

Overview of main concepts

Physical limits: how fast can a serial computer be?

1 Tflop/s, 1 Tbyte
sequential
machine



$r = 0.3 \text{ mm}$

- Consider the 1 Tflop/s sequential machine:
 - Data must travel some distance, r , to get from memory to CPU.
 - Go get 1 data element per cycle, this means 10^{12} times per second at the speed of light, $c = 3 \times 10^8 \text{ m/s}$. Thus $r < c/10^{12} = 0.3 \text{ mm}$.
- Now put 1 Tbyte of storage in a 0.3 mm 0.3 mm area:
 - in fact $0.3^2 \text{ mm}^2/10^{12} = 9 \cdot 10^{-2} \cdot 10^{-6} \text{ m}^2/10^{12} = 9 \cdot 10^{-20} \text{ m}^2 = (3 \cdot 10^{-10})^2 \text{ m}^2 = 3^2 \text{ \AA}^2 \rightarrow$
 - Each byte occupies less than 3 square Angstroms, or the size of a small atom! (1 Angstrom = $10^{-10} \text{ m} = 0.1 \text{ nanometer}$)

No choice but parallelism !!!

IBM stores one bit of data on a SINGLE atom: Breakthrough may lead to credit card-sized devices capable of holding the entire iTunes library of music

- Prototype device uses world's smallest ever magnet using one atom of holmium
- Stores 1,000 times more information in the same space as existing technology
- Could lead to creation of radically smaller hard drives with vastly greater storage
- Entire 35 million song iTunes library could be saved to a pocket sized drive

By [TIM COLLINS FOR MAILONLINE](#)

PUBLISHED: 14:35 GMT, 9 March 2017 | **UPDATED:** 14:35 GMT, 9 March 2017

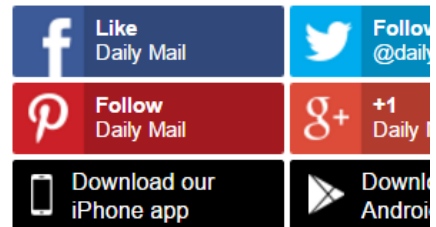


31
View comments

Imagine one day being able to carry vast libraries of data, currently stored in rooms full of servers, in your pocket.

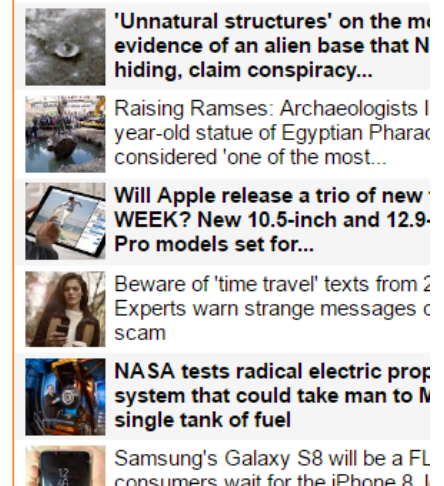
That's the dream of a team of IBM scientists who have built what they say is the world's smallest ever magnet, which uses a single atom to store information.

☒ Site ☐ Web



Today's headlines

Most



Automatic parallelism in modern machines

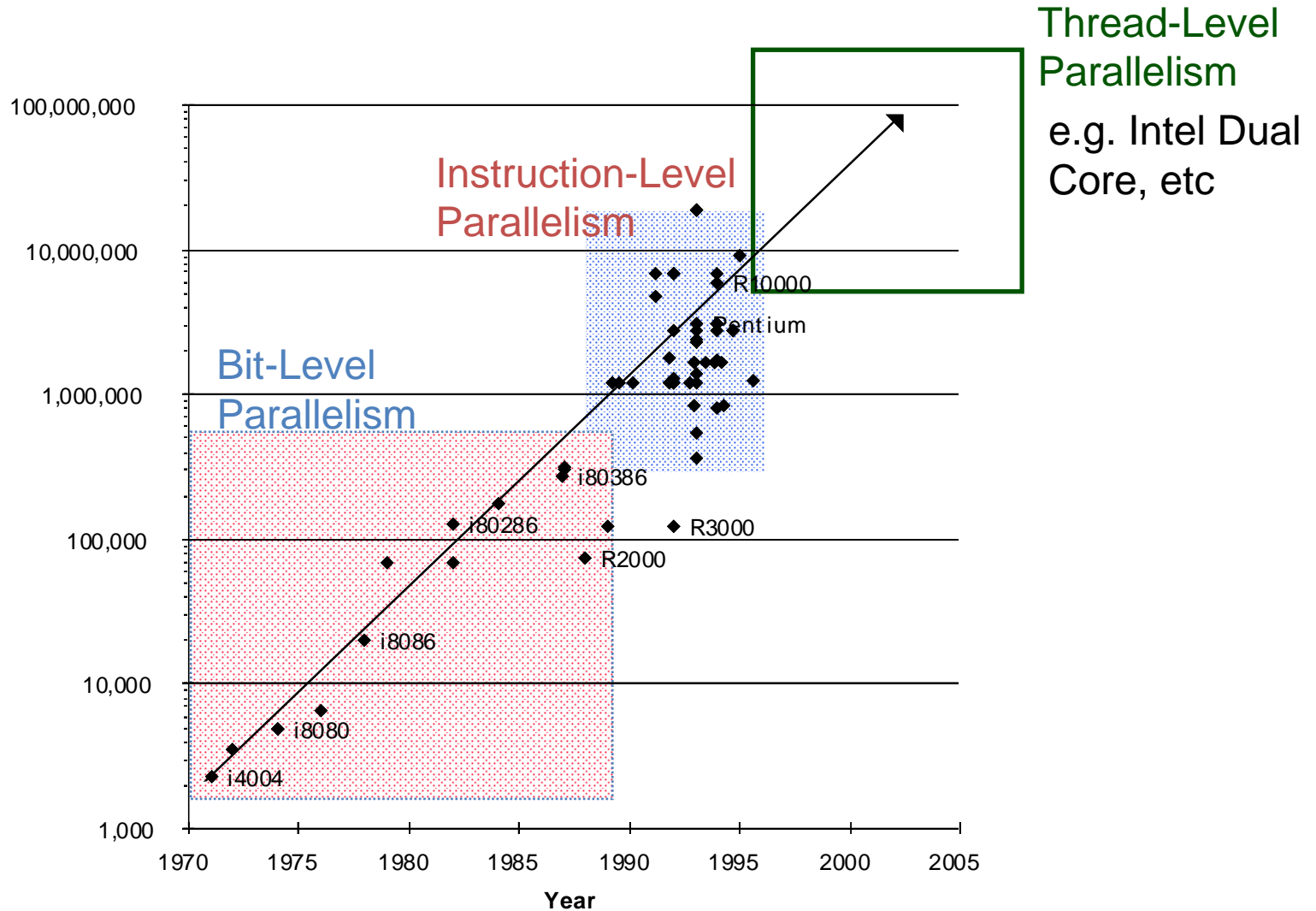
- **Bit level parallelism**: for floating point operations, etc.
- **Instruction level parallelism (ILP)**: execution of multiple instructions per clock cycle
- **Memory system parallelism**: overlap of memory operations with the computation
- **OS parallelism**: multiple tasks run in parallel (eg, threads)

There are obviously limits to these!

Hence, to achieve high performance, the programmer needs to
identify, schedule and coordinate
the **parallel tasks and data**
and

Become a Parallel Computing programmer!

