Introduction to Parallel Computing. Lesson 1

Adriano FESTA

Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica, L'Aquila

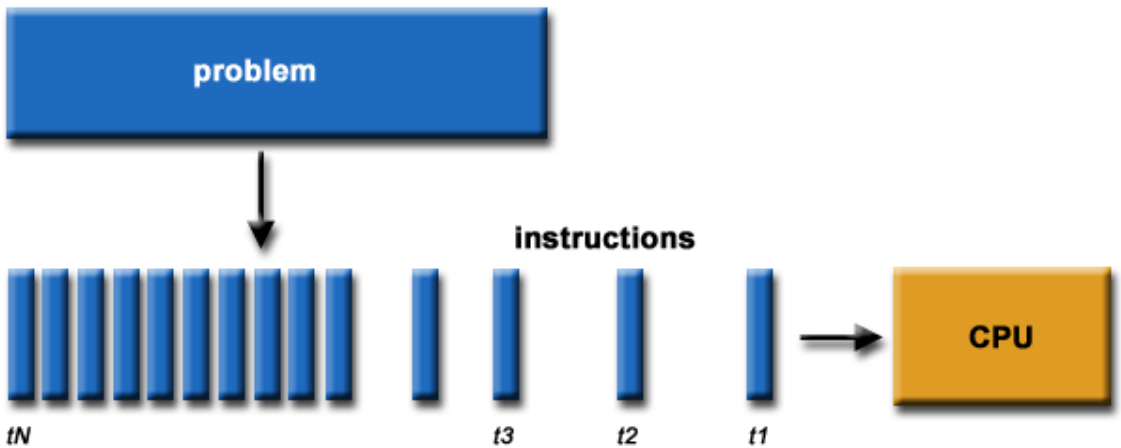
DISIM, L'Aquila, 25.02.2019

adriano.festa@univaq.it

What is Parallel Computing?

Traditionally, software has been written for serial computation:

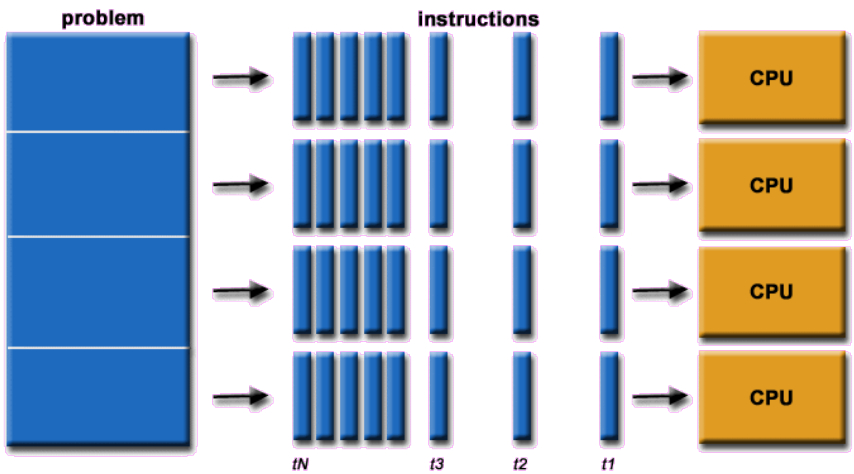
- To be run on a single computer having a single Central Processing Unit (CPU);
- A problem is broken into a discrete series of instructions.
- Instructions are executed one after another.
- Only one instruction may execute at any moment in time



What is Parallel Computing?

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



Parallel Computing: resources

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 -Field Programmable Gate Array ...)
- An arbitrary number of computers connected by a network;
- A combination of both.

Parallel computing: the computational problem

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.

Topic Overview

- Motivating Parallelism
- Scope of Parallel Computing Applications
- Parallel Computers
- Terminology
- Organization and Contents of the Course

Motivating Parallelism

- The role of parallelism in accelerating computing speeds has been recognized for several decades.
- Its role in providing multiplicity of datapaths and increased access to storage elements has been significant in commercial applications.
- The scalable performance and lower cost of parallel platforms is reflected in the wide variety of applications.

