Introduction to High Performance Computing

Fabio AFFINITO

www.cineca.it

What is the meaning of High Performance Computing?

What does HIGH PERFORMANCE mean???

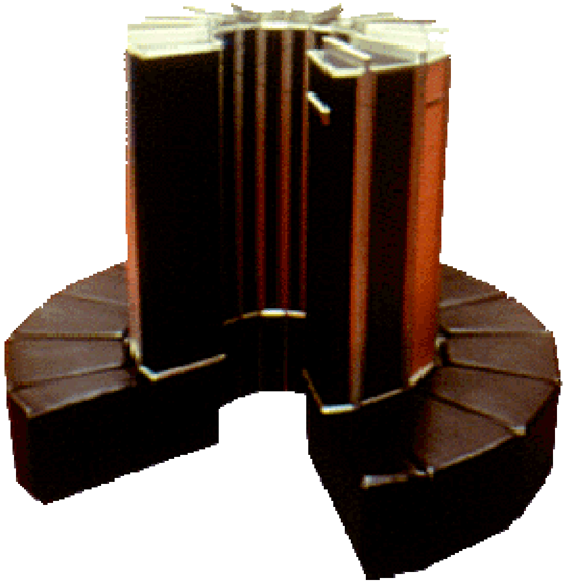
1976... Cray-1 supercomputer

First commercial successfull vector machine:

- weigth: 2400 Kg
- cost: approx. 8M$

- performance: 160MFlop/s

1 Mflop/s =
one million floating point
operations per second



2012... a commercial notebook, for example with a quad-core Intel Nehalem i7 3GHz CPU

- weight: approx 3-4 kg
- cost: approx. 1500 €

- performance: approx. 48GFlop/s

Peak performance: 300X
Cost: 1/8000

Looking for "HPC" on *Wikipedia*...

You are redirected on the "supercomputer" term.

> *A supercomputer is a computer at the frontline of current processing capacity, particularly speed of calculation.*
>
> *[…]*
>
> *Supercomputers are used for highly calculation-intensive tasks such as problems including quantum physics, weather forecasting, climate research, oil and gas exploration, molecular modeling (computing the structures and properties of chemical compounds, biological macromolecules, polymers, and crystals), and physical simulations (such as simulation of airplanes in wind tunnels, simulation of the detonation of nuclear weapons, and research into nuclear fusion).*

The need for metrics: **the peak performance**

Definition: The peak performance is the theoretical maximum performance (usually measured in terms of 64-bit floating point operation per second) achievable by a computing system.

So you have to take into the account:
- the clock frequency
- how many operations per clock tick

An example from Intel:

- Netburst core architecture: maximum of 2 floating point operations per clock tick. It means that a 3GHz netburst core could do a maximum of 6GFlop/s as theoretical clock performance (TPP).

- core/core2/i7 family of processors have doubled the floating point capacity (4 floating point operations per clock tick). So each core of a 3GHz has a TPP of 12GFlop/s

- Sandy Bridge has enhanced vectorizing capabilities that allows for 8 floating point operations per clock tick

Can you reach the peak of the mountain?

Climbing the way to the theoretical peak performance is a hard task: in real applications usually you can reach a 20-30% of the TPP (with great effort!).

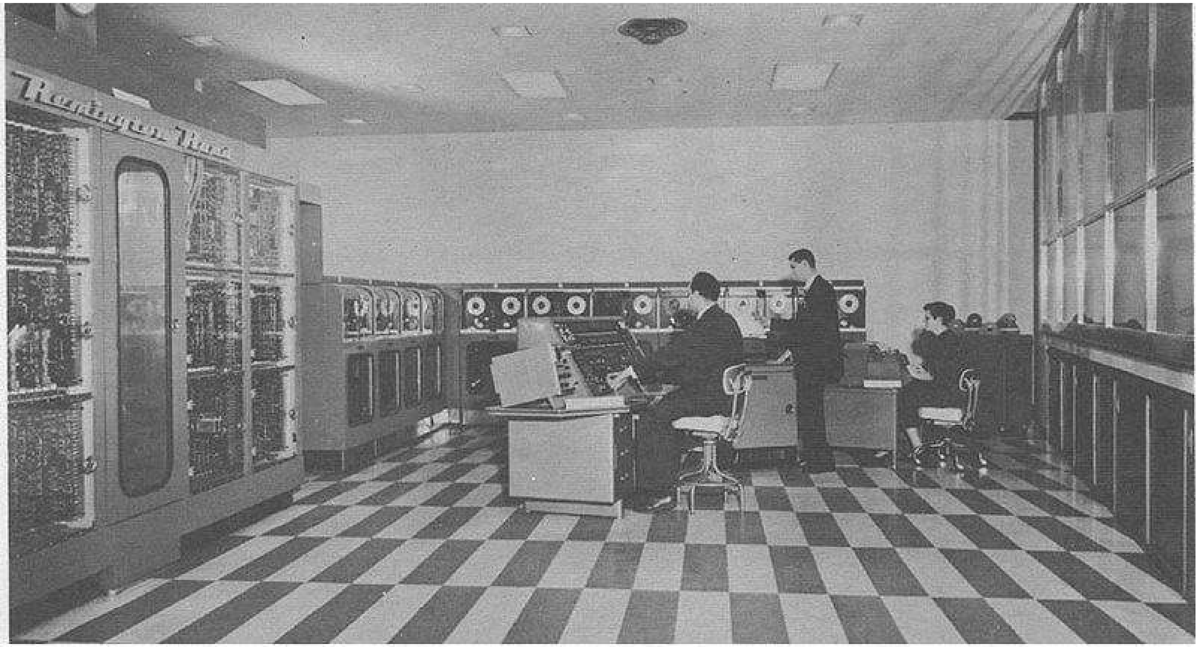
Why?

- it’s very difficult to exploit the vector capabilities of the modern architectures;
- bandwith of memory to cache is limited
- conflicts in cache/memory access
- communication time dominates over computation time
- number crunching is not the only important thing in life...

So?

TPP provides a way to a (fair?) comparison between different machines.

Now i want to tell you a story...



1939. Atanasoft & Berry build prototype electronic/digital computer at Iowa state.

1941. Conrad Zuse completes Z3, first functional programmable electromechanical digital computer

1943. Bletchley Park operates Colossus, computer based on vacuum tubes by Turing, Flowers and Newman

1946. ENIAC, Eckert and Mauchly at the Univ. Of Pennsylvania.

1951. UNIVAC I (also designed by Eckert and Mauchly), produced by Remington Rand, delivered by the US Census Bureau

1952. ILLIAC I (based on Eckert, Mauchly and von Neumann design), first electronic computer built and housed at a University

1962. Control Data Corp. introduces the first commercially successfull supercomputer the CDC6600, designed by Seymour Cray, TPP of 9 MFlop/s

1967. Texas Instruments' Advanced Scientific Computer, similar to CDC6600 includes vector processing instructions

1968. CDC6800 Cray's redesign of 6600 remains fastest supercomputer until mid 1970's. TPP of about 40MFlop/s

1976. Cray Research Inc.'s Cray-I starts vector revolution. TPP of about 250 MFlop/s

1982. Cray X-MP, Cray's first multiprocessor computer, originally 2 processor design had a TPP of 400 Mflop/s included shared memory between processors.

1993. Cray introduces the T3D. An MPP (massively parallel processing) based on 32 2048 DEC Alpha processors and a proprietary 3D torus network

1997. ASCI Red at Sandia delivers the first Tflop/s (on the Linpack) using Intel Pentium Pro processors and a custom interconnect

2002. NEC's Earth Simulator is a cluster of 640 vector supercomputers. I delivers 35.61 Tflop/s on the Linpack benchmark.

2005. IBM's Blue Gene systems regain top rankings (again, according to Lingpack) using massive munmbers of embedded processors and communication sub-systems, each running a stripped down Linux-based operating system.