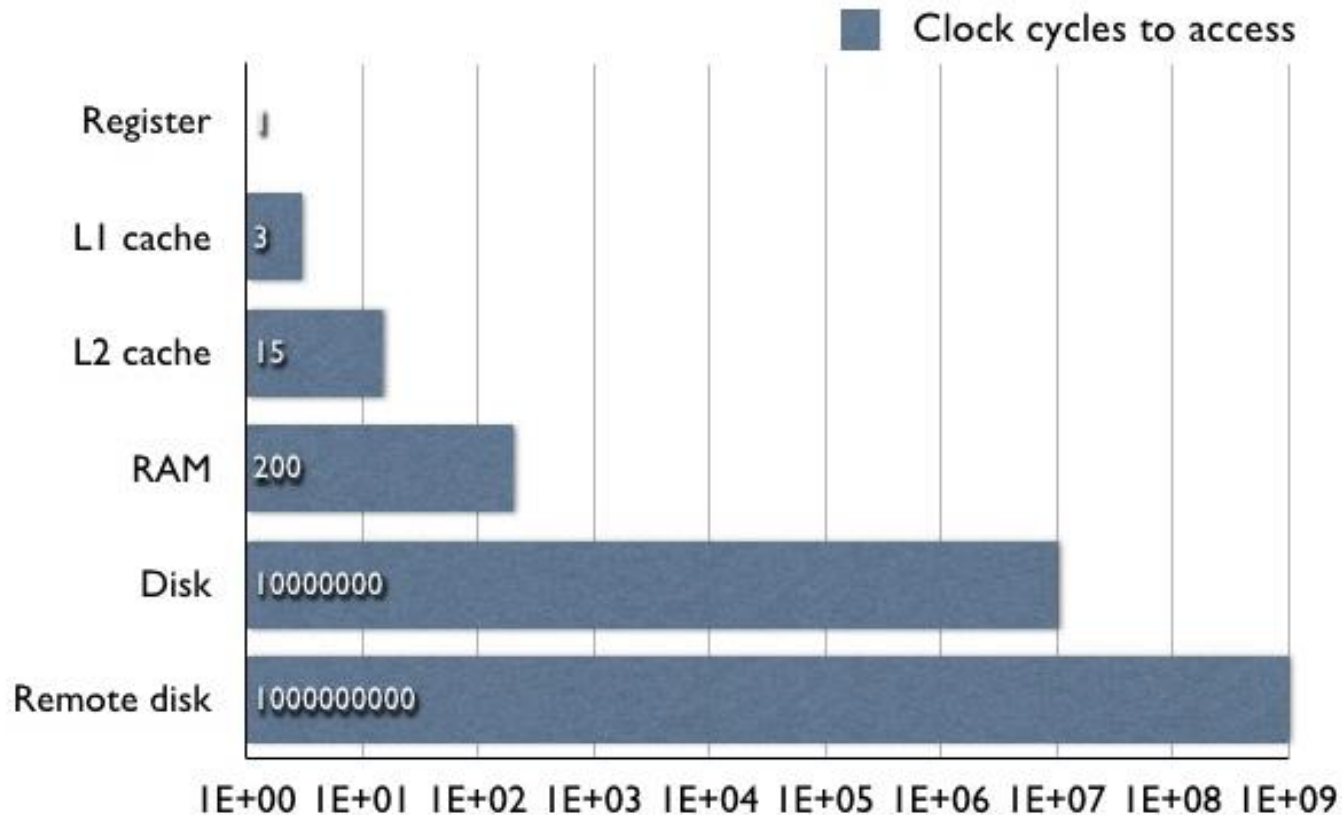
Microprocessor Transistors and Parallelism



More Exotic Solutions on the Horizon

- **GPUs - Graphics Processing Units (e.g., NVidia)**
 - Parallel processor attached to main processor
 - Originally special purpose, getting more general
- **FPGAs – Field Programmable Gate Arrays**
 - Inefficient use of chip area
 - More efficient than multicore now, maybe not later
 - Wire routing heuristics still troublesome
- **Dataflow and tiled processor architectures**
 - Have considerable experience with dataflow from 1980's
 - Are we ready to return to functional programming languages?
- **Cell**
 - Software controlled memory uses bandwidth efficiently
 - Programming model not yet mature
- **Quantum computing**
 - Use of quantum-mechanical phenomena
 - Qubits
 - Theoretical or physical computation

Memory Hierarchy



Processors vs. memory access

In 1986

Clock cycle \sim 120 nanoseconds

DRAM access time \sim 140 nanoseconds

» 1:1 ratio ☹

In 1996

Clock cycle \sim 4 nanoseconds

DRAM access time \sim 60 nanoseconds

» 20:1 ratio ☹☹

In 2002

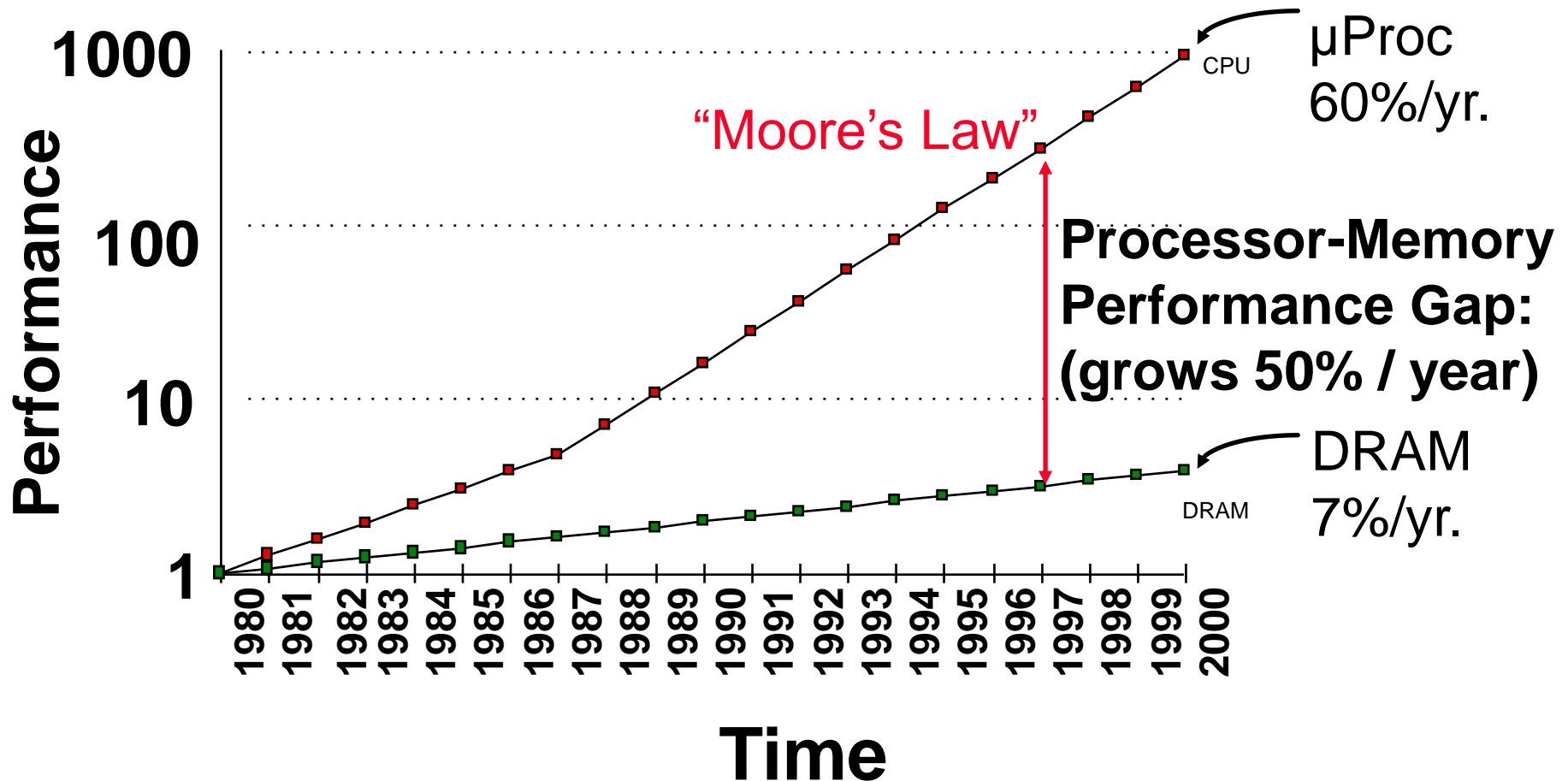
Clock cycle \sim 0.6 nanoseconds

DRAM access time \sim 50 nanoseconds

» 100::1 ratio ☹☹☹

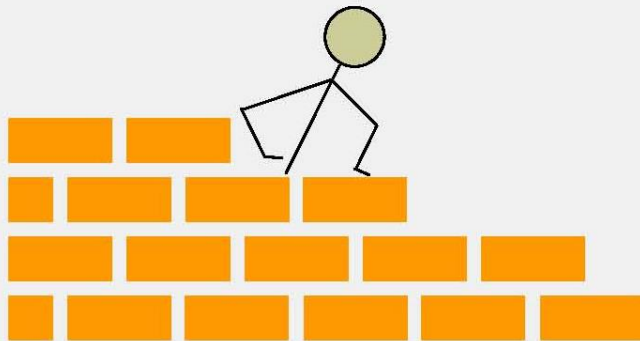
From 2003, in fact, first Dual Core processor(by Power PC, not Intel!)

