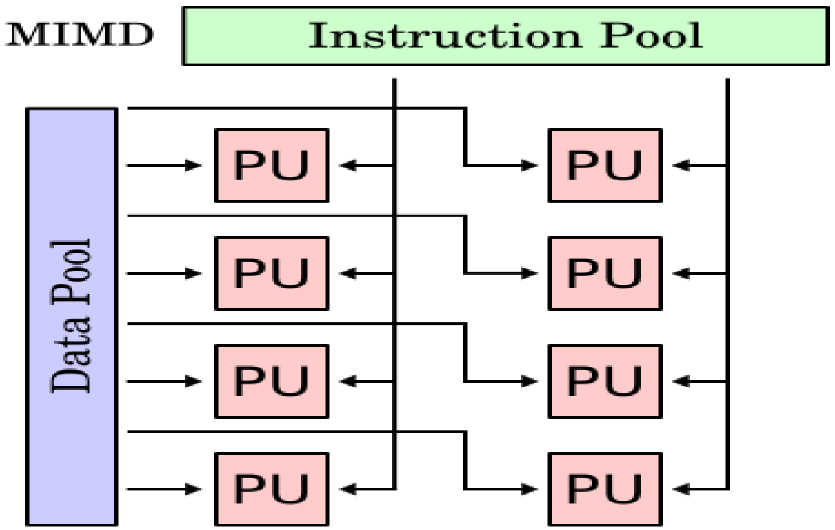
Different operations on the same data



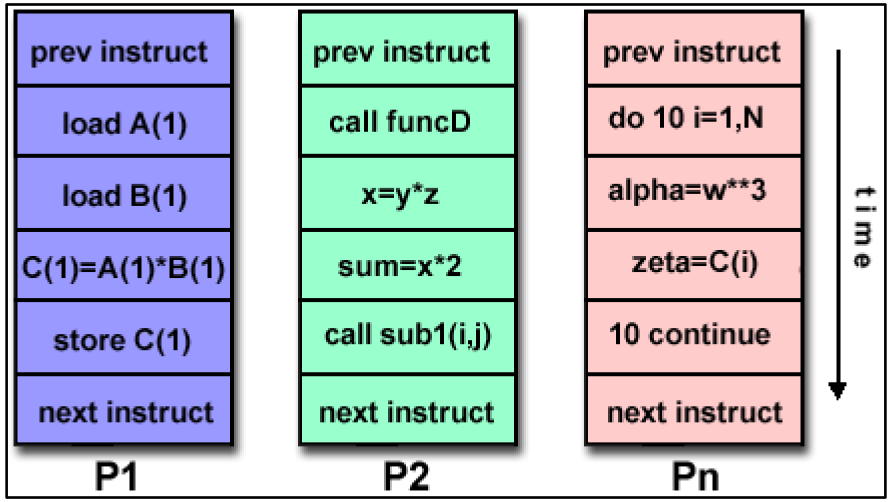
Fault tolerant computing

MISD used in Space Shuttle flight control computers

MIMD: Multiple Instruction Multiple Data



Multiple processors functioning asynchronously and independently



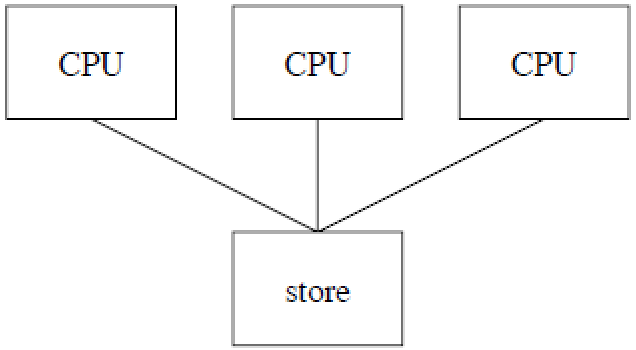
Modern multi-core processors are MIMD.
Multi-core processor is a special kind of a multiprocessor where all processors are on the same chip

Reference: http://www.ijarcce.com/upload/2014/january/IJARCCE6G__s_prabhudev_parallel.pdf