2008. IBM deploys a hybrid system of Opteron nodes coupled with Cell processors interconnected via Infiniband. It achieves first sustained Pflop/s on Top500

2010. Tianhe-I at the National Supercomputing Center in Tianjin, China. It is a mix of 14.336 Intel Xeon X5670 processors (86016 cores) and 7168 Nvidia Tesla M2050 general purpose GPUs, custom interconnect. 2566 Pflop/s.

2011. K computer, at RIKEN in Kobe, Japan, 68544 @2.0GHz 8-core Sparc64 processors (548.352 cores); custom interconnect (TOFU). 8162 Pflop/s

2012. Blue Gene Q starts production in CINECA and at JSC.

The TOP500 list.

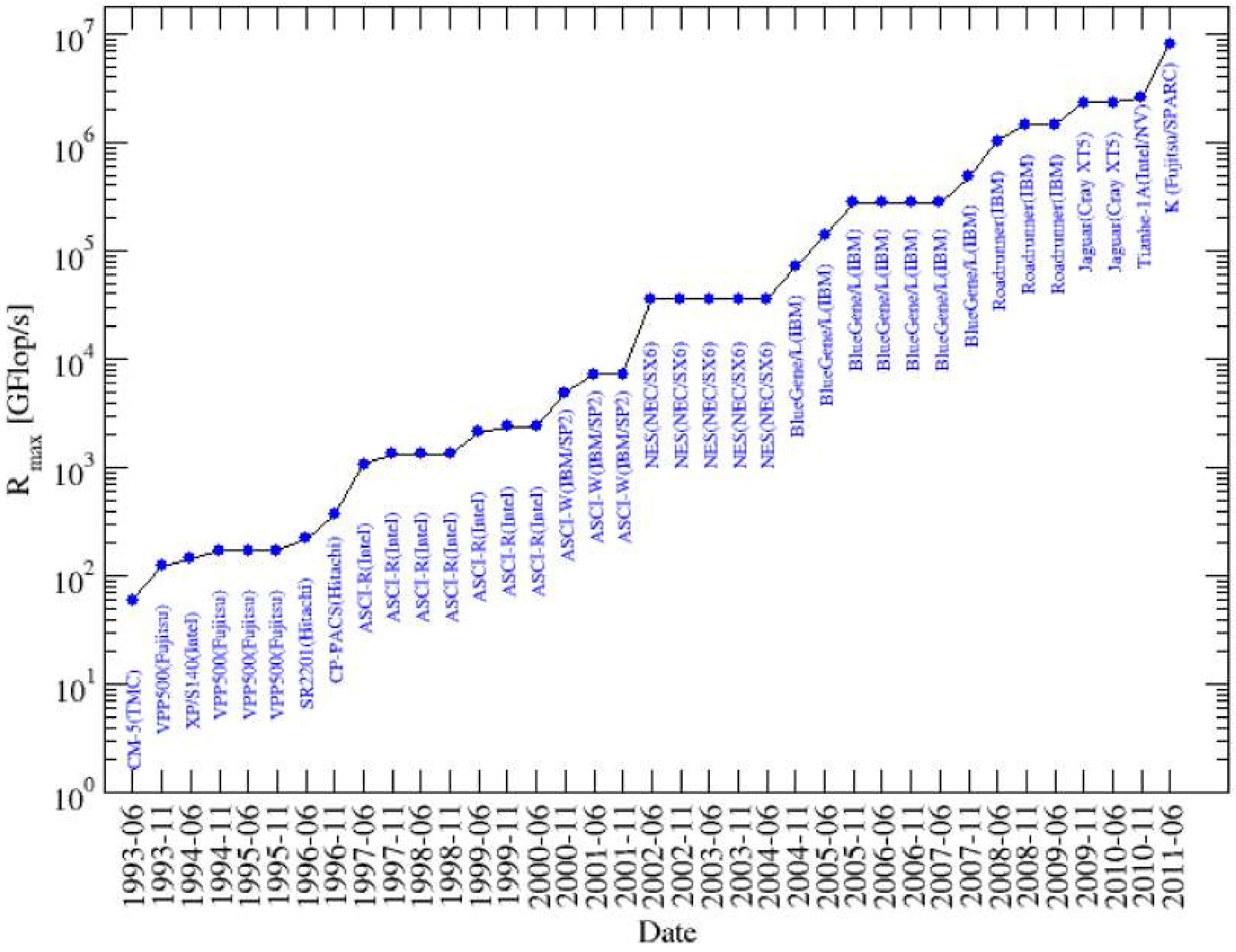
One measure of the performance of HPC system is given by the LINPACK benchmark that has been used since 1993.

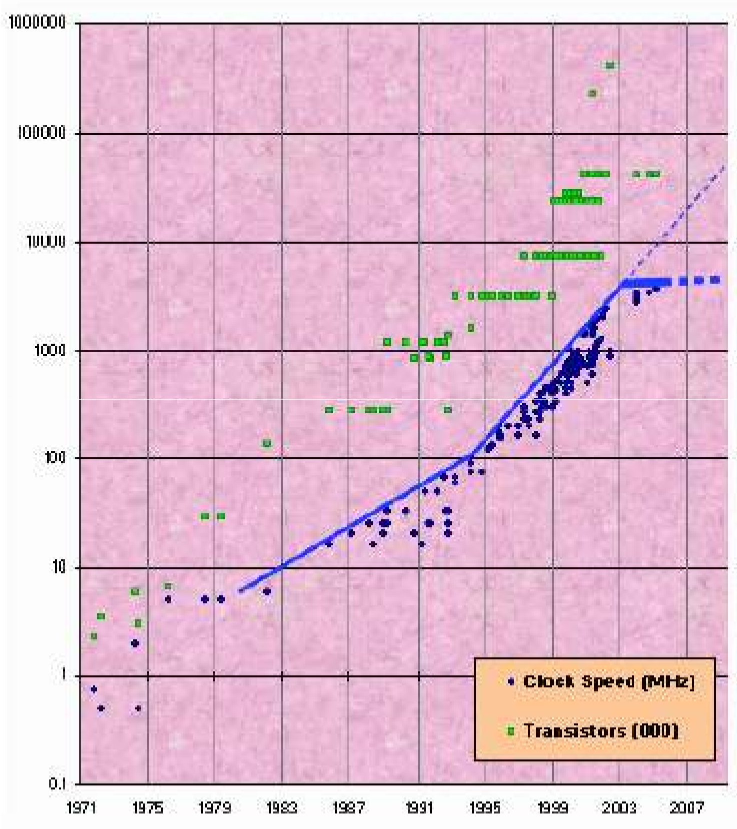
A top list based on the LINPACK benchmark is the www.top500.org list.

The LINPACK benchmark was introduced by Jack Dongarra:

- It solves a dense system of linear equations Ax=B
- For different problem sizes N, get maximum achieved performance R
- Must conform to the standard operation count for the LU factorization with partial pivoting i.e. 2/3 $N^3 + O(N^2)$
- A version is avalaible for distributed memory machines

#1 R_{max} in Top500 List 1993-present





Intel processors according to clock frequency and trasistor count (H.Sutter, Dr. Dobb’s Journal 30(3), March 2005)

- Processor frequency has reached a plateau due to power and design requirements

- Moore’s law of transistor density doubling every 18-24 months is still preserved

- Transistor logic is being devoted to innovations in multi-core processors

- It’s emerging a software issue: how do you manage with all those cores?

