

Computação Paralela

Mest. Int. Engenharia Computacional

Ano letivo 2019/2020

Manuel Barroso, Nuno Lau

Computação Paralela (2019/2020)

Escolaridade:

1h TP / semana 2h P / semana

Docentes (aulas TP e P):

Manuel Barroso Nuno Lau

scpip@ua.pt nunolau@ua.pt

Física (13.2.16) IEETA (IRIS Lab / 2.07)

Página web em elearning.ua.pt

login: utilizador universal

Slides adaptados da bibliografia

- Identificar e classificar os diferentes modelos e os diferentes níveis da programação paralela.
- Reconhecer programas suscetíveis de ser paralelizados e escolher a melhor abordagem.
- Desenvolver, depurar e otimizar um programa paralelo.
- Utilizar eficazmente clusters computacionais.

Programa (aulas teorico-práticas)

1. Computação paralela de alto desempenho.
2. Modelos de programação paralela.
3. OpenMP.
4. CUDA.
5. MPI.
6. Otimização.

As aulas práticas seguem uma filosofia do *saber fazer* e visam a realização de pequenos trabalhos sobre os diferentes assuntos.

- Paralelização com threads
- Extensões multimédia de processadores
- OpenMP
- CUDA
- MPI

- **Componente TeoricoPrática**
 - Teste final (durante época de exames)
 - Nota mínima: 8
- **Componente Prática**
 - 2 trabalhos
 - 50%.TP1 + 50%.TP2
 - Nota mínima: 8
- **Nota final**
 - $NF = 40\%.CTP + 60\%.CP$

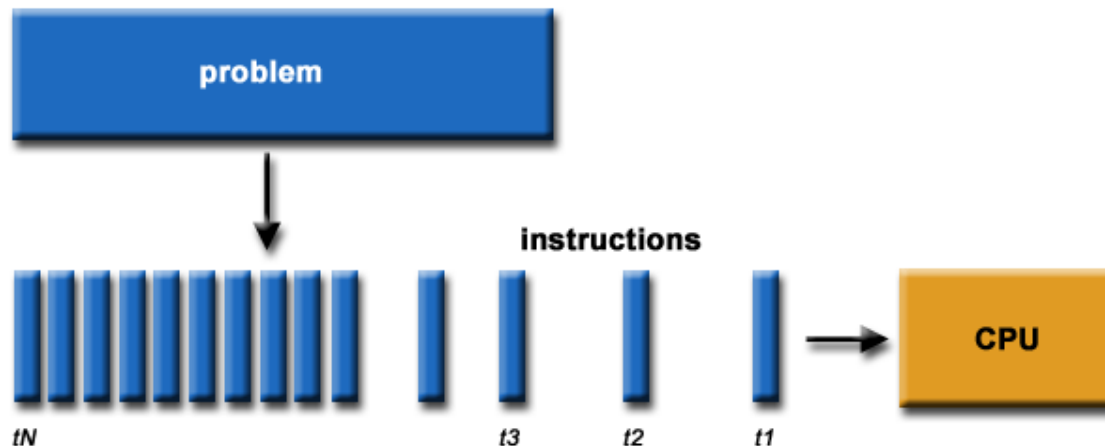
- Os trabalhos devem ser realizados individualmente
- Efetuar uma implementação precisa dos problemas propostos
- Os trabalhos são desenvolvidos também fora das aulas práticas
- **O plágio será fortemente penalizado**

- Rauber and G. Rünger, Parallel Programming: for Multicore and Cluster Systems, 2nd edition, Springer, 2013.
- Hager and G. Wellein, Introduction to High Performance Computing for Scientists and Engineers, CRC Press, 2010.
- J. Quinn, Parallel Programming: in C with MPI and OpenMP, McGraw-Hill, 2003.
- S. Pacheco, Parallel Programming with MPI, Morgan Kaufmann, 1997.
- Cheng, M. Grossman, and T. McKercher, Professional CUDA C Programming, Wrox, 2014.

Questões?

What is Parallel Computing? (1)

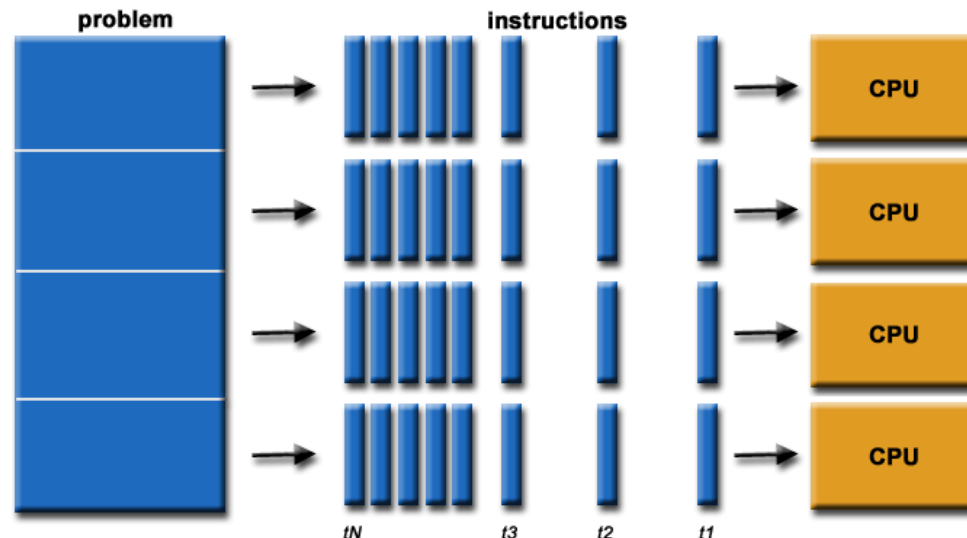
- Traditionally, software has been written for **serial** computation:
 - To be run on a single computer having a single Central Processing Unit (CPU);
 - A problem is broken into a discrete series of instructions.
 - Instructions are executed one after another.
 - Only one instruction may execute at any moment in time.