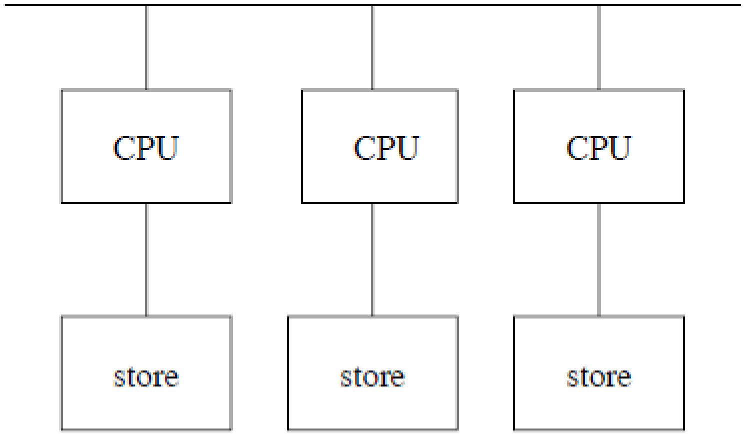
Memory Architectures

There are two possible memory architectures

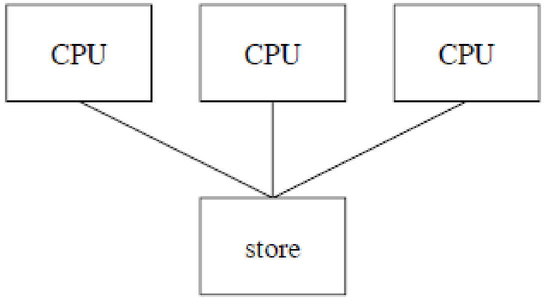
Shared Memory: each processor has access to the same memory space



Distributed Memory: each processor has its own data store (memory).



Shared Memory

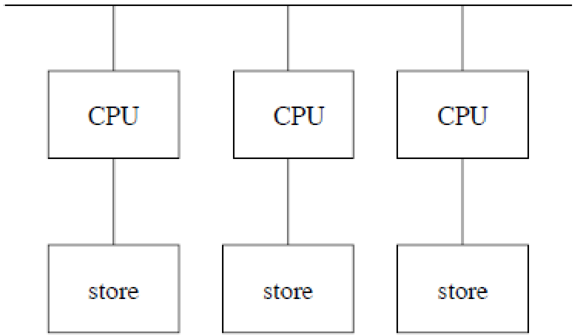


Sharing is an efficient use of the memory space, but:

- It leads to memory-to-CPU bottlenecks
- *Cache coherence* becomes a problem (to be solved by *coherence protocols).*

Because of this, shared memory architectures typically do not scale very well.