Processor-DRAM Gap (latency)

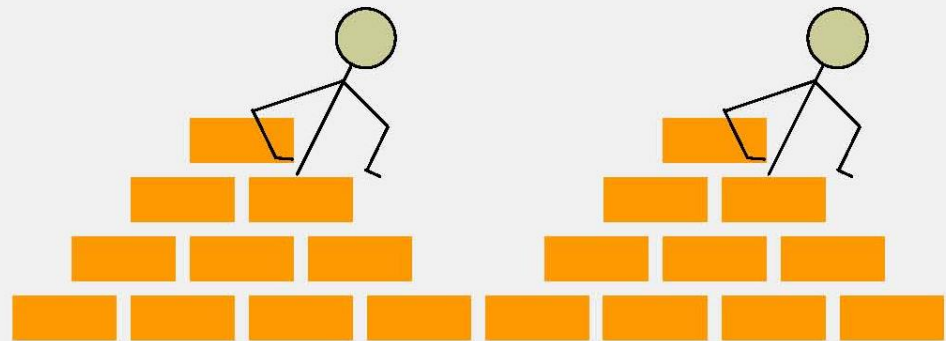


What is Parallel Computing?

- Parallel Processes
 - Divided a job in more tasks
 - Assign these task to many processes which work simultaneously
 - Coordinate, control and monitor these processes
- Everyday example
 - Wall construction



Sequential execution

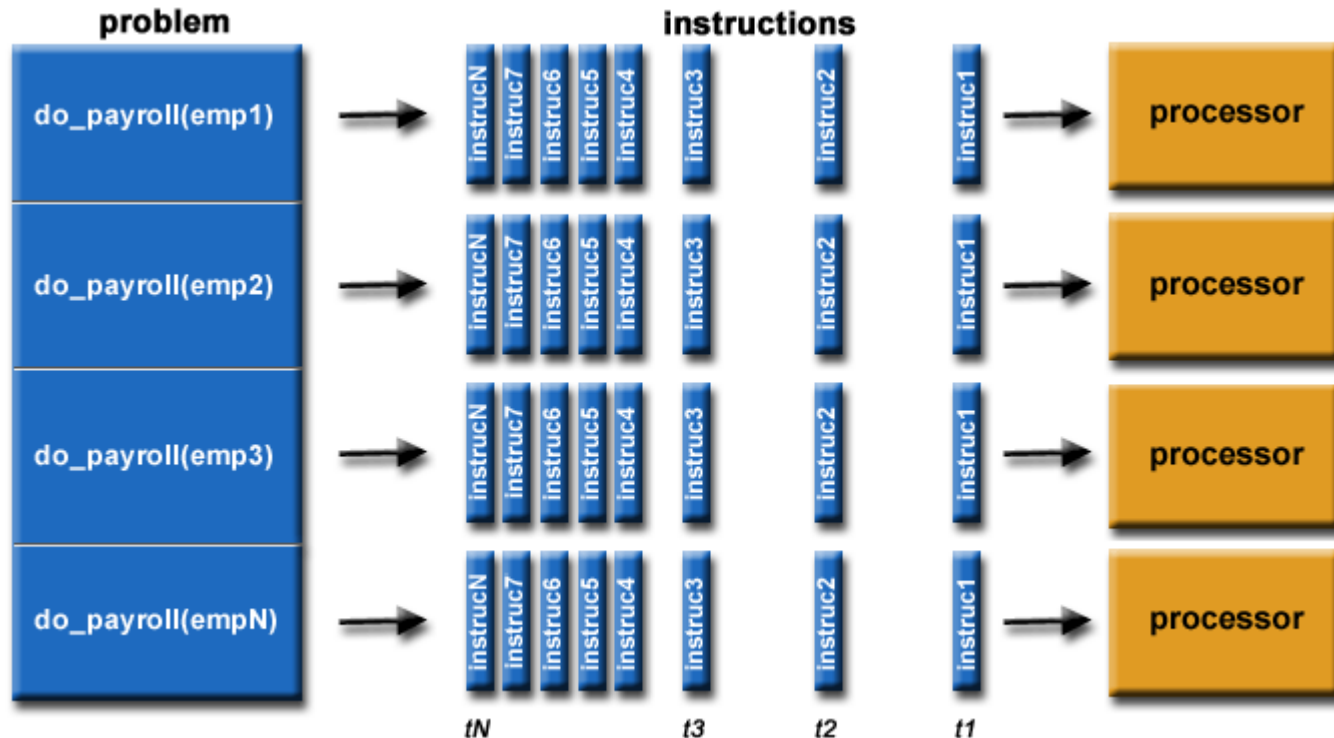


Parallel execution

Simplistic definition of Parallel Computing

In the more simple sense, Parallel Computing is the simultaneous use of multiple computation resources to solve a computational problem:

- 1 - Use of multiple CPUs
- 2 - A problem is broken into discrete parts that can be solved concurrently
- 3 - Each part is further broken down into a series of instructions
- 4 - The instructions of each part are performed simultaneously on different CPUs



What is Parallel Computing

- Unfortunately, it is not enough to give a "shovel" (program) to each "worker" (processor)!
- Steps in the parallelization of a job (in first approximation):
 1. Decide on the pattern of interconnection between the "processor" and "memory"
 2. Identify and implement system software for the hardware
 3. Identify data structures and algorithms for our problem
 4. Splitting of algorithms and data structures into sub-problems
 5. Identify "communications" that will be needed between the sub-problems
 6. Assign the sub-problems to "processors" and memory modules.

We will deal with points 3-6!

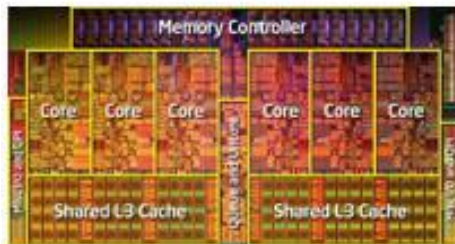
Supercomputers, CPUs, Nodes, Processors, Sockets