Acknowledgement: This lecture is based on slides by Florent Nolot,

What is Parallel Computing? (2)

- In the simplest sense, **parallel computing** is the simultaneous use of multiple compute resources to solve a computational problem.
 - To be run using multiple CPUs
 - A problem is broken into discrete parts that can be solved concurrently
 - Each part is further broken down to a series of instructions
- Instructions from each part execute simultaneously on different CPUs



- The compute resources can include:
 - A single computer with multiple processors
 - A single computer with (multiple) processor(s) and some specialized computer resources (GPU, FPGA ...)
 - An arbitrary number of computers connected by a network
 - A combination of both

- The computational problem usually demonstrates characteristics such as the ability to be:
 - Broken apart into discrete pieces of work that can be solved simultaneously
 - Execute multiple program instructions at any moment in time
 - Solved in less time with multiple compute resources than with a single compute resource

Parallel Computing: what for? (1)

- Parallel computing is an evolution of serial computing that attempts to emulate what has always been the state of affairs in the natural world: many complex, interrelated events happening at the same time, yet within a sequence.
- Some examples:
 - Planetary and galactic orbits
 - Weather and ocean patterns
 - Tectonic plate drift
 - Rush hour traffic in Paris
 - Automobile assembly line
 - Daily operations within a business
 - Building a shopping mall
 - Ordering a hamburger at the drive through.

Parallel Computing: what for? (2)

- Traditionally, parallel computing has been considered to be "the high end of computing" and has been motivated by numerical simulations of complex systems and "Grand Challenge Problems" such as:
 - weather and climate
 - chemical and nuclear reactions
 - biological, human genome
 - geological, seismic activity
 - mechanical devices - from prosthetics to spacecraft
 - electronic circuits
 - manufacturing processes

Parallel Computing: what for? (3)

- Today, commercial applications are providing an equal or greater driving force in the development of faster computers. These applications require the processing of large amounts of data in sophisticated ways. Example applications include:
 - parallel databases
 - data mining
 - Machine learning
 - oil exploration
 - web search engines, web based business services
 - computer-aided diagnosis in medicine
 - management of national and multi-national corporations
 - advanced graphics and virtual reality, particularly in the entertainment industry
 - networked video and multi-media technologies
 - collaborative work environments
- Ultimately, parallel computing is an attempt to maximize the infinite but seemingly scarce commodity called time.

Why Parallel Computing? (1)

- This is a legitime question!
- Parallel computing is complex on any aspect!
- The **primary reasons** for using parallel computing:
 - **Save time** - wall clock time
 - **Solve larger problems**
 - **Provide concurrency** (do multiple things at the same time)

Why Parallel Computing? (2)

- Other reasons might include:
 - **Taking advantage of non-local resources** - using available compute resources on a wide area network, or even the Internet when local compute resources are scarce.
 - **Cost savings** - using multiple "cheap" computing resources instead of paying for time on a supercomputer.
 - **Overcoming memory constraints** - single computers have very finite memory resources. For large problems, using the memories of multiple computers may overcome this obstacle.