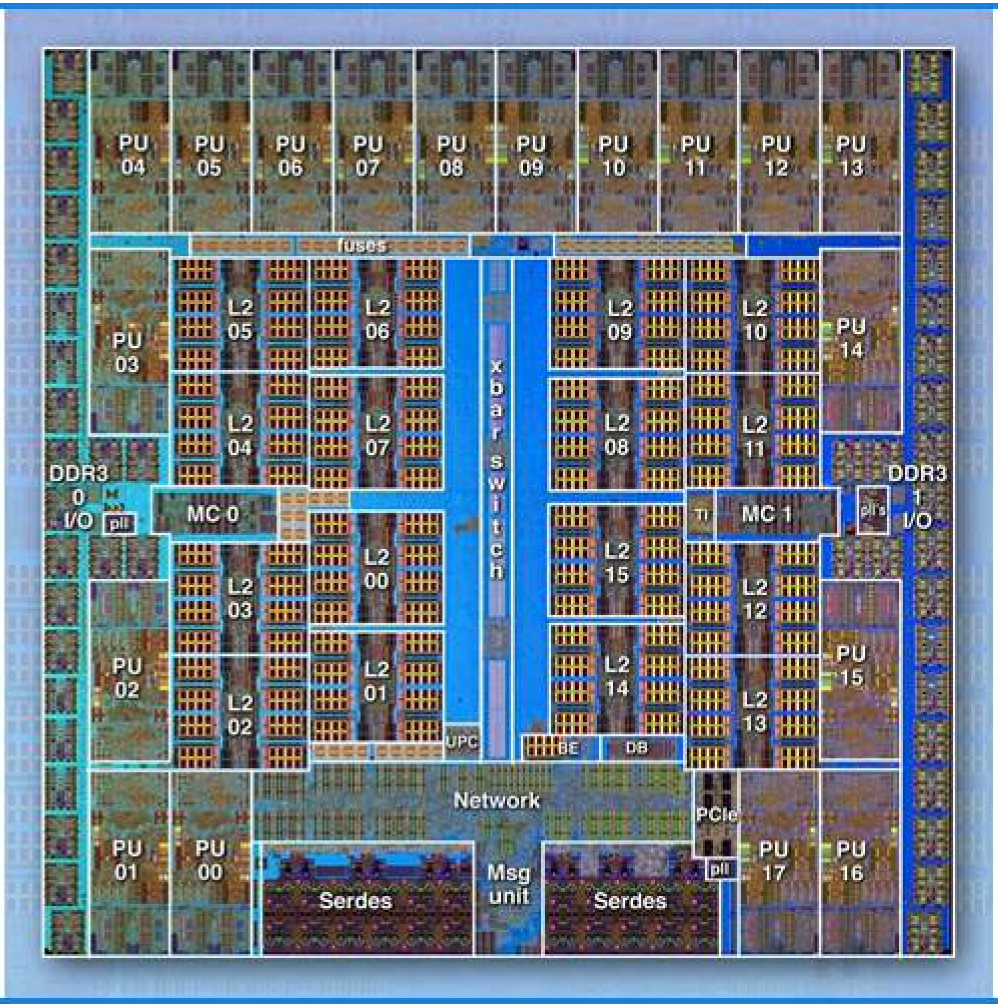
Around 2003, both Intel and AMD reached close to the limit of air cooling for their processors: why?

Power in a CPU is proportional to V^2fC

- Increasing frequency *f* also requires to increase voltage *V*: highly nonlinear!

- Increasing core count will increase capacity *C*, but it's only linear

The multi core era is a necessity!

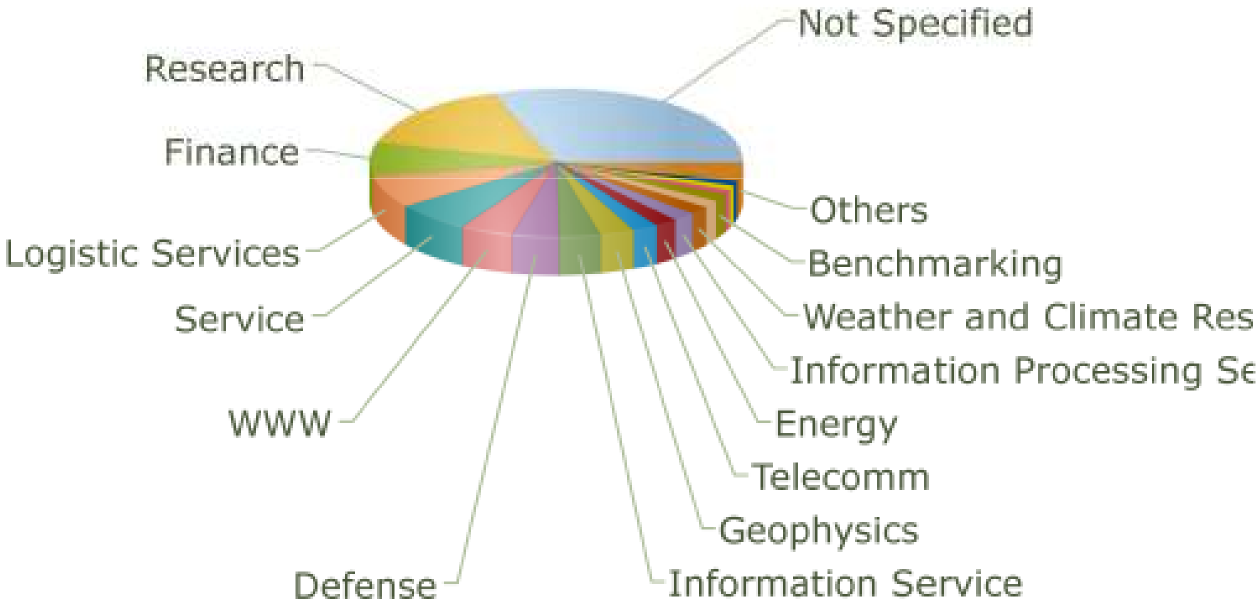


IBM Power A2 chip: 18 processing units

All the hardware is already parallel.

Let’s see how we can exploit that!

Who cares about HPC?



From top500 list in June 2011

Necessarily HPC = PARALLEL computation !!!

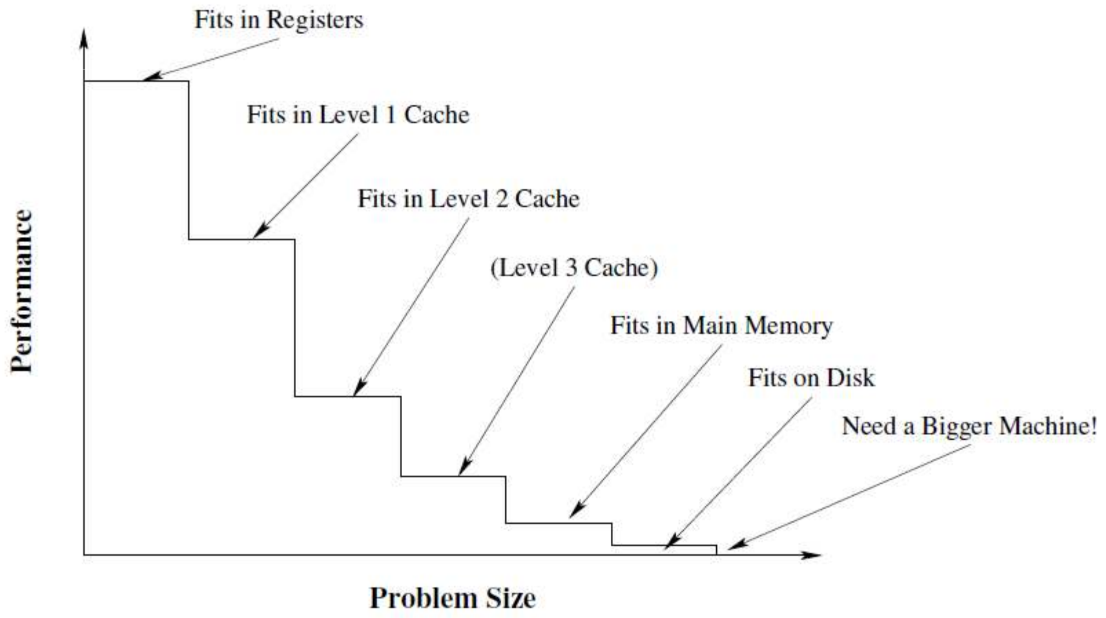
Why I need HPC?