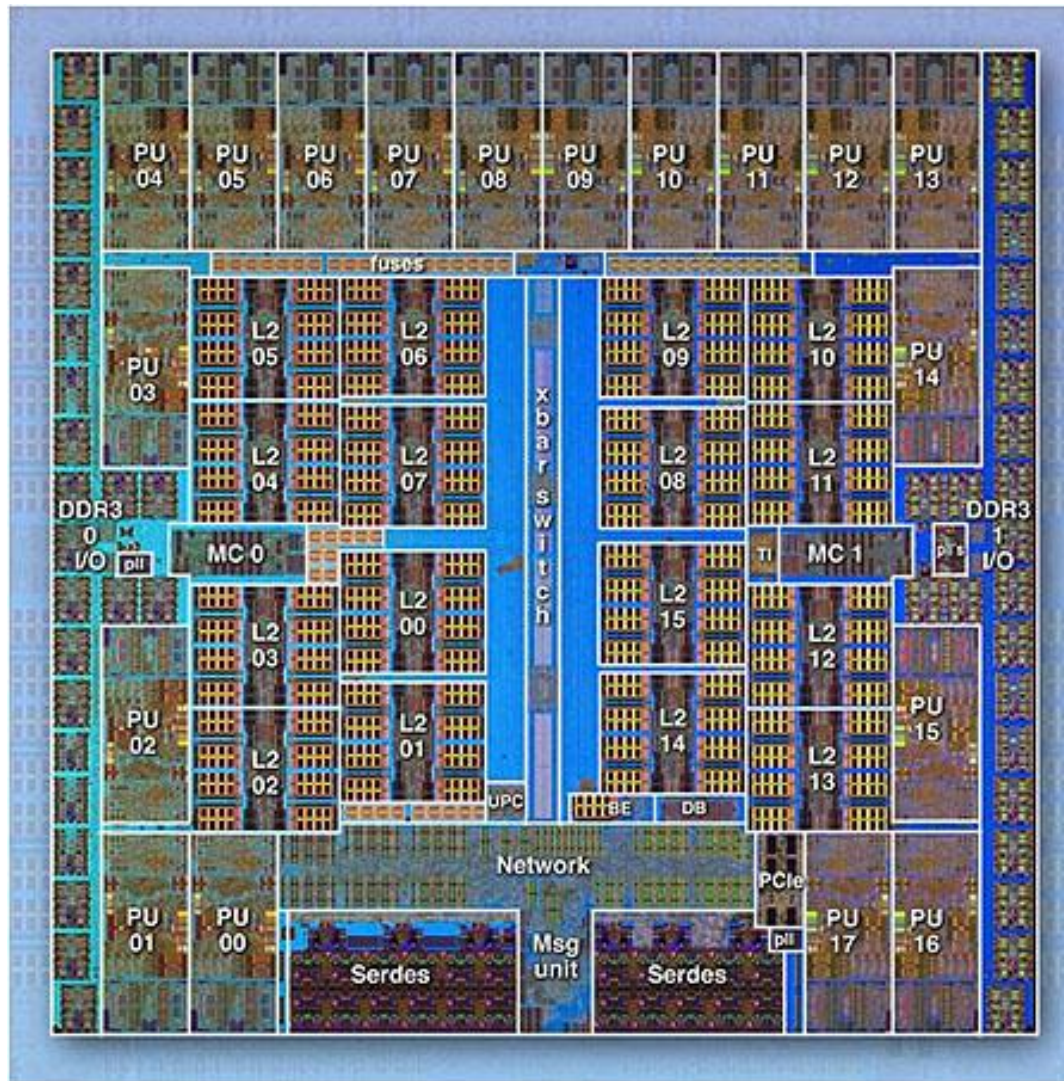


Supercomputer - each blue light is a node

Node - standalone
Von Neumann computer


CPU / Processor / Socket - each
has multiple cores / processors.





IBM BG/Q Compute Chip with 18 cores (PU) and 16 L2 Cache units (L2)

Scalability

A close-up portrait of an older man with glasses, wearing a suit and tie, looking slightly to the side with a serious expression.

WHAT ONE PROGRAMMER
CAN DO IN ONE MONTH, TWO
PROGRAMMERS CAN DO IN
~~TWO MONTHS.~~
HALF A MONTH

- Fred Brooks

atlaz.io

The main reasons for Parallel Computing

- Increase the size and complexity of problems that can be solved
 - Bigger problems may not be solvable on sequential computers in a reasonable time
 - Decompose in smaller problems (**weak scaling**)
 - Big problems may not be contained in a sequential computer's memory
 - Distribute the problem on many computers memories (**strong scaling**)
- Reduce the time to solve a problem
- Solve bigger problems, more quickly

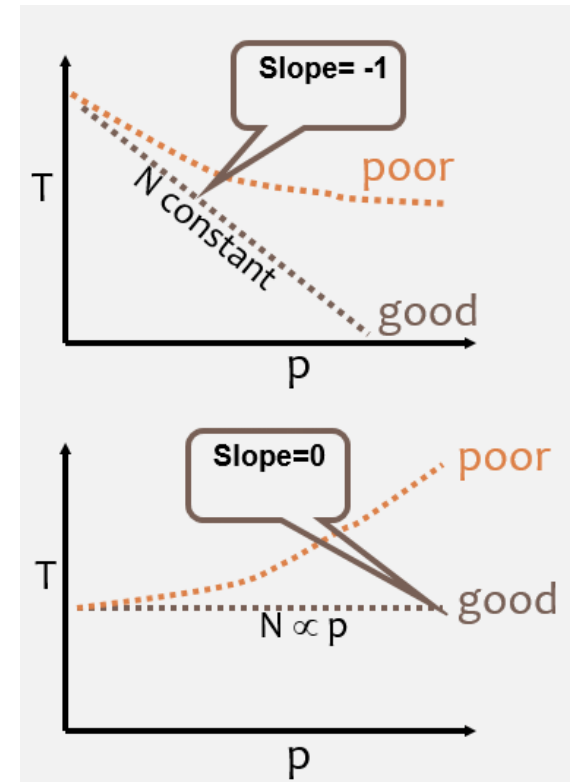
Scalability

•Strong scaling:

- The total problem size N stays fixed as more processors P are added.
- Goal is to run the same problem size faster
- Perfect scaling means problem is solved in $1/P$ time (compared to serial)

•Weak scaling:

- The problem size *per processor* stays fixed as more processors are added.
- Goal is to run larger problem in same amount of time
- Perfect scaling means problem $P \times N$ runs in same time as single processor run



Hardware factors play a significant role in scalability:

- Memory-cpu bus bandwidth on an SMP machine
- Communications network bandwidth
- Amount of memory available on any given machine or set of machines
- Processor clock speed

Why do fast (parallel) machines run slow?

Latency

Waiting for access to memory or other parts of the system

Overhead

Extra work that has to be done to manage program concurrency and parallel resources the real work you want to perform

Starvation

Not enough work to do due to insufficient parallelism or poor load balancing among distributed resources

Contention

Delays due to fighting over what task gets to use a shared resource next. Network bandwidth is a major constraint.