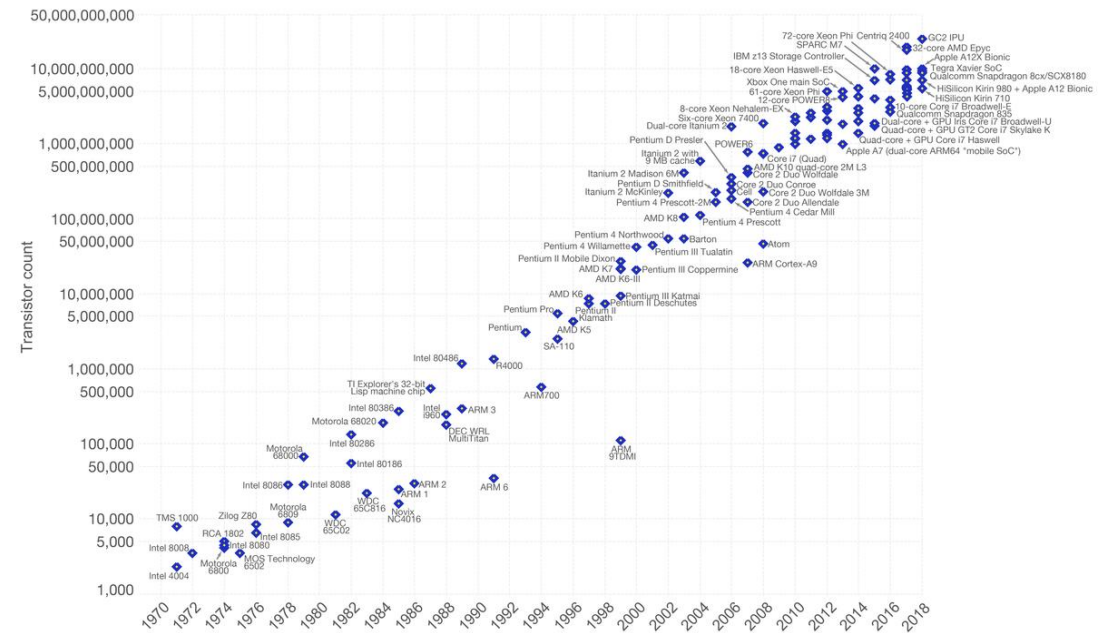
Limitations of Serial Computing

- **Limits to serial computing** - both physical and practical reasons pose significant constraints to simply building ever faster serial computers.
- **Transmission speeds** - the speed of a serial computer is directly dependent upon how fast data can move through hardware. Absolute limits are the speed of light (30 cm/nanosecond) and the transmission limit of copper wire (9 cm/nanosecond). Increasing speeds necessitate increasing proximity of processing elements.
- **Limits to miniaturization** - processor technology is allowing an increasing number of transistors to be placed on a chip. However, even with molecular or atomic-level components, a limit will be reached on how small components can be.
- **Economic limitations** - it is increasingly expensive to make a single processor faster. Using a larger number of moderately fast commodity processors to achieve the same (or better) performance is less expensive.

- The number of transistors on integrated circuits doubles approx every 2 years
- Chip performance double every 18-24 months
- Power consumption is proportional to frequency
- Clock speed saturates at 3 to 4 GHz. (End of free lunch. Future is parallel.)

Moore's Law – The number of transistors on integrated circuit chips (1971-2018)

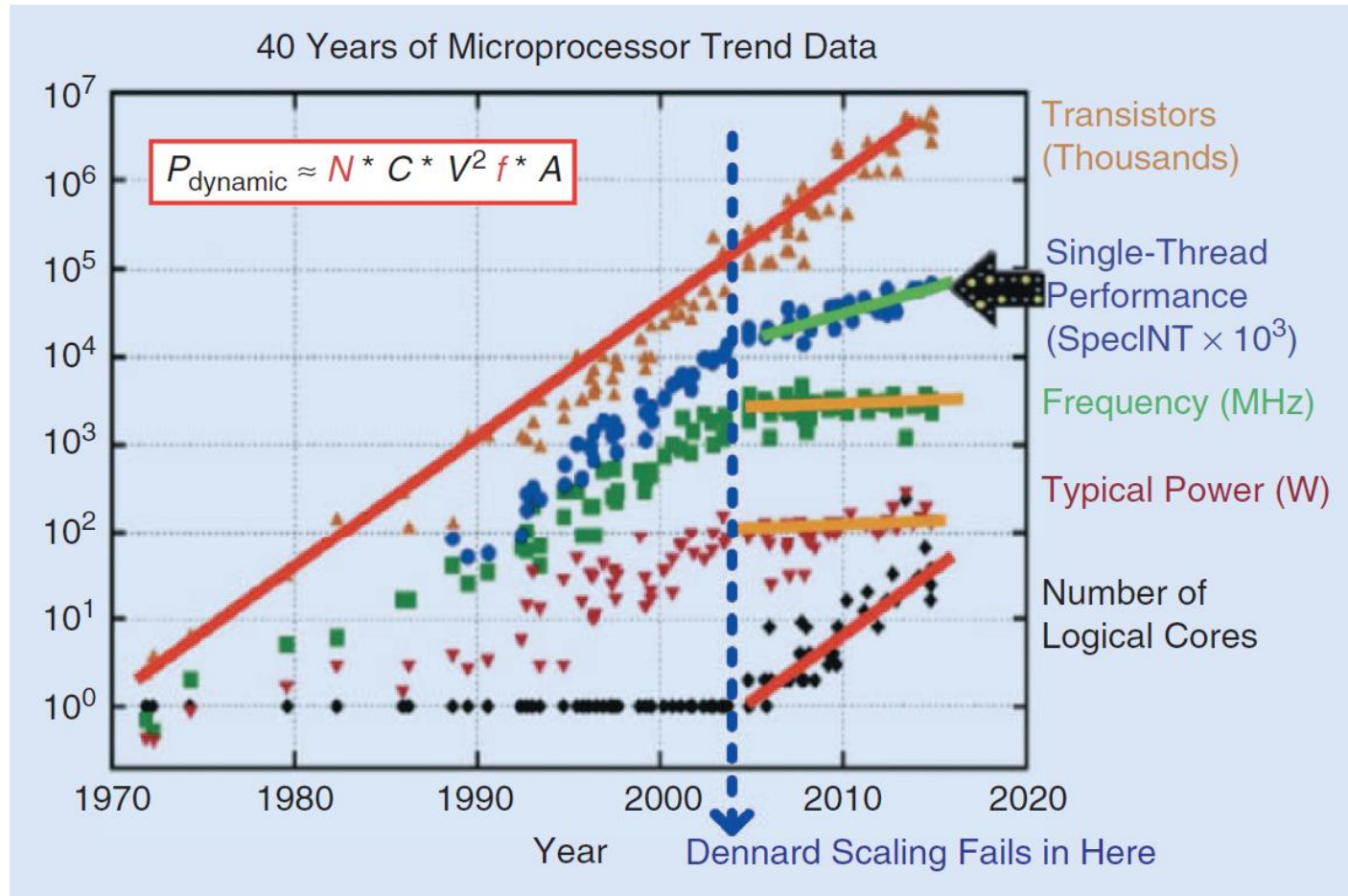
Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important as other aspects of technological progress – such as processing speed or the price of electronic products – are linked to Moore's law.