Motivating Parallelism

- Developing parallel hardware and software has traditionally been time and effort intensive.
- If one is to view this in the context of rapidly improving uniprocessor speeds, one is tempted to question the need for parallel computing.
- There are some unmistakable trends in hardware design, which indicate that uniprocessor (or implicitly parallel) architectures may not be able to sustain the rate of *realizable* performance increments in the future.
- This is the result of a number of fundamental physical and computational limitations.
- The emergence of standardized parallel programming environments, libraries, and hardware have significantly reduced time to (parallel) solution.

The Computational Power Argument

Moore's law[1] i.e. the observation that the number of transistors in a dense integrated circuit doubles about every two years, states (1965):

"The complexity for minimum component costs has increased at a rate of roughly a factor of two per year. Certainly over the short term this rate can be expected to continue, if not to increase. Over the longer term, the rate of increase is a bit more uncertain, although there is no reason to believe it will not remain nearly constant for at least 10 years. That means by 1975, the number of components per integrated circuit for minimum cost will be 65,000."

[1] Gordon Earle Moore (born January 3, 1929) is an American businessman, engineer, and the co-founder and chairman emeritus of Intel Corporation

The Computational Power Argument

Moore attributed this doubling rate to exponential behavior of die[2]
In 1975, he revised this law as follows:

"There is no room left to squeeze anything out by being clever. Going forward from here we have to depend on the two size factors - bigger dies and finer dimensions."

He revised his rate of circuit complexity (i.e. the number of transistors) doubling to 18 months and projected from 1975 onwards at this reduced rate.

[2] A die, in the context of integrated circuits, is a small block of semiconducting material on which a given functional circuit is fabricated. Typically, integrated circuits are produced in large batches on a single wafer of electronic-grade silicon (EGS) or other semiconductor (such as GaAs) through processes such as photolithography. The wafer is cut (diced) into many pieces, each containing one copy of the circuit. Each of these pieces is called a die. sizes, finer minimum dimensions, and "circuit and device cleverness".

The Computational Power Argument

- If one is to buy into Moore's law, the question still remains – how does one translate transistors into useful OPS (operations per second)?

  (By the way, a typical processor chip for desktop computers from 2009 consists of 400–800 million transistors.)
- The logical recourse is to rely on parallelism, both implicit and explicit.
- Most serial (or seemingly serial) processors rely extensively on implicit parallelism.
- We focus in this class, for the most part, on explicit parallelism.

The Memory/Disk Speed Argument

- While clock rates of high-end processors have increased at roughly 40% per year over the past decade, DRAM access times have only improved at the rate of roughly 10% per year over this interval.
- This mismatch in speeds causes significant performance bottlenecks.
- Parallel platforms provide increased bandwidth to the memory system.
- Parallel platforms also provide higher aggregate caches.
- Principles of locality of data reference and bulk access, which guide parallel algorithm design also apply to memory optimization.
- Some of the fastest growing applications of parallel computing utilize not their raw computational speed, rather their ability to pump data to memory and disk faster.

The Data Communication Argument

- As the network evolves, the vision of the Internet as one large computing platform has emerged.
- This view is exploited by applications such as SETI@home[3] and Folding@home[4].
- In many other applications (typically databases and data mining) the volume of data is such that they cannot be moved.
- Any analyses on this data must be performed over the network using parallel techniques.

[3] SETI@home ("SETI at home") is an Internet-based public volunteer computing project employing the BOINC software platform created by the Berkeley SETI Research Center and is hosted by the Space Sciences Laboratory, at the University of California, Berkeley. Its purpose is to analyze radio signals, searching for signs of extraterrestrial intelligence, and as such is one of many activities undertaken as part of the worldwide SETI effort.