Supercomputers

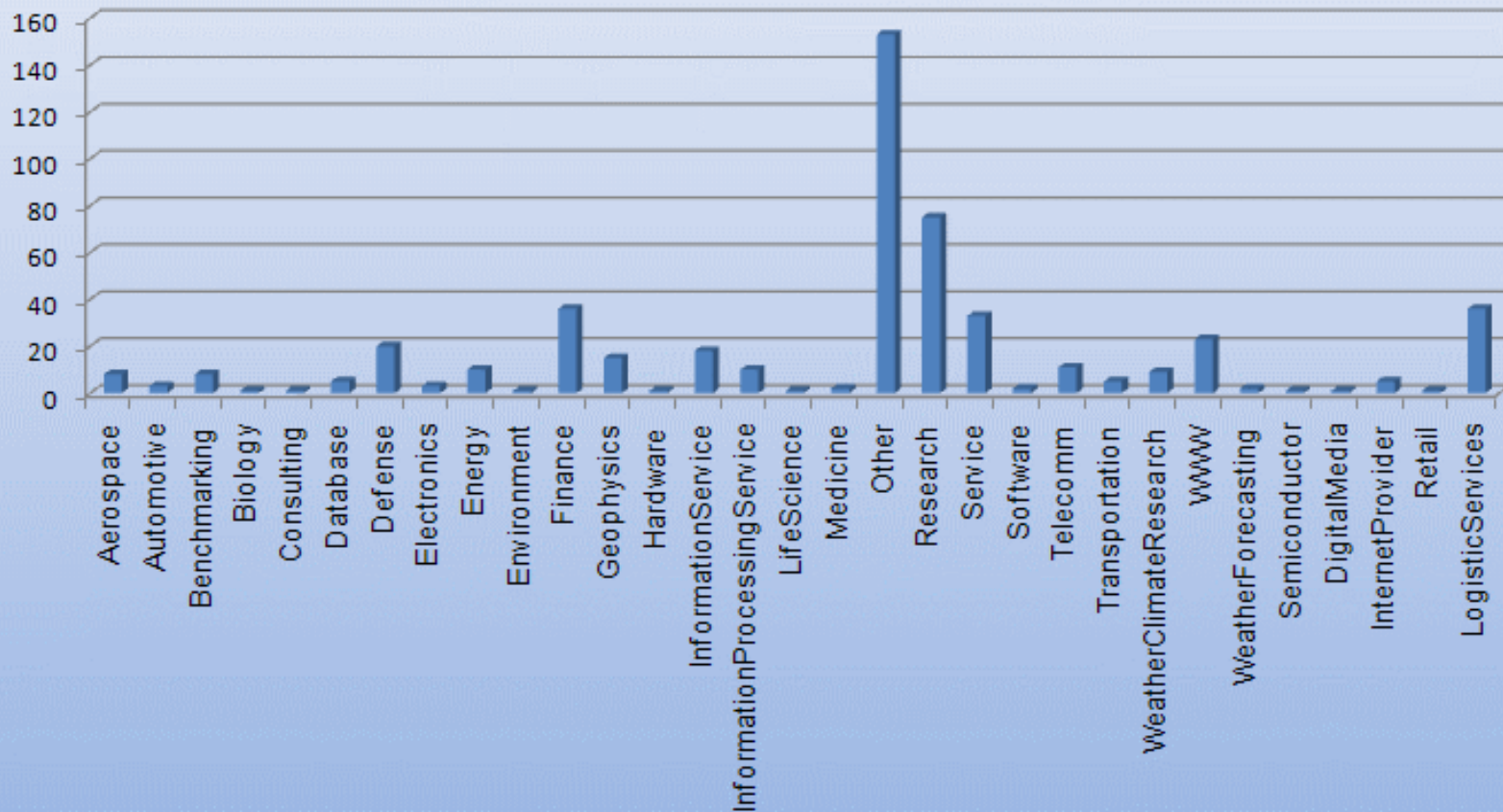
Check this out
www.top500.org

... after checking this ...

http://en.wikipedia.org/wiki/Teraflops_Research_Chip

Who uses parallel computing?

Top500 HPC Application Areas



What is a Benchmark?

- Comparison of different systems in solving "something hard" as "dense systems of linear equations"
- The comparison involves different computers
- Famous Benchmarks :
 - LINPACK: Linear equation resolver
 - Typical floating-point operations and use of data structures (vector/array)

Performance

- The measurement of the performance of a certain parallel machine makes sense for:
 - The magnitude of the “bigger” problems (ie the size of the problem that "make sense")
 - The performances reflect the largest problem that can be executed on a given machine
- **R_{\max}** – The performance in Tflops of the largest problem that runs on the machine
- **N_{\max}** – The size of the largest problem that runs on the machine
- **$N_{1/2}$** – the size where half R_{\max} is obtained
- **R_{peak}** – Theoretical peak performance (in Tflops) of the machine

TOP 7 – November 2020

Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	Supercomputer Fugaku - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442,010.0	537,212.0	29,899
2	Summit - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148,600.0	200,794.9	10,096
3	Sierra - IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM / NVIDIA / Mellanox DOE/NNSA/LLNL United States	1,572,480	94,640.0	125,712.0	7,438
4	Sunway TaihuLight - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway, NRCPC National Supercomputing Center in Wuxi China	10,649,600	93,014.6	125,435.9	15,371
5	Selene - NVIDIA DGX A100, AMD EPYC 7742 64C 2.25GHz, NVIDIA A100, Mellanox HDR Infiniband, Nvidia NVIDIA Corporation United States	555,520	63,460.0	79,215.0	2,646
6	Tianhe-2A - TH-IVB-FEP Cluster, Intel Xeon E5-2692v2 12C 2.2GHz, TH Express-2, Matrix-2000, NUDT National Super Computer Center in Guangzhou China	4,981,760	61,444.5	100,678.7	18,482
7	JUWELS Booster Module - Bull Sequana XH2000 , AMD EPYC 7402 24C 2.8GHz, NVIDIA A100, Mellanox HDR InfiniBand/ParTec ParaStation ClusterSuite, Atos Forschungszentrum Juelich (FZJ)	449,280	44,120.0	70,980.0	1,764

Terminology

Task

A discrete logic section of computational work. A task is typically a program or "piece" of the program that is executed by a processor

Parallel Task

A task that can be executed by multiple processors in a "safe" (i.e., provides correct results)

Serial Execution

Execution of a program sequentially, one instruction at a time. In the simplest sense, this is what happens using a machine with a processor. However, virtually all parallel tasks have sections of a parallel program to be executed in a serial fashion

Parallel Execution

Execution of a program consists of several tasks with each task that is "able" to carry out the same or different instructions at the same time

Terminology

Shared Memory

From a hardware point of view, it describes an architecture where each processor has direct access (usually via bus) to a common physical memory.

From a software point of view, it describes a model where parallel tasks have the same "picture" of memory, and can access and address the same logical memory locations, regardless of where it is located

Distributed Memory

From a hardware point of view, it refers to a memory access based on network, for a not in common physical memory .

From a software point of view, the task can only " see " from a logical point of view the local memory of the machine, and can use communications to access the memory of other machines where other tasks are running

Communications

Parallel tasks typically need to exchange data. There are several ways to achieve this, for example through a shared memory bus , or via a network. However , the exchange of data is reported as effective **communication** , regardless of the method used

Terminology

Synchronization

The coordination of parallel tasks in real-time, often associated with communications. It is often implemented by establishing a sync point in an application where a task can not continue until another task (or more) achieve the same (or logically equivalent) point. Synchronization usually involves the expectation of at least one task, and can therefore cause an increase in the "wall - clock" of parallel applications

Parallel Overhead

Amount of time taken to coordinate parallel tasks , in contrast to useful work. The determining factors are:

- 1.Task start-up time
- 2.Synchronizations
- 3.Data Communications
- 4.Software overhead imposed by parallel compilers , libraries , tools, operating systems, etc.
- 5.Task termination time

Granularity

In parallel computing, granularity is a qualitative measure of the relationship between computation and communication.

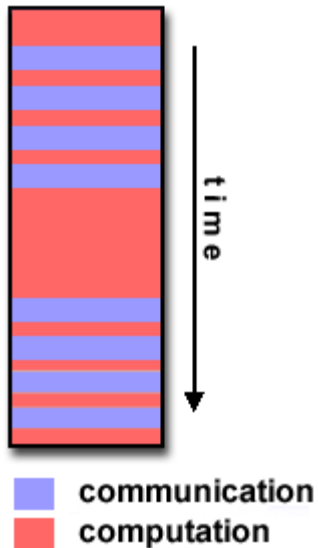
Coarse : relatively large portions of computational work are done between communication

Fine : relatively small portions of computational work are done between communication

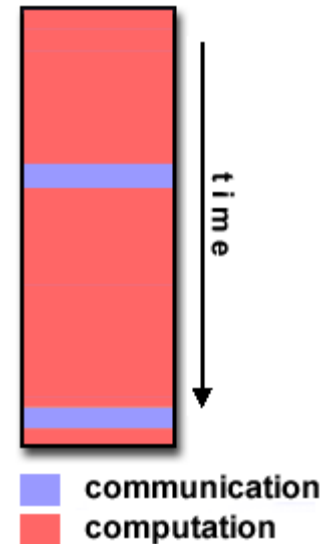
Granularity

- Computing / Communication Ratio

Fine-grain Parallelism :



Coarse-grain Parallelism :