Data source: Wikipedia (https://en.wikipedia.org/wiki/Transistor_count)
The data visualization is available at [OurWorldinData.org](https://ourworldindata.org). There you find more visualizations and research on this topic.

Licensed under [CC-BY-SA](#) by the author Max Roser.

Moore's Law



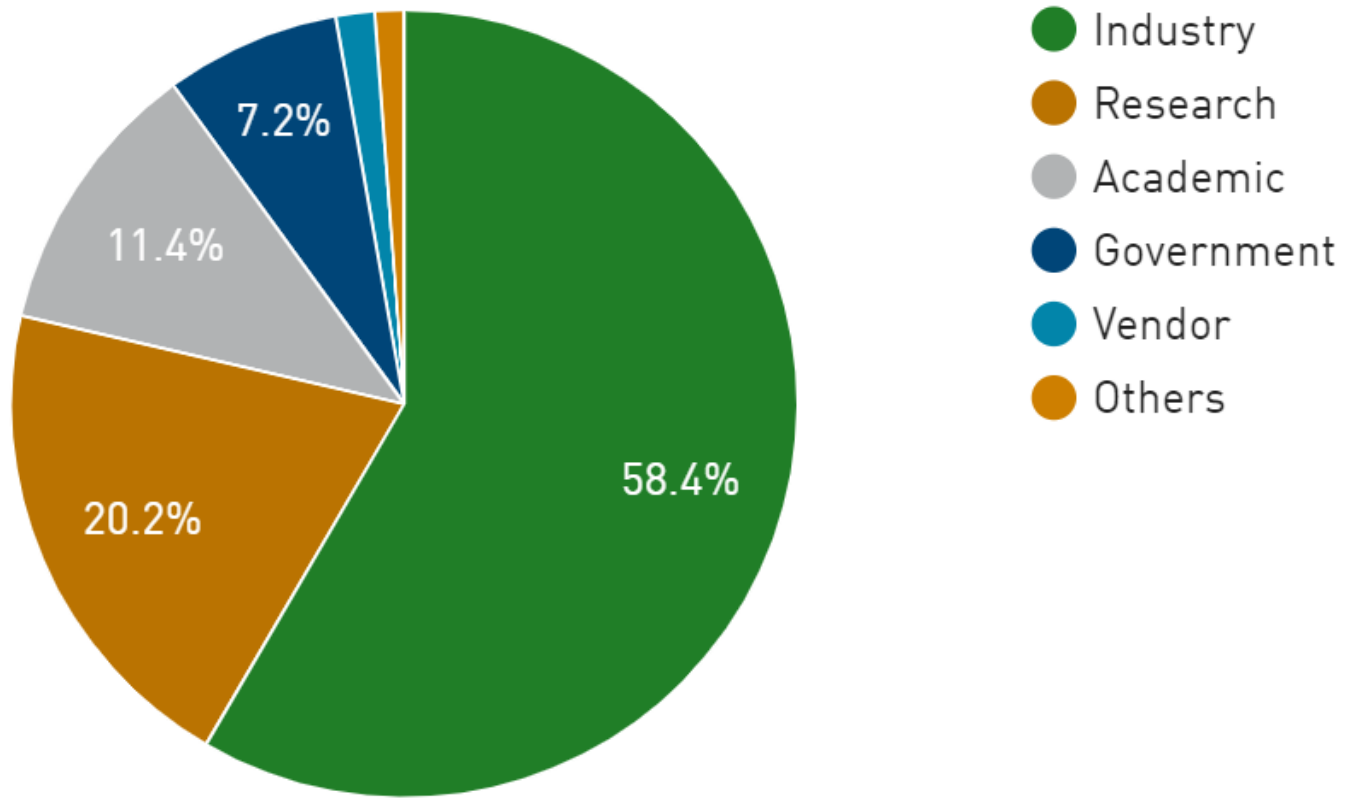
- during the past 10 years, the trends indicated by ever faster networks, distributed systems, and multi-processor computer architectures (even at the desktop level) clearly show that ***parallelism is the future of computing.***
- It will be multi-forms, mixing general purpose solutions (your PC...) and very speciliazed solutions as GPGPU from Nvidia, Tensor Processing Units from Google, Vision Processing Unit ...