Distributed memory

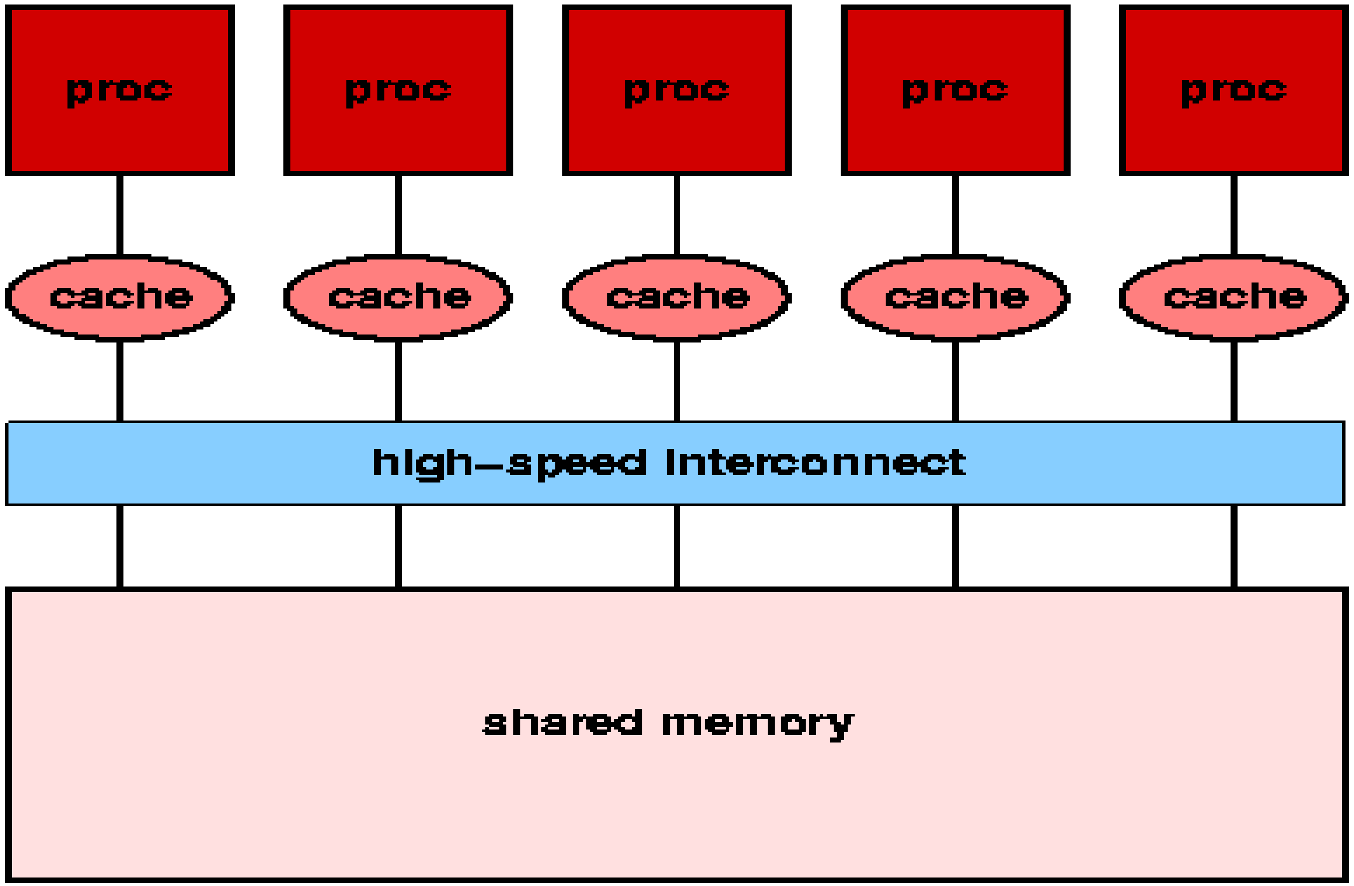


- Straightforward to scale up for large numbers of parallel processors

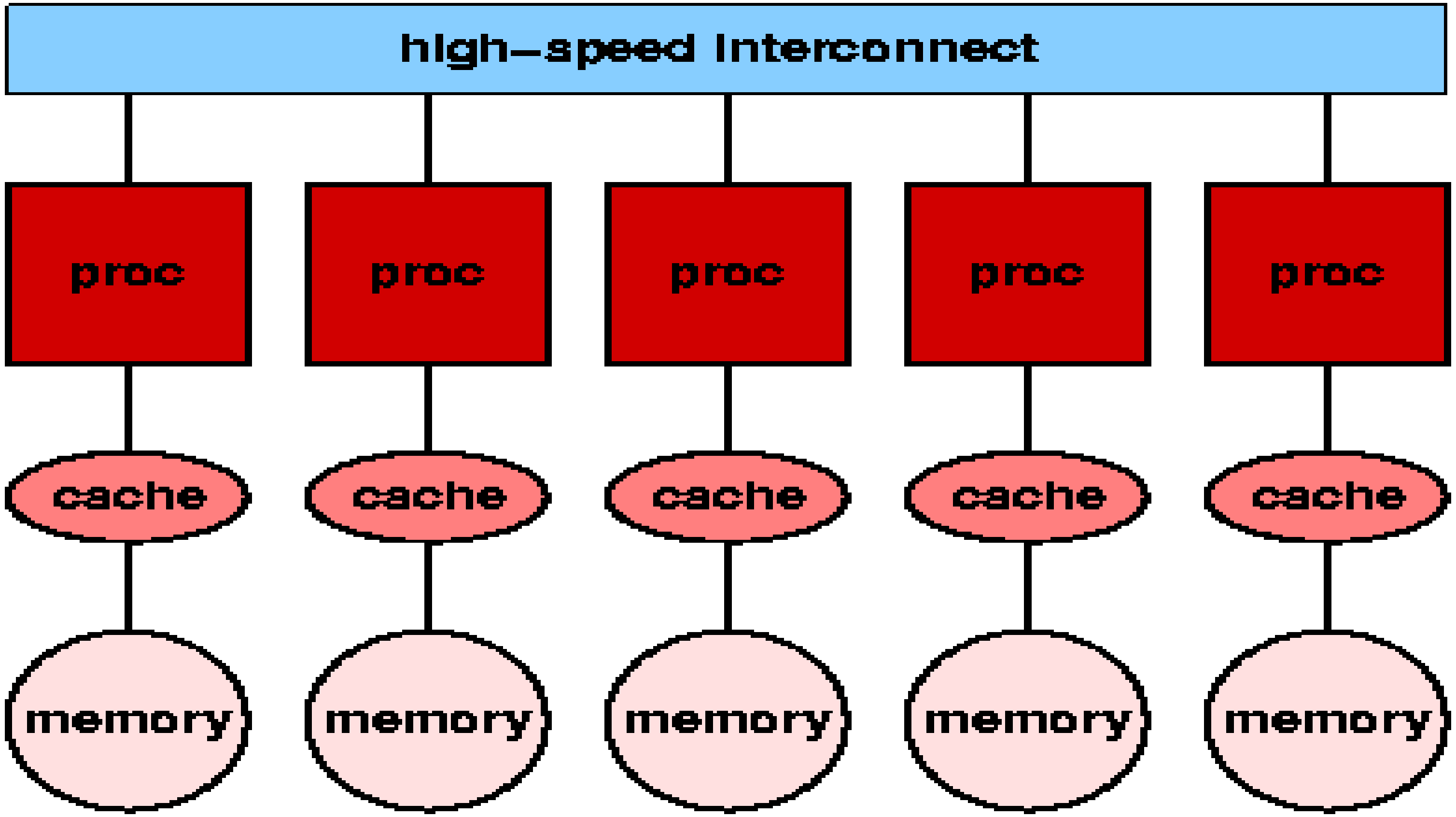
- Communication between processors is indirect --- by message passing --- and hence inefficient.

- Distributing (before computation) and then reassembling the data (after computation) is a non-trivial task.

Shared Memory



Distributed memory



Compromise solutions

- Virtual shared memory --- distributed memory which “seems” shared to the CPUs.

- Non-uniform memory access (NUMA) --- shared memory which has parts which are faster for each CPU.

Shared memory MIMD

- Shared memory MIMD is becoming dominant in general purpose computing, due to the diminishing returns on building larger/faster uniprocessors

- Multi-core processors ---- multiple CPUs on the same chip --- are now standard.

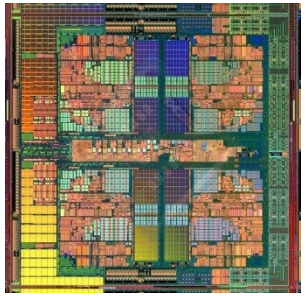
- These are typically used for applications which are intrinsically parallel --- *multitasking*

- A key challenge is designing software which exploits their capabilities effectively.

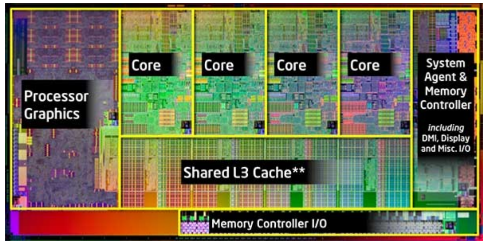
Shared memory MIMD: example multi-core

Multiple cores on Chip: each core simpler and lower power than a single large core