-Because I want to solve my problem **faster** (for example in weather forecast or in financial analysis)

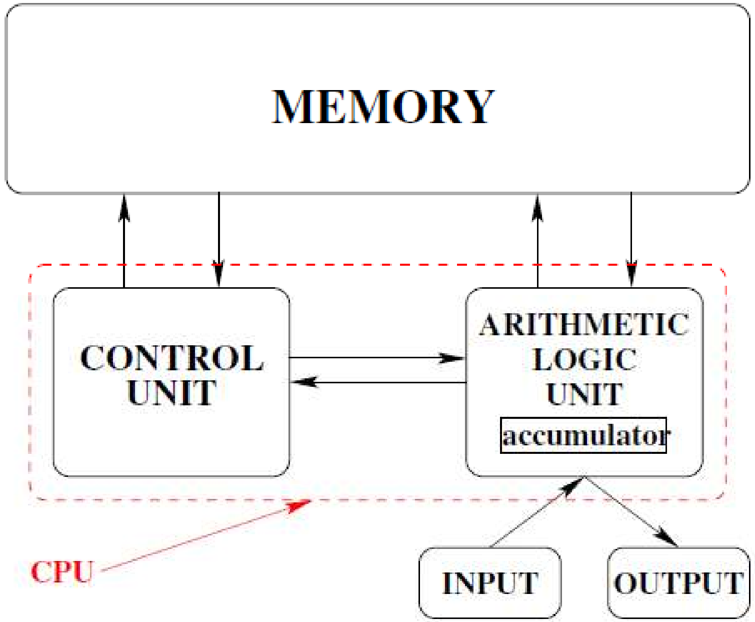
-Because I want to solve a **bigger** problem than I (or anyone else) have ever before been able to tackle in a reasonable time

The damnation of data storage.

Unfortunately it’s not all about Flop/s: **the memory wall**.

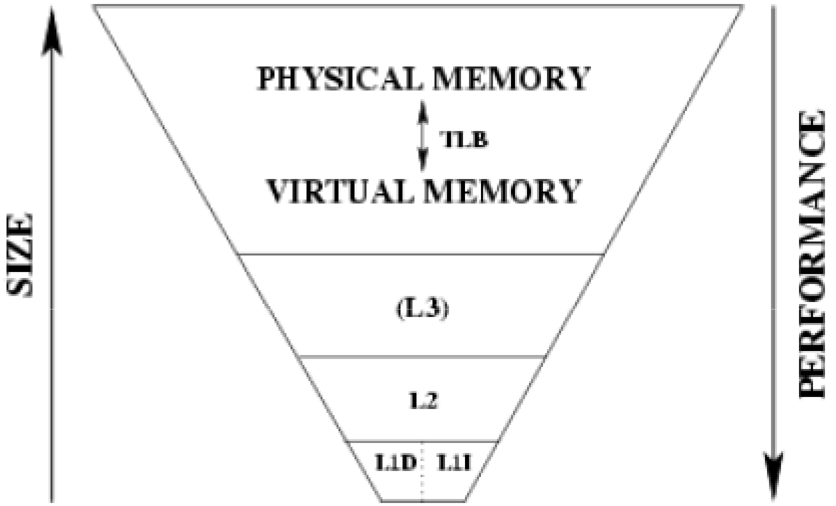


Let’s go back to the von Neumann, 1946.



Pretty much every modern processor follows this design.

The memory hierarchy



As the caches increases in size, they decrease in performance (bandwidth and latency)

The code performance improve when the instructions that compose a heavy computational kernel (eg. a loop) fit into the cache

The same applies to the data, but in this case the work of optimization involves also the programmer and not just the system software.

DEC Alpha 21164 (500 MHz):

Memory access time

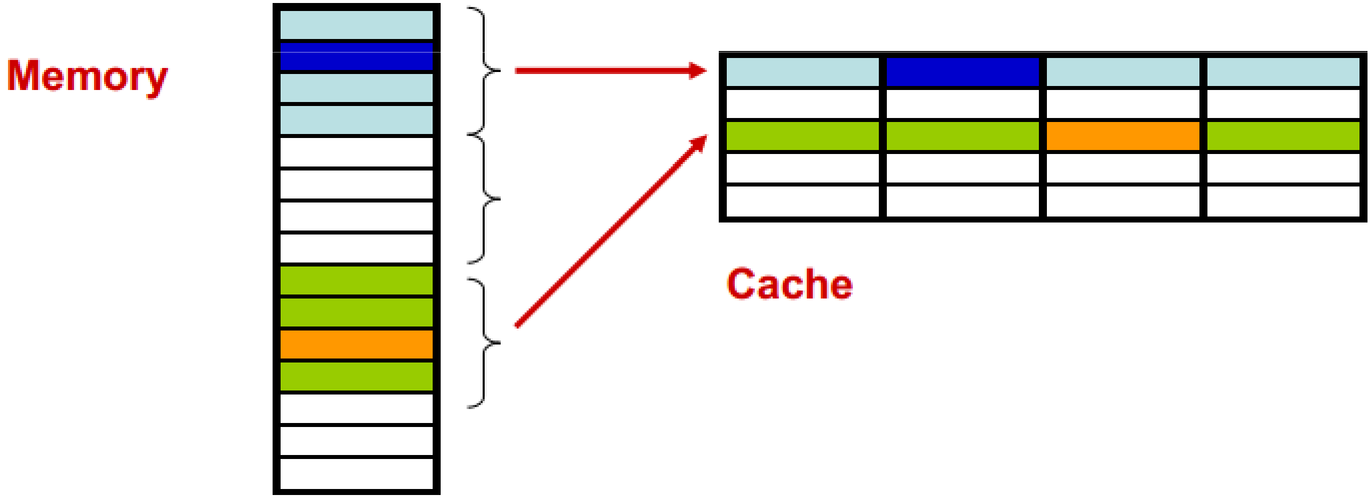
Registers	2 ns
L1 On-chip	4 ns
L2 On-Chip	5 ns
L3 Off-Chip	30 ns
Memory	220 ns

IBM SP Power 6 (4.7 GHz):

Memory access time (in clock cycles)

Registers	
L1: 2 x 64KB	< 5
L2: 2 x 4MB	22 cc
L3: 32 MB	160 cc
Memory 128 GB	400 cc

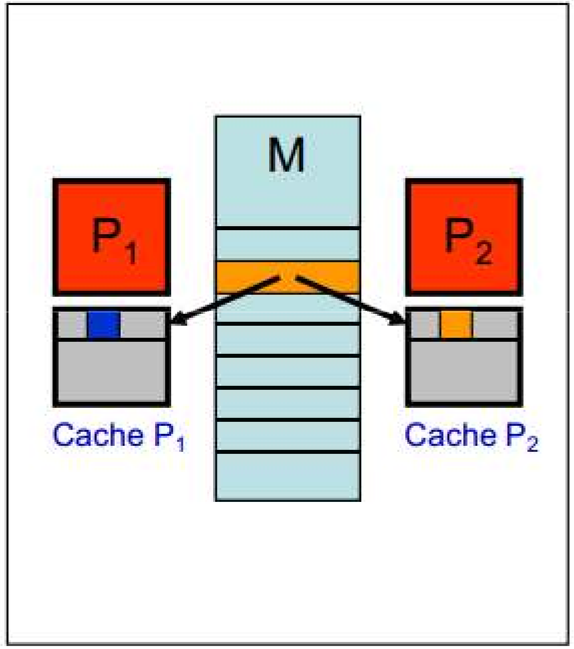
The cache is divided into slots of the same size (lines)
Each line contains k consecutive memory locations (ie 4 words).
When a data is required from memory, (if not already in the cache) the system loads from memory, the entire cache line that contains the data, overwriting the previous contents of the line.



When the CPU writes out a value the data must be updated in main memory

- Cache *write-back*: data written in the cache will stay there until the cache line is not required to store other data. When the cache line must be replaced, the data is written in memory.
- Cache *write-through:* data are written in cache and immediately to main memory.