Who and What? (1)

- Top500.org provides statistics on parallel computing users - the charts below are just a sample. Some things to note:
 - Sectors may overlap - for example, research may be classified research. Respondents have to choose between the two.

Who and What? (2)

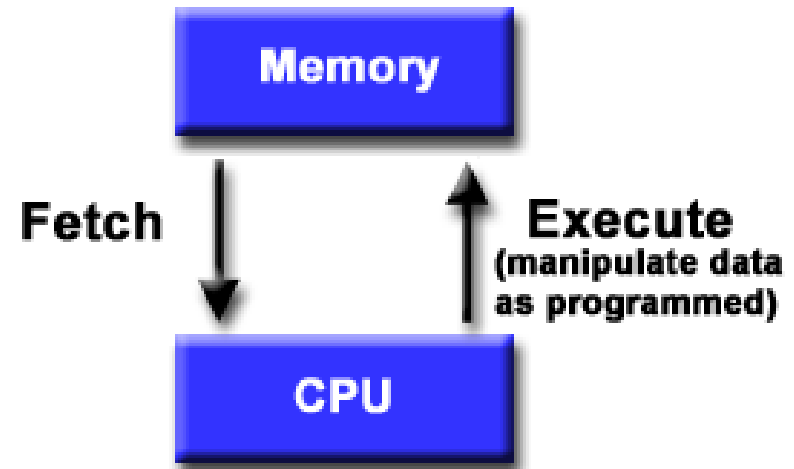
Segments System Share



Concepts and Terminology

- For over 40 years, virtually all computers have followed a common machine model known as the **von Neumann computer**. Named after the Hungarian mathematician John von Neumann.
- A von Neumann computer uses the **stored-program concept**. The CPU executes a stored program that specifies a sequence of read and write operations on the memory.

- Basic design
 - Memory is used to store both program and data instructions
 - Program instructions are coded data which tell the computer to do something
 - Data is simply information to be used by the program
- A central processing unit (CPU) gets instructions and/or data from memory, decodes the instructions and then ***sequentially*** performs them.



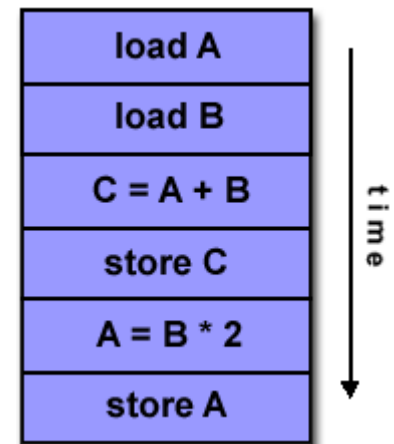
- There are different ways to classify parallel computers. One of the more widely used classifications, in use since 1966, is called **Flynn's Taxonomy**.
- Flynn's taxonomy distinguishes multi-processor computer architectures according to how they can be classified along the two independent dimensions of ***Instruction*** and ***Data***. Each of these dimensions can have only one of two possible states: ***Single*** or ***Multiple***.

- The matrix below defines the 4 possible classifications according to Flynn

S I S D Single Instruction, Single Data	S I M D Single Instruction, Multiple Data
M I S D Multiple Instruction, Single Data	M I M D Multiple Instruction, Multiple Data

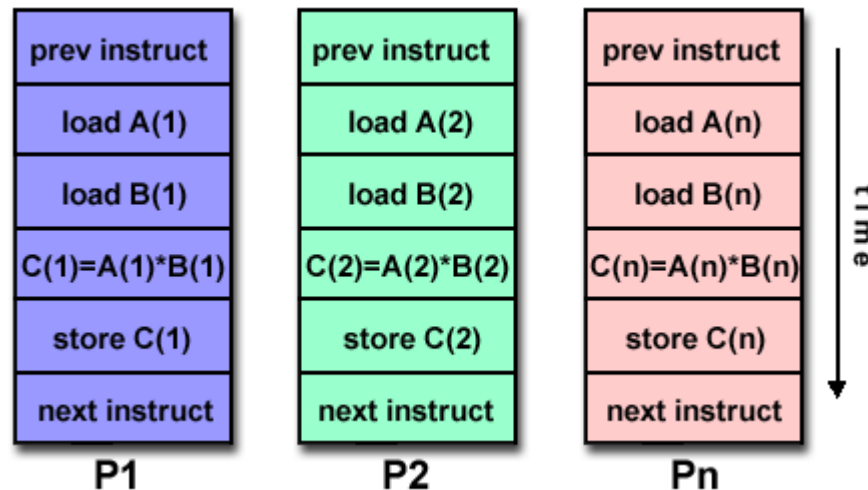
Single Instruction, Single Data (SISD)

- **A serial (non-parallel) computer**
- Single instruction: only one instruction stream is being acted on by the CPU during any one clock cycle
- Single data: only one data stream is being used as input during any one clock cycle
- Deterministic execution
- This is the oldest and until recently, the most prevalent form of computer
- Examples: most PCs, single CPU workstations and mainframes