[4] Folding@home (FAH or F@h) is a distributed computing project for disease research that simulates protein folding, computational drug design, and other types of molecular dynamics. The project uses the idle processing resources of thousands of personal computers owned by volunteers who have installed the software on their systems. Its main purpose is to determine the mechanisms of protein folding, which is the process by which proteins reach their final three-dimensional structure, and to examine the causes of protein misfolding.

Scope of Parallel Computing Applications

- Parallelism finds applications in very diverse application domains for different motivating reasons.
- These range from improved application performance to cost considerations.

Applications in Engineering and Design

- Design of airfoils (optimizing lift, drag, stability), internal combustion engines (optimizing charge distribution, burn), high-speed circuits (layouts for delays and capacitive and inductive effects), and structures (optimizing structural integrity, design parameters, cost, etc.).
- Design and simulation of micro- and nano-scale systems.
- Process optimization, operations research.

Scientific Applications

- Functional and structural characterization of genes and proteins.
- Advances in computational physics and chemistry have explored new materials, understanding of chemical pathways, and more efficient processes.
- Applications in astrophysics have explored the evolution of galaxies, thermonuclear processes, and the analysis of extremely large datasets from telescopes.
- Weather modeling, mineral prospecting, flood prediction, etc., are other important applications.
- Bioinformatics and astrophysics also present some of the most challenging problems with respect to analyzing extremely large datasets.

Commercial Applications

- Some of the largest parallel computers power the wall street!
- Data mining[5] and analysis for optimizing business and marketing decisions.
- Large scale servers (mail and web servers) are often implemented using parallel platforms.
- Applications such as information retrieval and search are typically powered by large clusters.

[5]Data mining is the process of discovering patterns in large data sets involving methods at the intersection of machine learning, statistics, and database systems.

Applications in Computer Systems

- Network intrusion detection, cryptography, multiparty computations are some of the core users of parallel computing techniques.
- Embedded systems increasingly rely on distributed control algorithms.
- A modern automobile consists of tens of processors communicating to perform complex tasks for optimizing handling and performance.
- Conventional structured peer-to-peer networks impose overlay networks and utilize algorithms directly from parallel computing.

Parallel Computers or cluster systems

A computer cluster is a set of loosely or tightly connected computers that work together so that, in many respects, they can be viewed as a single system. Unlike grid computers, computer clusters have each node set to perform the same task, controlled and scheduled by software.

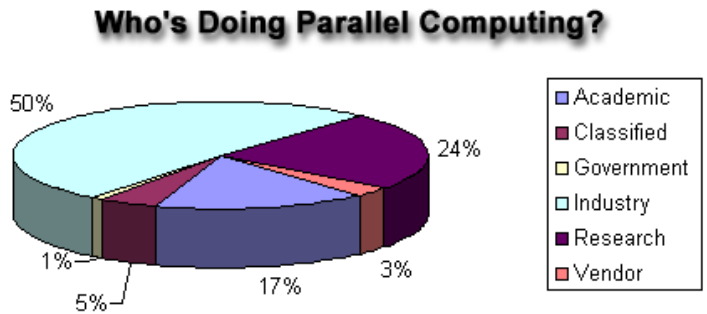
The components of a cluster are usually connected to each other through fast local area networks, with each node (computer used as a server) running its own instance of an operating system.