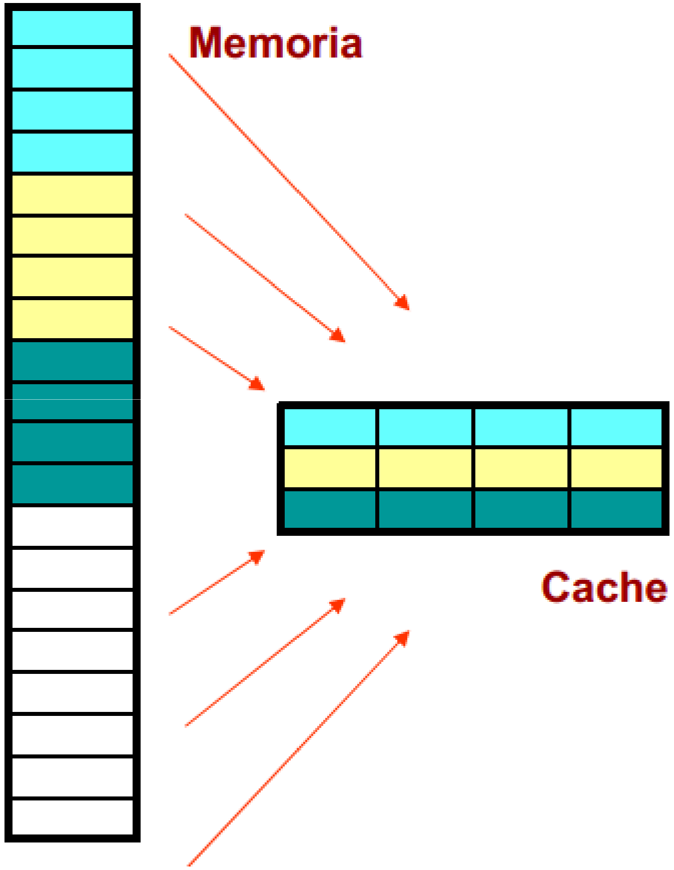
  In multi-processors systems, cache coherency must be managed: we need to access updated data in mail memory



Time by time the cache can contain only a sub-set of the data in memory.
A set of memory locations must be associated to a cache line (mapping).

Based on the mapping cache can be organized in one of the following ways:

- ***Direct mapped***
- ***Fully associative***
- ***Set associative***

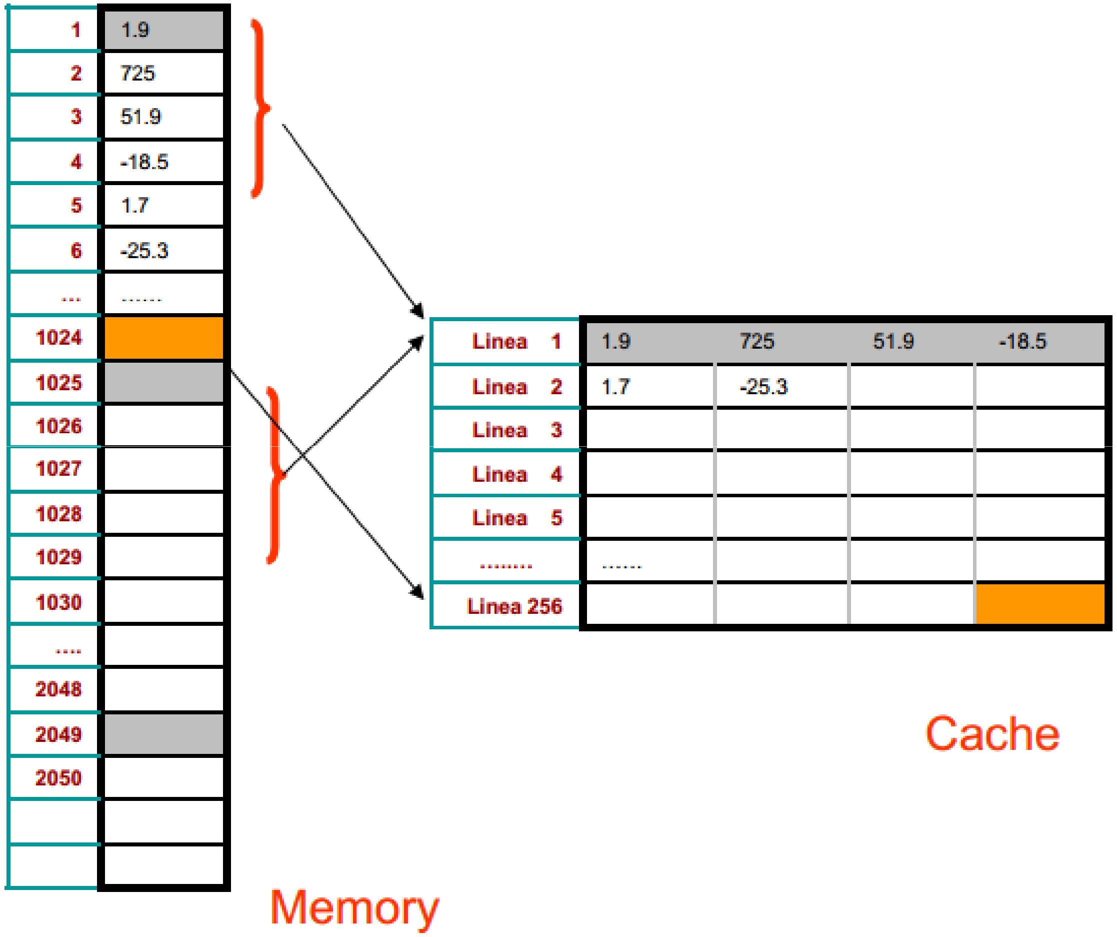


The way in which memory locations are mapped in the cache lines can affect the performance of the programs: two memory locations heavily used can be mapped or not on the same line.

With this scheme the first memory location (word 1) is mapped in line 1, as well as the location *d+1*, *2d+1*, *3d+1* etc. where

$d = N°_{\text{lines}} * N°_{\text{words per line}}$

If different memory references in turn point to the same cache line (eg, word 1, word 1025, word 2049), each reference causes a **cache miss** and the line just inserted must be replaced.
A lot of extra work (**overhead**) is generated.
This phenomenon is called **thrashing**.



1	1.9
2	725
3	51.9
4	-18.5
5	1.7
6	-25.3
...	......
1024	🟧
1025	▨
1026	
1027	
1028	
1029	
1030	
....	
2048	
2049	▨
2050	

Memory

Linea 1	1.9	725	51.9	-18.5
Linea 2	1.7	-25.3		
Linea 3				
Linea 4				
Linea 5				
....	......			
Linea 256				🟧

Cache

Address

Content

Indicates the word within the line

Indicates the line

Memory

0	0000	000	00	1.9
1	0000	000	01	725
2	0000	000	10	51.9
3	0000	000	11	-18.5
4	0000	001	00	1.7
5	0000	001	01	-25.3
..	...	...	..	......
10	0000	010	10	⬜
11	0000	010	11	⬜
12	0000	011	00	🟩
..	...	...	..	......
29	0000	111	01	🟧
30	0000	111	10	🟧
31	0000	111	11	🟧
32	0001	000	00	🟦
33	0001	000	01	🟦
..	...	...	..	......
62	0001	111	10	🟧
63	0001	111	11	🟧
64	0010	000	00	🟦
65	0010	000	01	🟦

Linea 0	1.9	725	51.9	-18.5
Linea 1	1.7	-25.3		
Linea 2			⬜	⬜
Linea 3	🟩			
Linea 4				
Linea 5				
Linea 6				
Linea 7		🟧	🟧	🟧

Cache

8 lines

4 word per line

With this scheme, each memory location can be mapped to any cache line, regardless of the memory location.

The name comes from the type of memory used to build this type of cache (**associative memory**).