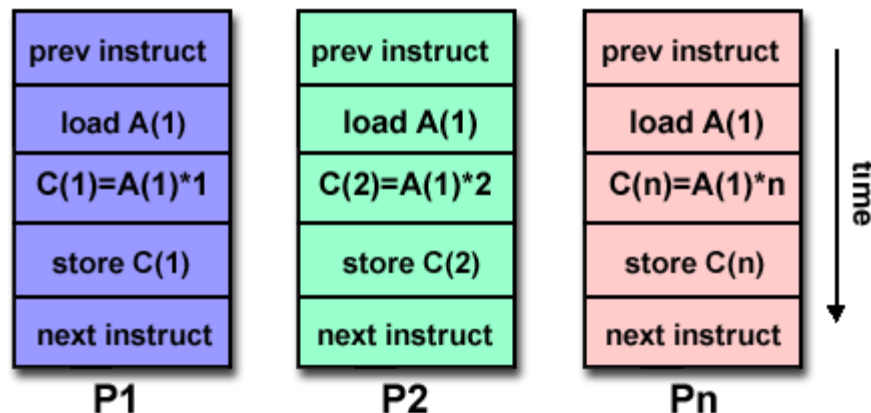
Single Instruction, Multiple Data (SIMD)

- **A type of parallel computer**
- Single instruction: All processing units execute the same instruction at any given clock cycle
- Multiple data: Each processing unit can operate on a different data element
- This type of machine typically has an instruction dispatcher, a very high-bandwidth internal network, and a very large array of very small-capacity instruction units.
- Best suited for specialized problems characterized by a high degree of regularity, such as image processing.
- Synchronous (lockstep) and deterministic execution
- Two varieties: Processor Arrays and Vector Pipelines
- Examples:
 - Processor Arrays: Connection Machine CM-2, Maspar MP-1, MP-2
 - Vector Pipelines: IBM 9000, Cray C90, Fujitsu VP, NEC SX-2, Hitachi S820



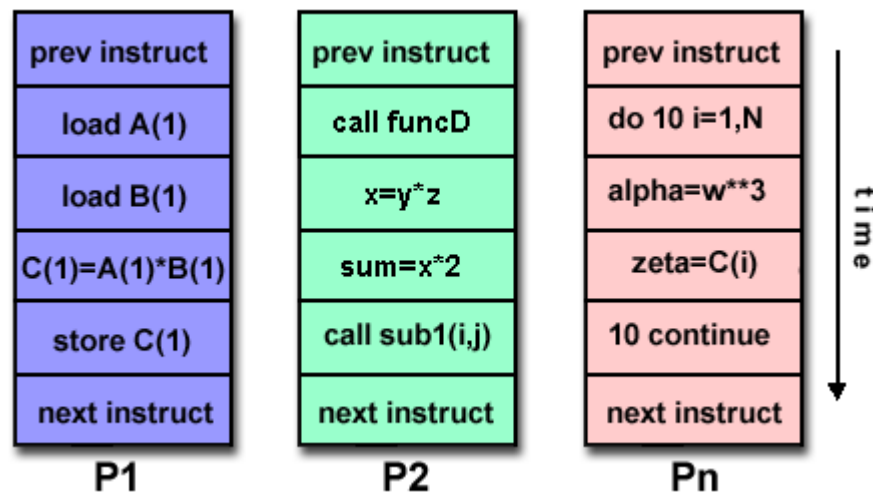
Multiple Instruction, Single Data (MISD)

- A single data stream is fed into multiple processing units.
- Each processing unit operates on the data independently via independent instruction streams.
- **Few actual examples** of this class of parallel computer have ever existed. One is the experimental Carnegie-Mellon C.mmp computer (1971).
- Some conceivable uses might be:
 - multiple frequency filters operating on a single signal stream
- Multiple cryptography algorithms attempting to crack a single coded message.



Multiple Instruction, Multiple Data (MIMD)

- Currently, **the most common type of parallel computer**. Most modern computers fall into this category.
- Multiple Instruction: every processor may be executing a different instruction stream
- Multiple Data: every processor may be working with a different data stream
- Execution can be **synchronous** or **asynchronous**, **deterministic** or **non-deterministic**
- Examples: most current supercomputers, networked parallel computer "grids" and multi-processor SMP computers - including some types of PCs.



Like everything else, parallel computing has its own "jargon". Some of the more commonly used terms associated with parallel computing are listed below. Most of these will be discussed in more detail later.

- **Task**
 - A logically discrete section of computational work. A task is typically a program or program-like set of instructions that is executed by a processor.
- **Parallel Task**
 - A task that can be executed by multiple processors safely (yields correct results)
- **Serial Execution**
 - Execution of a program sequentially, one statement at a time. In the simplest sense, this is what happens on a one processor machine. However, virtually all parallel tasks will have sections of a parallel program that must be executed serially.

- **Parallel Execution**

- Execution of a program by more than one task, with each task being able to execute the same or different statement at the same moment in time.

- **Shared Memory**

- From a strictly hardware point of view, describes a computer architecture where all processors have direct (usually bus based) access to common physical memory. In a programming sense, it describes a model where parallel tasks all have the same "picture" of memory and can directly address and access the same logical memory locations regardless of where the physical memory actually exists.