Terminology

Massive Parallelism

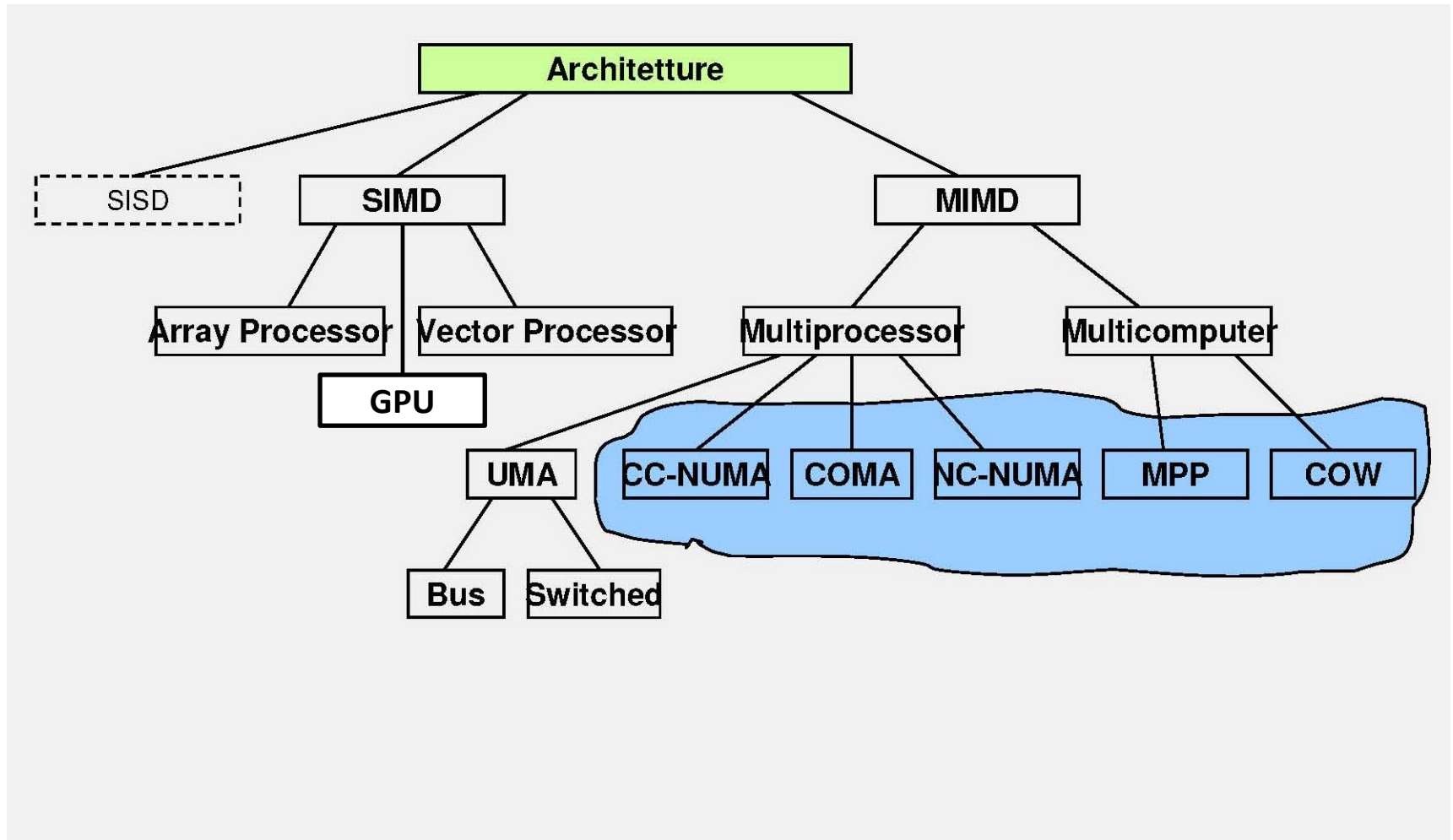
Refers to hardware that includes a given parallel system consisting of many physical processors. "Many" refers to date in systems with more than 10,000 processors (i.e., GPGPU)

Scalability

Refers to the ability of parallel systems (hardware and / or software) to allow a **proportional increase** in the speed-up with the increase of processors. Factors that contribute to scalability include:

- Hardware - especially the memory bandwidth-cpu bandwidths and network communications
- The used algorithm!
- The parallel overhead
- Characteristics of your specific application and coding

Overview of parallel architectures

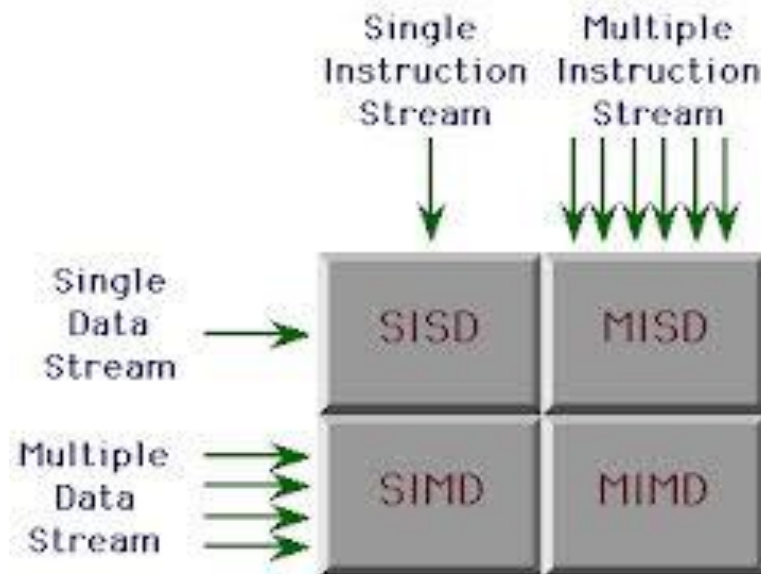


Classification of Parallel Computers

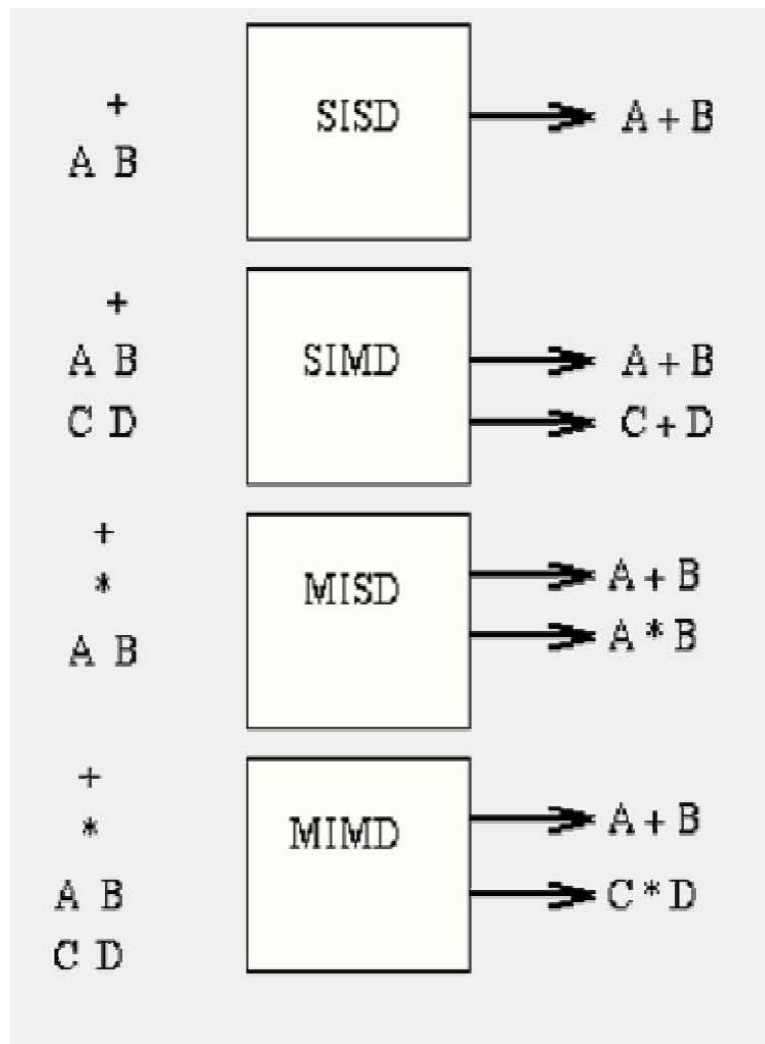
- Purely functional, by Flynn:
 - SIMD, SIMD, MISD, MIMD
- Structural
 - Memory based
 - Shared memory, distributed memory
 - Based on the interconnection network
 - Bus, crossbar, direct and indirect networks