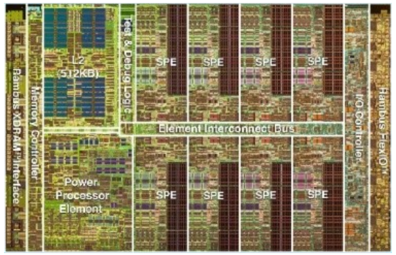
Large scale parallelism on chip



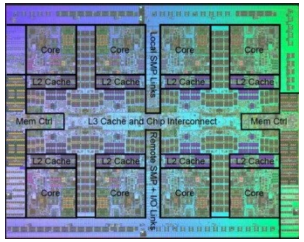
AMD Barcelona
4 cores



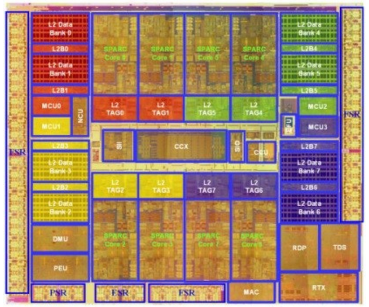
Intel Core i7
4 cores



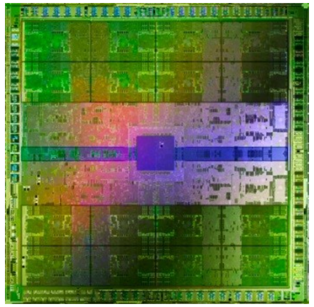
IBM Cell BE
8+1 cores



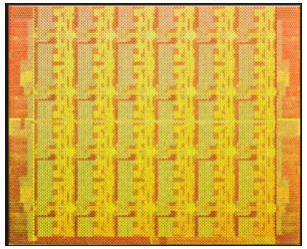
IBM POWER7
8 cores



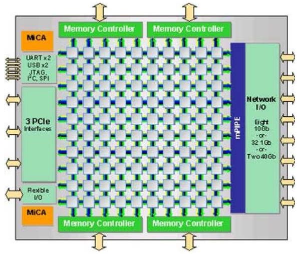
Sun Niagara II
8 cores



Nvidia Fermi
448 “cores”



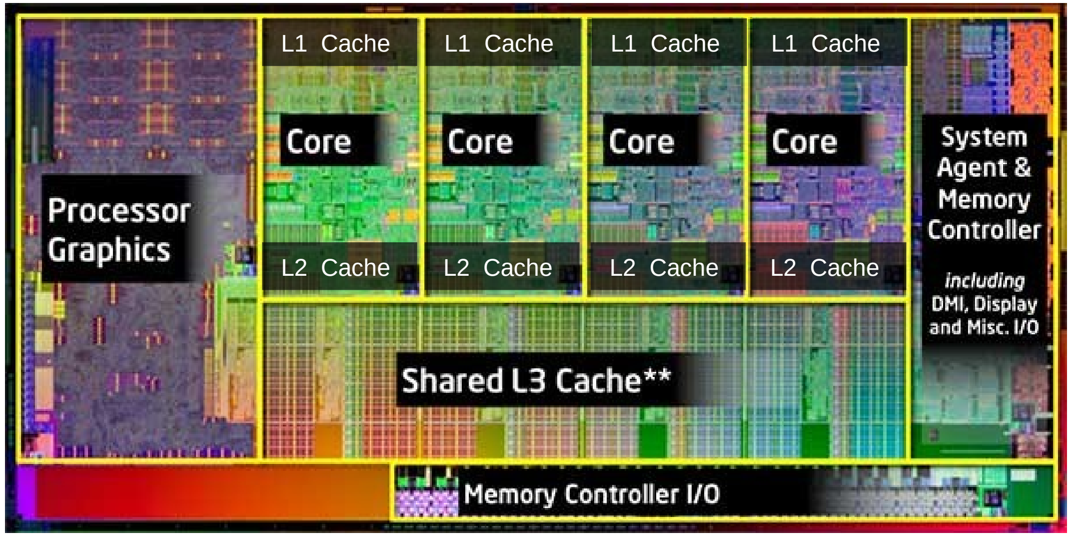
Intel SCC
48 cores, networked



Tilera TILE Gx
100 cores, networked

Slide credit: Prof Onur Mutlu

Shared memory MIMD



*Intel Core i7
4 cores

- Typical advantages of multi-core processors:
  - Editing a photo while recording a TV show through a digital video recorder
  - Downloading software while running an anti-virus program

- Parallel programming techniques more and more important

*The Intel 14th generation Core i9 has 16 cores

Summary

In this lecture we have discussed different forms of parallel processors:

- Different control architectures: their uses, advantages and disadvantages, and their place in *Flynn's taxonomy.*

- Stores for parallel computing: shared versus distributed memory.

Next time:

Parallel architectures:

- Amdahl's law --- how the sequential element of a computation limits the speedup by parallelization.