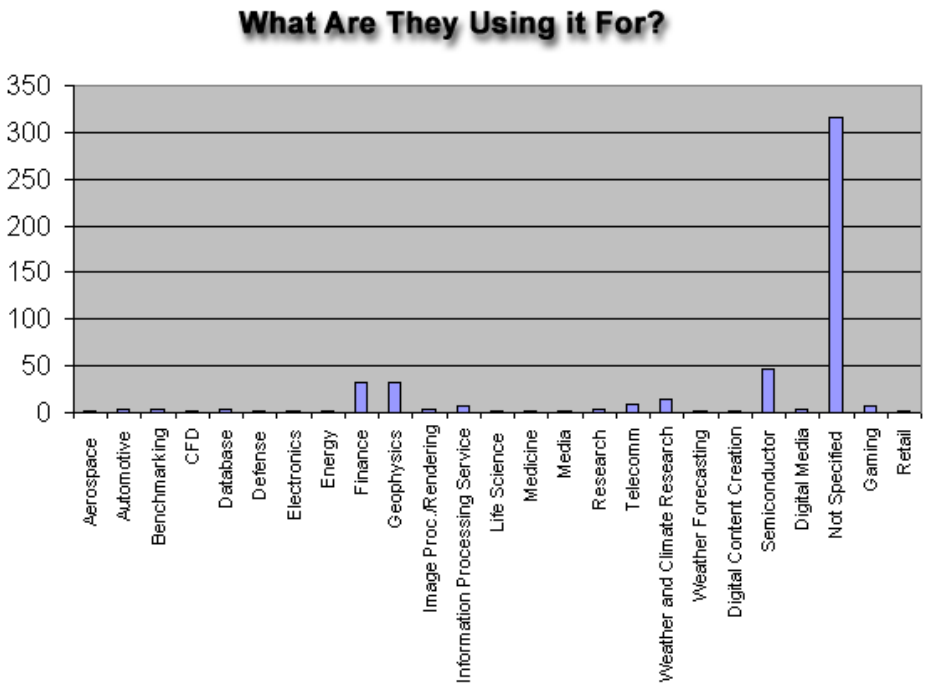
In most circumstances, all of the nodes use the same hardware and the same operating system.

Cluster systems

The physical computation unites are generally called cores.
The necessity to improve the clock speed (bottleneck coming by the overheating generated by the presence of more transistors for computational units) brought to the use of multicore systems even in the most common personal computers.

Who uses Parallel Computing?





One example: Dell XPS-13

Intel® Core™ i7-8565U Processor

8M Cache, up to 4.60 GHz

Add to Compare

Specifications

- **Essentials**
- Performance
- Supplemental Information
- Memory Specifications
- Processor Graphics
- Expansion Options
- Package Specifications
- Advanced Technologies
- Security & Reliability

Ordering and Compliance

Product Images

Downloads and Software

Essentials

Export specifications

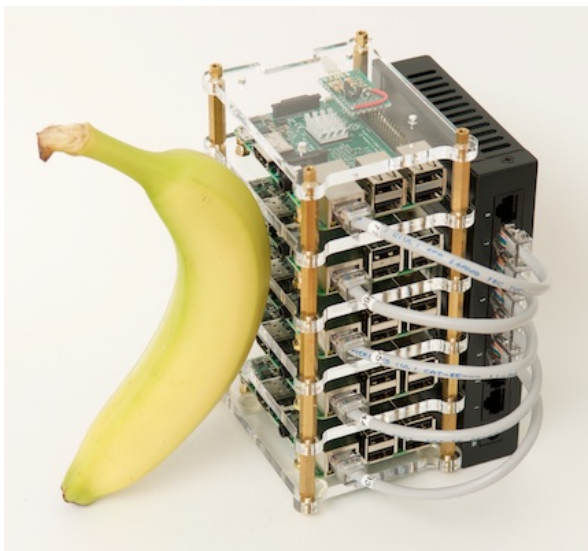
Product Collection	8th Generation Intel® Core™ i7 Processors
Code Name	Products formerly Whiskey Lake
Vertical Segment	Mobile
Processor Number	i7-8565U
Status	Launched
Launch Date	Q3'18
Lithography	14 nm
Recommended Customer Price	$409.00