When the CPU needs a given data, this is required in all the cache lines simultaneously. If a line contains the data, this is sent to CPU, otherwise a cache miss occurs.

In general, the least recently used line will be overwritten with the new data. (**LRU policy**)

Fully associative caches are costly but provide a higher usage when compared to the direct mapped cache

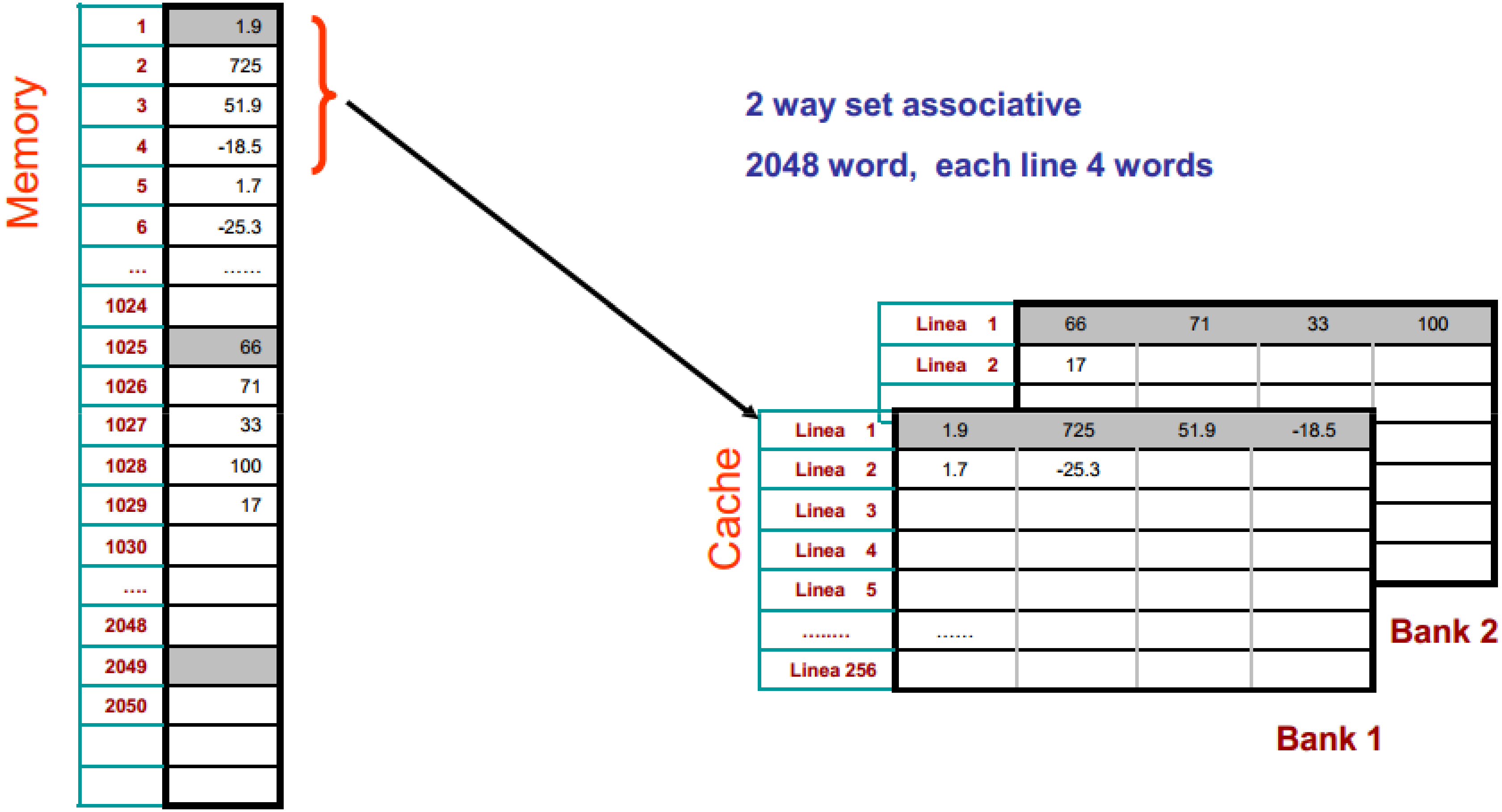
Practically it is a direct mapped cache replicated in multiple copies (banks) i.e. 2 or 4 separate banks of cache (*two-way, four-way set associative*).

With this scheme, if a memory location is mapped in line k, a further reference to a memory location always mapped to the same line, will be allocated in line K of a different bank.

Set Associative cache is less susceptible to cache thrashing compared to a direct mapped cache with the same size.

2 way set associative

2048 word, each line 4 words



1	1.9
2	725
3	51.9
4	-18.5
5	1.7
6	-25.3
...	......
1024	
1025	66
1026	71
1027	33
1028	100
1029	17
1030	
....	
2048	
2049	
2050	

Bank 2

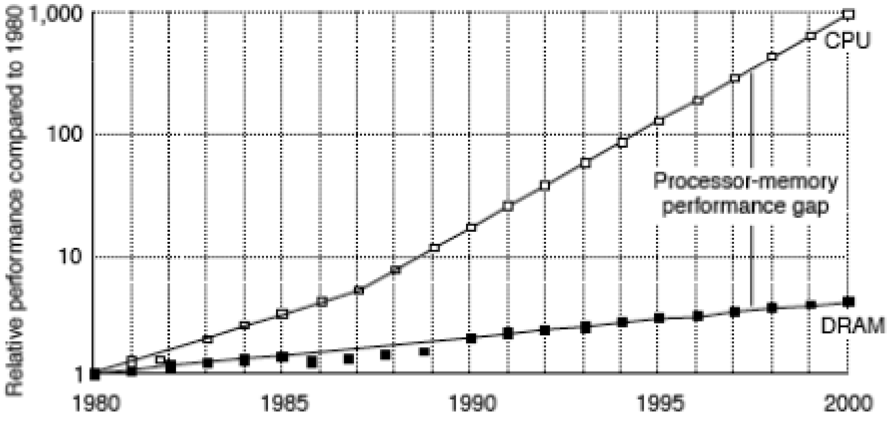
Linea 1	66	71	33	100
Linea 2	17			

Bank 1

Linea 1	1.9	725	51.9	-18.5
Linea 2	1.7	-25.3		
Linea 3				
Linea 4				
Linea 5				
....	......			
Linea 256				

CPU/Memory performance gap

- CPU performance increases roughly 80% per year
- Memory performance increases 7% per year



D. Patterson et al. (IEEE Micro 17,34 – 1997)

Scaling concepts

By “scaling” we intends the relative performance of a parallels versus a serial implementation.

We can define a speed-up factor S(p)

$$S(p) = \frac{\text{Sequential execution time}}{\text{Parallel execution time using p processors}}$$

In the ideal case S(p)=p

If we define *ts* the (best) sequential execution time we can note that:

$$S(p) = \frac{ts}{tp(p)} \le p$$

From here we can define the parallel efficiency:

$$E(p) = \frac{S(p)}{p} = \frac{ts}{p \times tp}$$

In practise, there are limitation to the speedup of a code:

- A fraction of the code f cannot be made to execute in parallel

- Parallel overhead (communication, libraries, etc.)

-...