- **Distributed Memory**

- In hardware, refers to network based memory access for physical memory that is not common. As a programming model, tasks can only logically "see" local machine memory and must use communications to access memory on other machines where other tasks are executing.

- **Communications**

- Parallel tasks typically need to exchange data. There are several ways this can be accomplished, such as through a shared memory bus or over a network, however the actual event of data exchange is commonly referred to as communications regardless of the method employed.

- **Synchronization**

- The coordination of parallel tasks in real time, very often associated with communications. Often implemented by establishing a synchronization point within an application where a task may not proceed further until another task(s) reaches the same or logically equivalent point.
- Synchronization usually involves waiting by at least one task, and can therefore cause a parallel application's wall clock execution time to increase.

- **Granularity**

- In parallel computing, granularity is a qualitative measure of the ratio of computation to communication.
- **Coarse:** relatively large amounts of computational work are done between communication events
- **Fine:** relatively small amounts of computational work are done between communication events

- **Observed Speedup**

- Observed speedup of a code which has been parallelized, defined as:
$$\frac{\text{wall-clock time of serial execution}}{\text{wall-clock time of parallel execution}}$$
- One of the simplest and most widely used indicators for a parallel program's performance.

- **Parallel Overhead**

- The amount of time required to coordinate parallel tasks, as opposed to doing useful work. Parallel overhead can include factors such as:
 - Task start-up time
 - Synchronizations
 - Data communications
 - Software overhead imposed by parallel compilers, libraries, tools, operating system, etc.
 - Task termination time

- **Massively Parallel**

- Refers to the hardware that comprises a given parallel system - having many processors. The meaning of many keeps increasing, but BG/L pushes this number to 6 digits

- **Scalability**

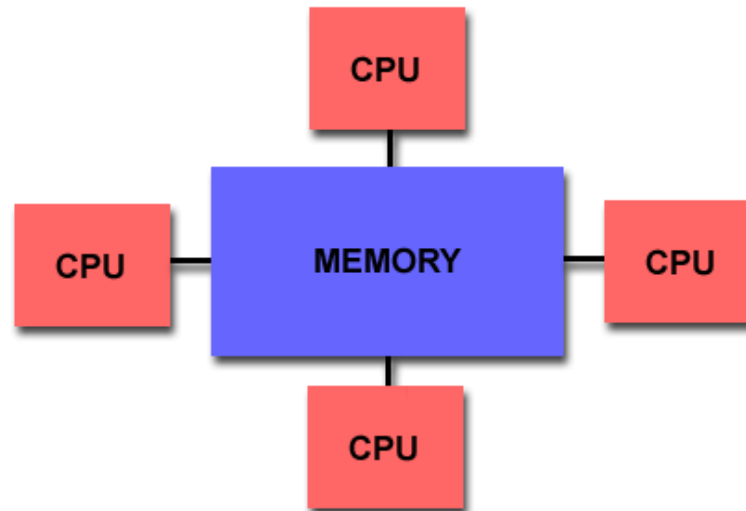
- Refers to a parallel system's (hardware and/or software) ability to demonstrate a proportionate increase in parallel speedup with the addition of more processors. Factors that contribute to scalability include:
 - Hardware - particularly memory-cpu bandwidths and network communications
 - Application algorithm
 - Parallel overhead related
 - Characteristics of your specific application and coding

Parallel Computer Memory Architectures

- Shared Memory
- Distributed Memory
- Hybrid Distributed-Shared Memory

Shared Memory

- Shared memory parallel computers vary widely, but generally have in common the ability for all processors to access all memory as global address space.



- Multiple processors can operate independently but share the same memory resources.
- Changes in a memory location effected by one processor are visible to all other processors.
- Shared memory machines can be divided into two main classes based upon memory access times: **UMA** and **NUMA**.

Shared Memory : UMA vs. NUMA

- **Uniform Memory Access (UMA):**
 - Most commonly represented today by Symmetric Multiprocessor (SMP) machines
 - Identical processors
 - **Equal access and access times to memory**
 - Sometimes called CC-UMA - Cache Coherent UMA. Cache coherent means if one processor updates a location in shared memory, all the other processors know about the update. Cache coherency is accomplished at the hardware level.
- **Non-Uniform Memory Access (NUMA):**
 - Often made by physically linking two or more SMPs
 - One SMP can directly access memory of another SMP
 - **Not all processors have equal access time to all memories**
 - Memory access across link is slower
 - If cache coherency is maintained, then may also be called CC-NUMA - Cache Coherent NUMA