Flynn's Classification

- **Michael Flynn** introduced in 1966 the following **taxonomy**, where “interesting” parallel computer classes are **MIMD** (Multiple Instruction stream-Multiple Data Stream) and **SIMD** (Single Instruction stream-Multiple Data Stream)

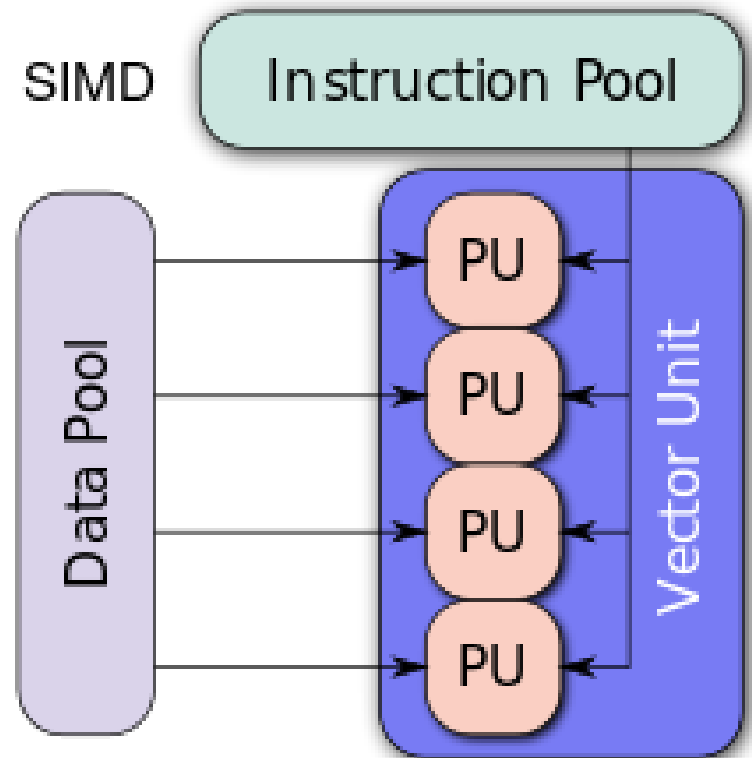


Potentialities of the 4 models



SIMD

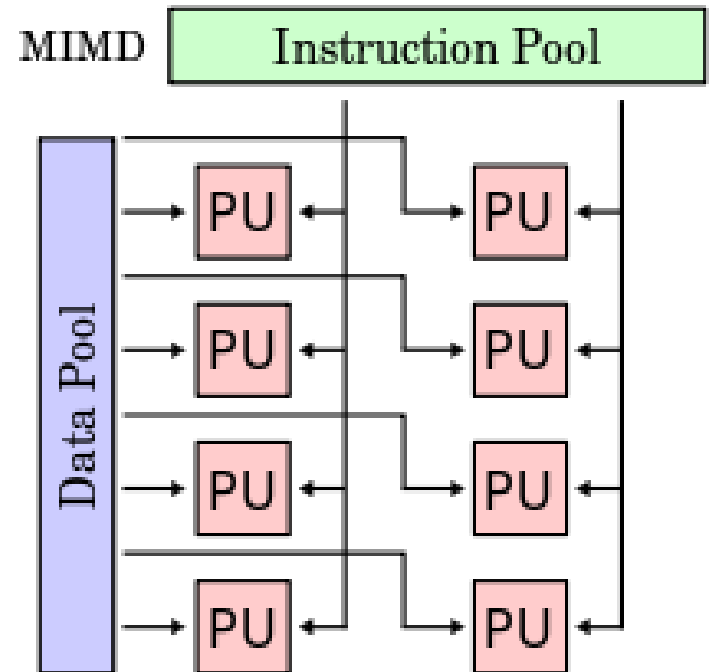
- **Synchronous operations**
 - At each clock step all processors execute the same instruction on different data
- **Array processors** (Connection machine, MasPAR MP, etc)
- Pipeline vector computer (Cray 1 & 2)
- **SIMD machines** are suitable for solving **data-parallel problems**, where the same instruction is applied to distinct partitions:
 - Ex: Matrix sum: $A + B = C$
 - If $A[2][2]$ and $B[2][2]$ are distributed to 4 processors, in parallel we have:
 - $A_{11} + B_{11} = C_{11}$ $A_{12} + B_{12} = C_{12}$
 - $A_{21} + B_{21} = C_{21}$ $A_{22} + B_{22} = C_{22}$



MIMD

Multiprocessor, Multicomputer

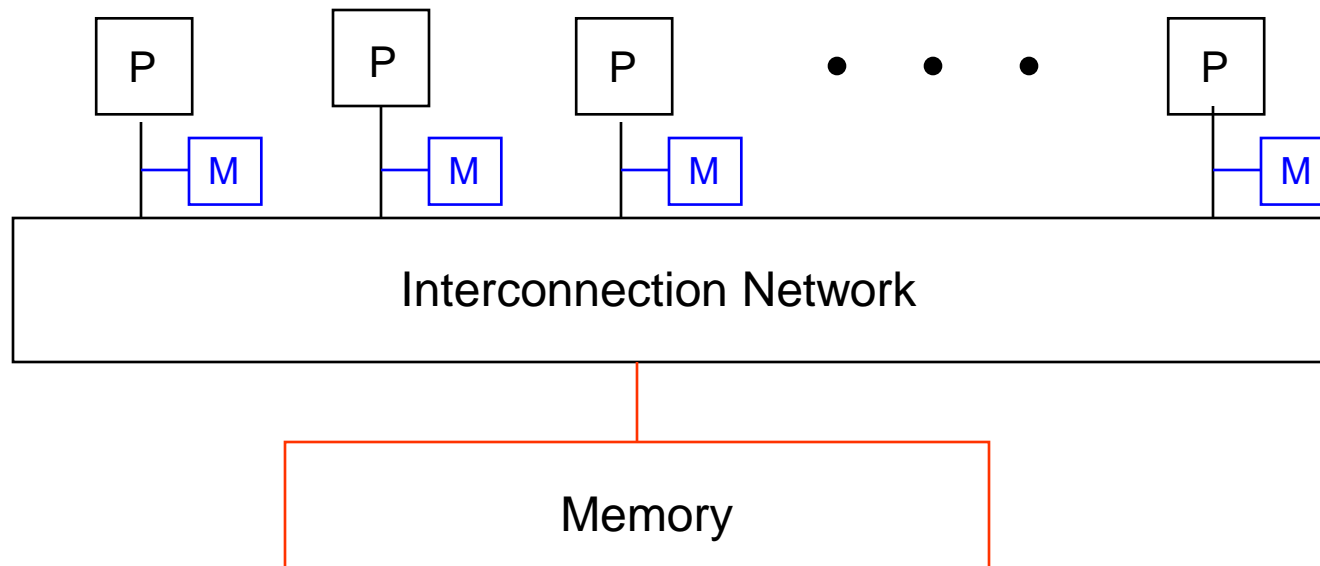
- Each processor operates in **asynchronous** mode under the program control and on different data
- **Shared memory MIMD computers** are known as **multiprocessor** or tightly coupled machines (ex: all current multi-core computers, etc)
- **Network based MIMD computers** are known as **multicomputer** or loosely coupled machines (es: Transputers, CRAY T3E, Beowulf clusters, etc)