Using the definition of the serial fraction of the code f we can note that

$$tp \geq f \cdot ts + (1 - f) \cdot {}^{ts}\!/_{p}$$

$$S(p) = \frac{ts}{tp(p)} \le p$$

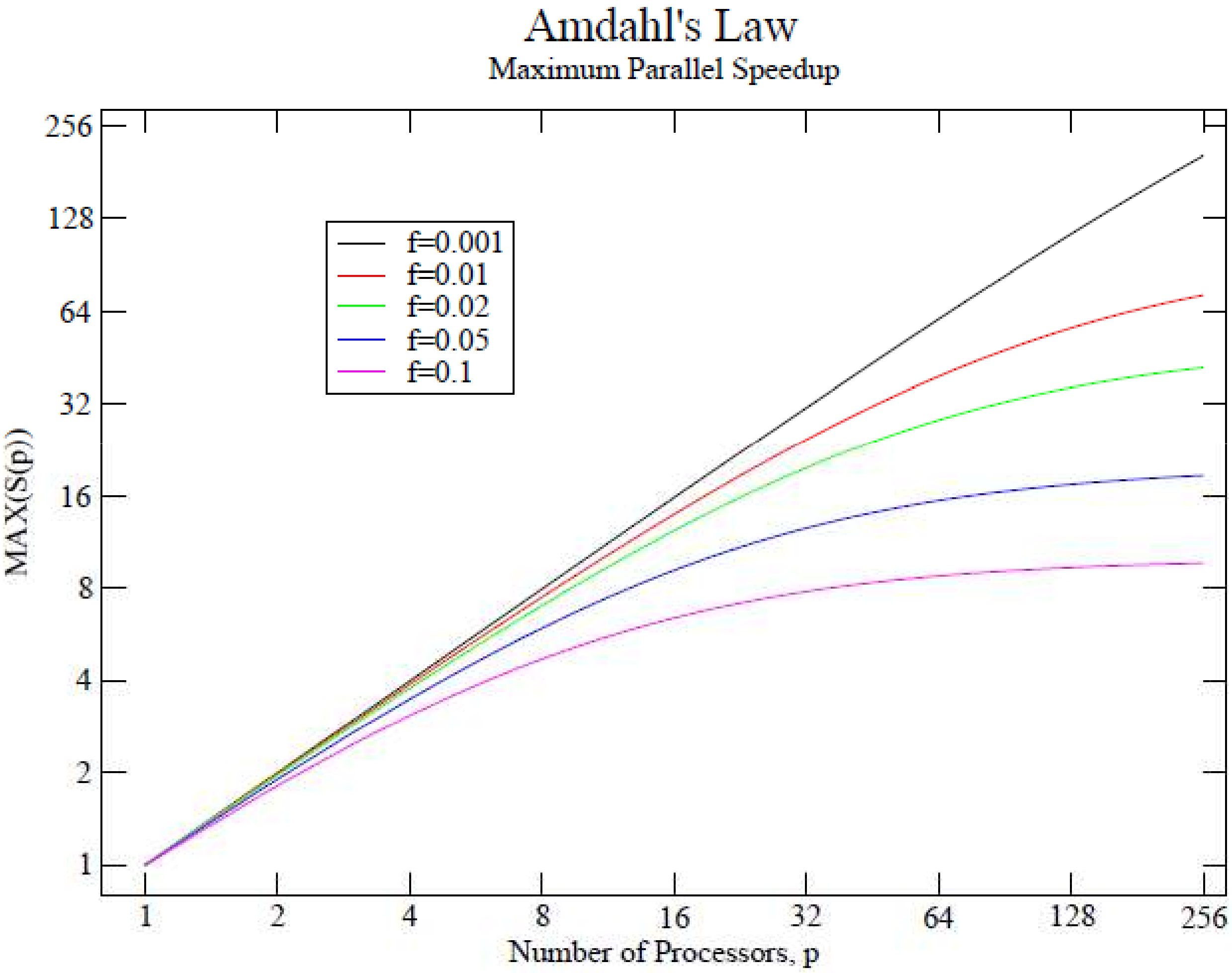
$$tp \ge f \cdot ts + (1-f) \cdot {}^{ts}\!/\!_{p}$$

Legge di Amdahl

$$S(p) \le \frac{p}{1 + f(p-1)}$$

$$\lim_{p \to \infty} S(p) \le \frac{1}{f}$$

If the serial fraction is 5% the maximum parallel speedup is only 20


Amdahl's Law
Maximum Parallel Speedup
MAX(S(p))
256
128
64
32
16
8
4
2
1
f=0.001
f=0.01
f=0.02
f=0.05
f=0.1
1
2
4
8
16
32
64
128
256
Number of Processors, p

Flynn’s taxonomy

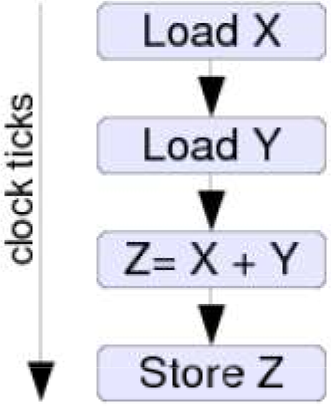
Based on the flow of information (both data and instructions) in a computer:

SISD. Single Instruction Single Data (*old-fashioned*)

SIMD. Single Instruction Multiple Data (*data-parallel*)

MISD. Multiple Instruction Single Data (*never implemented*)

MIMD. Multiple Instruction Multiple Data (*task parallel*)



clock ticks

Load X

Load Y

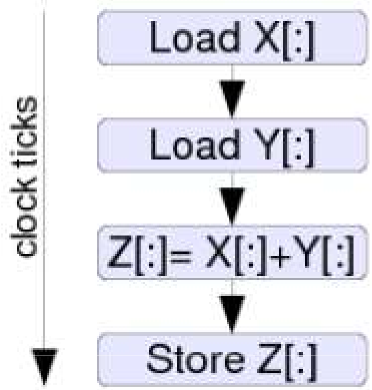
Z= X + Y

Store Z

SISD

- sequential, deterministic

-Sequential (non parallel) instruction flow
-Predictable and deterministic



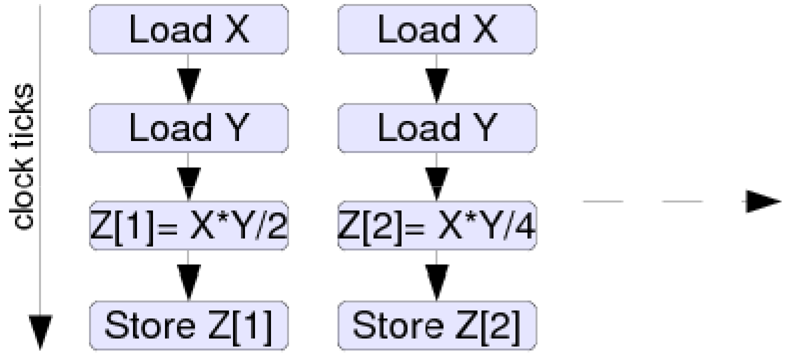
SIMD

- type of parallel computation
- may be implemented as vector pipeline and/or processor arrays

-Single instruction operates on different data in a given clock tick

-Requires predictable data access patterns: “vectors” that are contiguous in memory

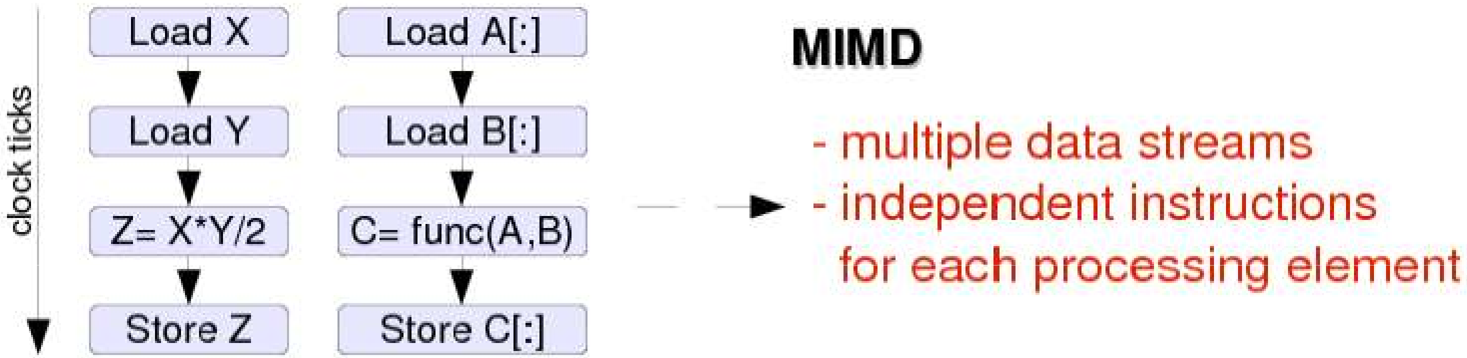
-Almost all modern processors use some form (for example SSE, AVX for Intel)