- **Advantages**

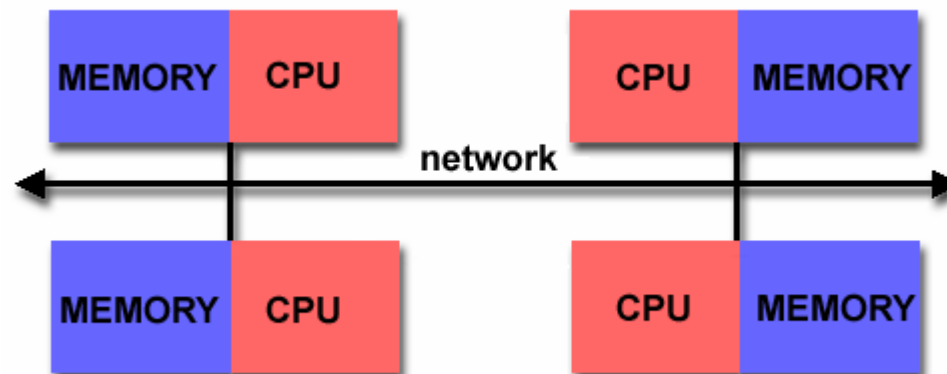
- Global address space provides a user-friendly programming perspective to memory
- Data sharing between tasks is both fast and uniform due to the proximity of memory to CPUs

- **Disadvantages:**

- Primary disadvantage is the lack of scalability between memory and CPUs. Adding more CPUs can geometrically increase traffic on the shared memory-CPU path, and for cache coherent systems, geometrically increase traffic associated with cache/memory management.
- Programmer responsibility for synchronization constructs that insure "correct" access of global memory.
- Expense: it becomes increasingly difficult and expensive to design and produce shared memory machines with ever increasing numbers of processors.

Distributed Memory

- Like shared memory systems, distributed memory systems vary widely but share a common characteristic. Distributed memory systems require a **communication network** to connect inter-processor memory.
- **Processors have their own local memory.** Memory addresses in one processor do not map to another processor, so there is no concept of global address space across all processors.
- Because each processor has its own local memory, it operates independently. Changes it makes to its local memory have no effect on the memory of other processors. Hence, the concept of **cache coherency does not apply**.
- When a processor needs access to data in another processor, it is usually the task of the programmer to explicitly define how and when data is communicated. Synchronization between tasks is likewise the programmer's responsibility.
- The network "fabric" used for data transfer varies widely, though it can be as simple as Ethernet.



Distributed Memory: Pro and Con

- **Advantages**

- Memory is scalable with number of processors. Increase the number of processors and the size of memory increases proportionately.
- Each processor can rapidly access its own memory without interference and without the overhead incurred with trying to maintain cache coherency.
- Cost effectiveness: can use commodity, off-the-shelf processors and networking.

- **Disadvantages**

- The programmer is responsible for many of the details associated with data communication between processors.
- It may be difficult to map existing data structures, based on global memory, to this memory organization.
- Non-uniform memory access (NUMA) times

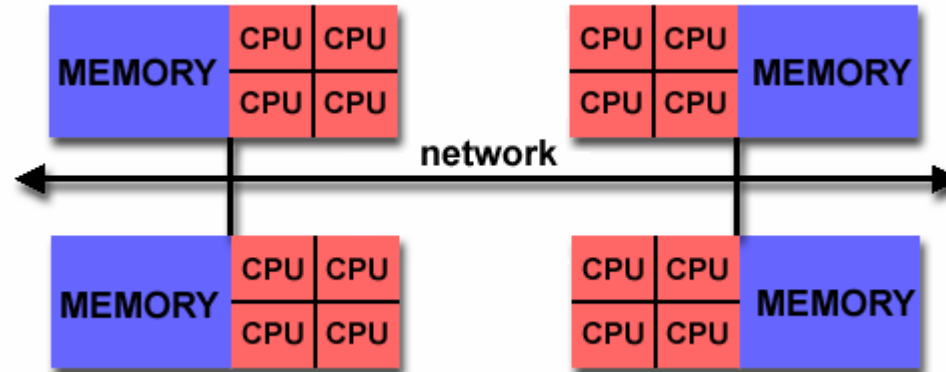
Hybrid Distributed-Shared Memory

Summarizing a few of the key characteristics of shared and distributed memory machines

Comparison of Shared and Distributed Memory Architectures			
Architecture	CC-UMA	CC-NUMA	Distributed
Examples	SMPs Sun Vexx DEC/Compaq SGI Challenge IBM POWER3	Bull NovaScale SGI Origin Sequent HP Exemplar DEC/Compaq IBM POWER4 (MCM)	Cray T3E Maspar IBM SP2 IBM BlueGene
Communications	MPI Threads OpenMP shmem	MPI Threads OpenMP shmem	MPI
Scalability	to 10s of processors	to 100s of processors	to 1000s of processors
Draw Backs	Memory-CPU bandwidth	Memory-CPU bandwidth Non-uniform access times	System administration Programming is hard to develop and maintain
Software Availability	many 1000s ISVs	many 1000s ISVs	100s ISVs

Hybrid Distributed-Shared Memory

- The largest and fastest computers in the world today employ both shared and distributed memory architectures.



- The shared memory component is usually a cache coherent SMP machine. Processors on a given SMP can address that machine's memory as global.
- The distributed memory component is the networking of multiple SMPs. SMPs know only about their own memory - not the memory on another SMP. Therefore, network communications are required to move data from one SMP to another.
- Current trends seem to indicate that this type of memory architecture will continue to prevail and increase at the high end of computing for the foreseeable future.
- Advantages and Disadvantages: whatever is common to both shared and distributed memory architectures.