Structural classification based on memory

- In a parallel system, **2 typical memory configurations** are possible:
- **Shared-memory**
 1. A single memory is present, where all **processing elements (PE)** can access
 2. Message exchanges between PEs occurs by writes in this memory
 3. If the number of PEs is high, memory can become a **bottleneck**
- **Distributed memory**
 1. Each PE has its local memory
 2. Communication occur by means of message exchanges (**Message Passing**)

Generic Parallel Architecture



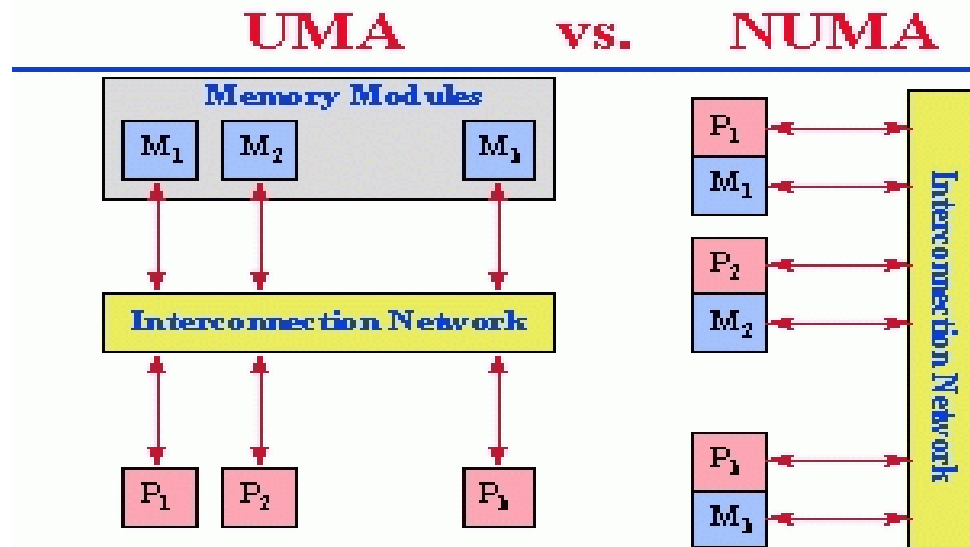
Where is memory physically located?

Shared memory vs Message passing

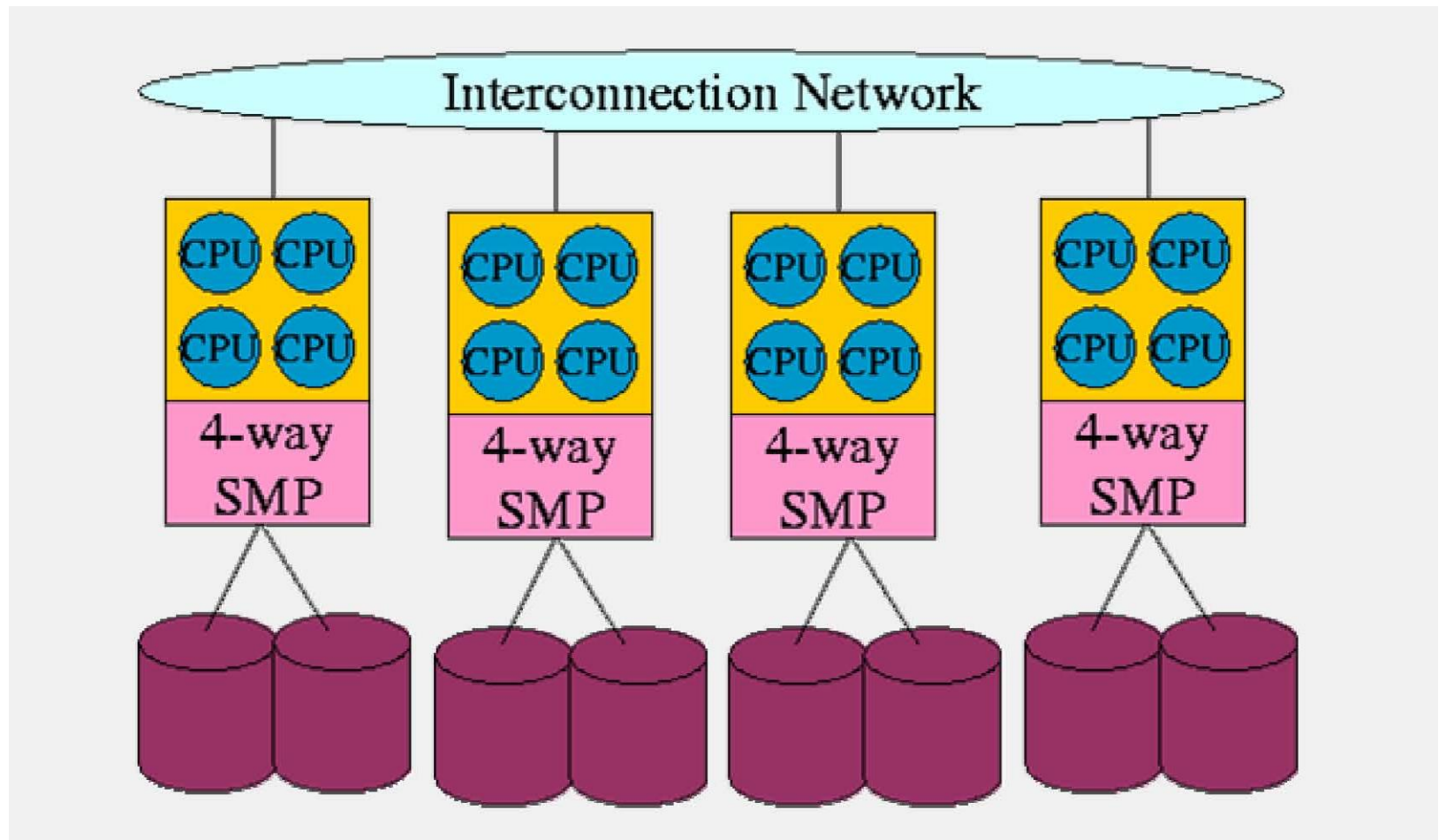
- In both SIMD and MIMD machines, **cooperation** is achieved by means of **data exchange**
- This can occur by:
 - Using **Shared memory (SM)** and shared variables
 - Using **Distributed memory (DM)** and message passing
- In the case of **SM**, we have a **single address space**, where all processors can access data
 - Problems linked to scalability, partially solved by cache usage
 - Cooperation suitable only for few processors (2-32 MIMD multiprocessors)
- In the case of **DM**, we have a **private addressing to each processor**, so message passing is necessary for communication
 - Typical modality for many MIMD computers and workstation based clusters
 - If necessary, a single address space can be **emulated** by a suitable software (e.g. Virtual Shared Memory)

Shared Memory Multiprocessors

- Load/Store operations access directly all memory locations of the system memory
 - Simple mechanism for communication and sharing
 - Transparent respect to location
 - Same operations available from uni-processors
- Central memory or distributed «near» processors
- Cost of Uniform Memory Access (UMA) or Non Uniform Memory Access (NUMA)



Typical Workstation Cluster



Structural classification on interconnection basis

- Typical interconnection network include:
 - Single Shared bus
 - Multiple bus
 - Crossbar
 - Ring
 - Hypercube
 - tree

Programming languages adopted in High Performance Computing

- **Message Passing Interface (MPI)**
 - Explicit sharing of variables from one processor to another, so that each process can manage a memory segment of the program
- **OpenMP**
 - Manages the distribution of process and data in SMP machines, communications occur by means of shared memory
- **High Performance Fortran**
 - Creates a single virtual shared memory segment between processors: the programmer chooses how to distribute data in memory
- **CUDA/ OpenCL**
 - De-facto programming languages for GPGPU based systems