MISD

- single data stream
- independent instructions
  for each processing element

-Multiple instructions applied independently on the same set of data
-“Theoretical” rather than practical architecture – few implementations (if any)



MIMD

- multiple data streams
- independent instructions for each processing element

-Multiple instructions are applied independently on different data

-Very flexible – can be asynchronous, non deterministic

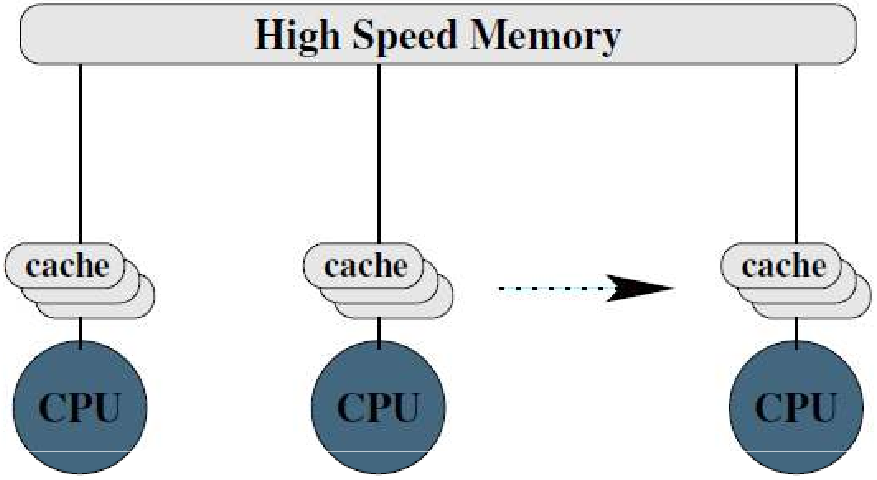
-Most modern supercomputers follow MIMD design, albeit with SIMD subsystems

What kind of parallelism?

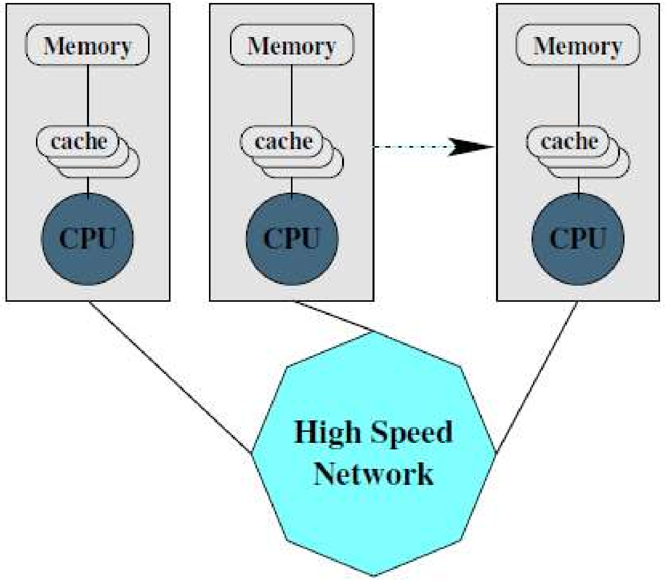
-Shared-memory (SMP). Multiple processor units act on the same memory locations. (OpenMP, HPF, ..)

-Distributed memory (message-passing). Each processor has its own memory and communicates with other “task units” using a interconnecting network. (MPI)

-Hybrid parallelism: SMP + Message-Passing.

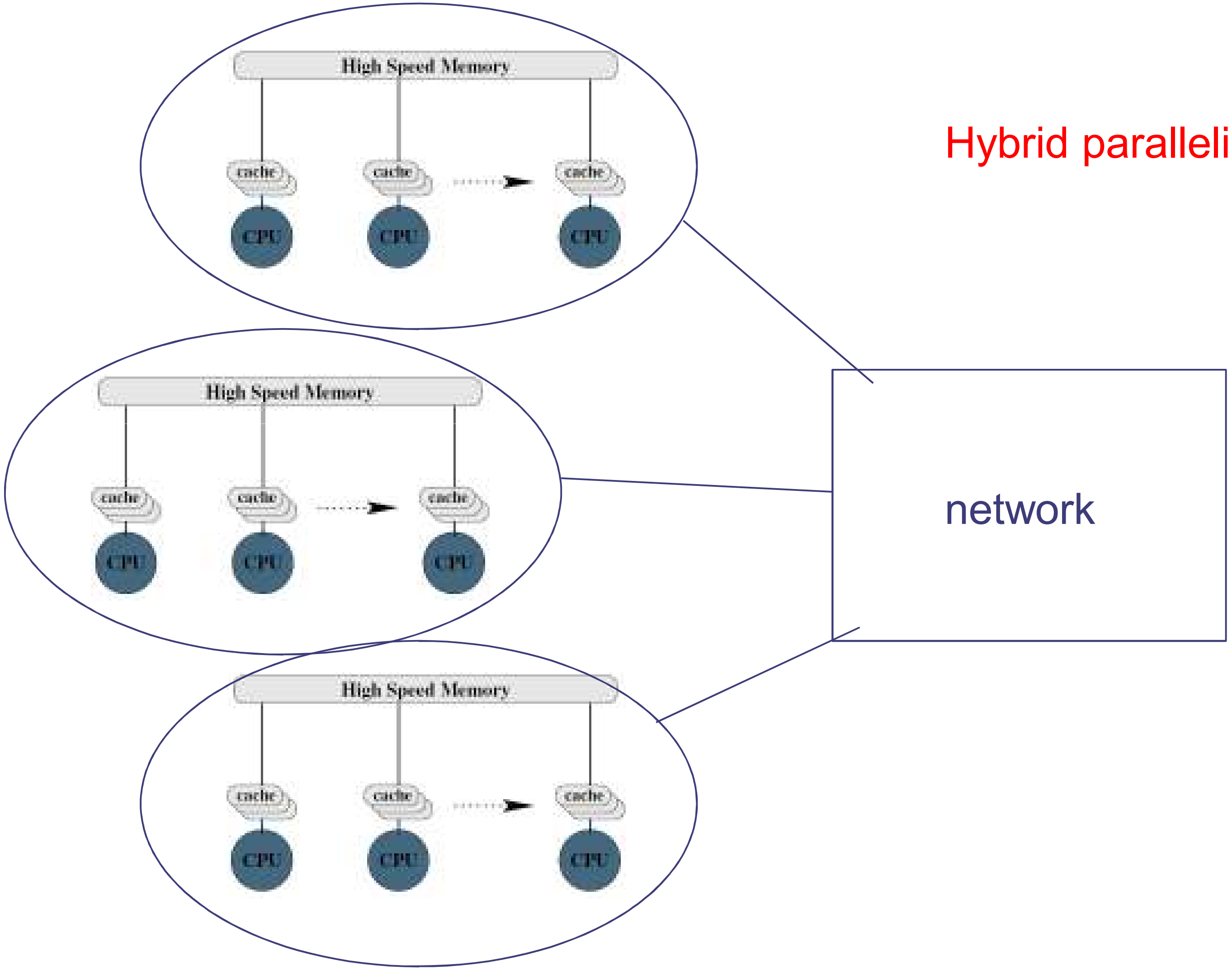


SMP



Distributed memory

Hybrid parallelism


High Speed Memory
cache
cache
cache
CPU
CPU
CPU
High Speed Memory
cache
cache
cache
CPU
CPU
CPU
High Speed Memory
cache
cache
cache
CPU
CPU
CPU
network