Performance

# of Cores	4
# of Threads	8
Processor Base Frequency	1.80 GHz
Max Turbo Frequency	4.60 GHz
Cache	8 MB SmartCache
Bus Speed	4 GT/s OPI
TDP	15 W
Configurable TDP-up Frequency	2.00 GHz
Configurable TDP-up	25 W

Other examples:

IntelCore i7



Raspberry Pi Dramble 6-node Pi cluster

One more example: Caliban@Disim

18 Computing Servers with 4 processors 8 core AMD Opteron 2.0 GHz, for a total of 576 cores, Total Memory 1.5 TB, Storage Area Network (SAN) ISCSI RAID, 72 TB

Terminology: basic elements

Parallel compilers are in generally not completely satisfactory, they need some help from the programmer

- Tasks = Collection of blocks of operation which compose the program
- Granularity = property related to the size on what the parallel program is decomposed
- Potential parallelism = inherent property related to the possibility of parallelization of a program
- processes/threads= Tasks assigned to computational units

Von Neumann architecture

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.

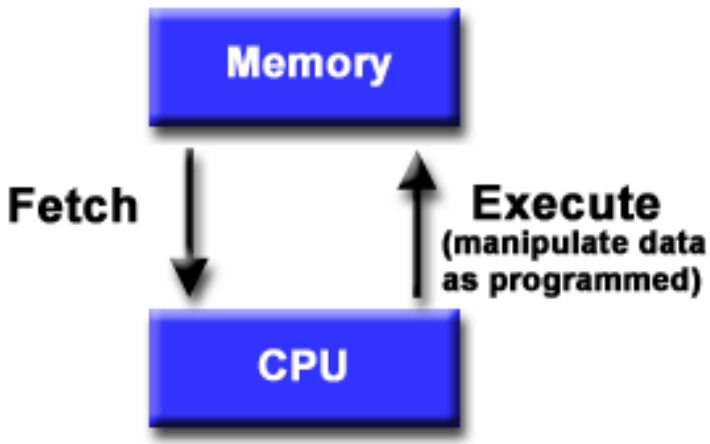
A von Neumann computer uses the stored-program concept. The CPU executes a stored program that specifies a sequence of read and write operations on the memory.

Von Neumann architecture: basic design

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.



Flynn's classical taxonomy

- 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.

Flynn's classical taxonomy

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**

Terminology

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.

Terminology: basic actions

- Scheduling = Assigning tasks to threads (What? Done by the user or by the compiler)
- Mapping = Assigning tasks to threads (Where? Done generally by the compiler)
- Synchronization = Tasks may be dependent, timing is important

Synchronization: timing and machine architecture

- Shared memory = each threads has access to a common area of the memory. Time of access is important!
- Distributed memory = each process has a specific memory unit to store and read data
- Generally with threads we refer to shared memory units, with processes to distributed architectures (need of a passing interface to communicate operations)
- 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: timing and machine architecture

- 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.
- Barrier operations, idle times: Synchronization usually involves waiting by at least one task, and can therefore cause a parallel application's wall clock execution time to increase.

Terminology: evaluation performances

- Parallel execution time = time needed to run the program in parallel, including execution time of the threads and idle times.
- Idle times = waiting time for threads/processes
- Speed up = improvement performed by the parallel version of the program compared to its sequential version
- Load balancing = rational distribution of the computing load on the threads/processes in order to improve efficiency.

Terminology: evaluation performances

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:

$$\text{Speed-up} = \frac{\text{wall-clock time of serial execution}}{\text{wall-clock time of parallel execution}}$$

- One of the simplest and most widely used indicators for a parallel program's performance.

Terminology: evaluation performances

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

Contents of this Course

- Fundamentals: This part of the class covers basic parallel platforms, principles of algorithm design, group communication primitives, and analytical modeling techniques.
- Parallel Programming: This part of the class deals with programming using message passing libraries and threads.
- Parallel Algorithms: This part of the class covers basic algorithms for matrix computations, graphs, sorting, discrete optimization, and dynamic programming.