0.1 Parallel computing: an overview

(Lesson 1 of 08/10/19)
Compiled: 10 ottobre 2019

Parallel computing is needed to better harness modern computer hardware, and efficiently solve difficult challenges.
This is a task very different from traditional computing (Von Neumann), requiring new architectures.
Depending on the number of **data streams**, that is independent accesses to data, and **processors**, that is units capable of computation, we can distinguish 4 basic possibilities.

For example, for a *single input* with *multiple processors* we have SIMD. Parallel computing usually follows the MIMD architecture - with multiple data streams and many processors.

New hardware can be used for parallel computing: one example are GPUs, which are able to do a lot more of parallel computation than a CPU can do, as they consists of a larger portion of transistors devoted to basic computations (where CPUs focus on control and data flow). Basically, GPUs are faster when applying simple operations (e.g. multiplication) to many inputs at once, while CPUs can perform much more difficult tasks, at the cost of speed. Note that this speedup is only thanks to parallel execution: if we compare a GPU and a CPU on the same (sequential) task, the CPU will perform better.

GPUs also host very fast RAM memory on-board, leading to a faster access of data. Of course this is a trade-off: adding memory very near to the processors will decrease their number, or efficiency.

Obviously, to harness the power of parallel computing also new algorithms are needed - and software needs to be optimized for running on a parallel architecture.

0.1.1 Harvard architecture

In all of the previous examples, data and code share the same buses ("links" between hardware components).
However, simply separating instructions and data can lead to a speedup, and it's the main focus of the Harvard architecture.
In fact, modern CPUs are some kind of hybrid between the von Neumann and Harvard's architecture.

Some examples of practical applications are Digital Signal Processors, which employ a *fixed set of instructions*, that cannot be modified at runtime, and also Micro-Controllers.

0.1.2 ASICs vs FPGA

All circuits we discussed, like CPUs, GPUs, DSPs and Micro-Controllers, are *fixed*: i.e. are a set of gates connected in a certain fixed designed way (ASIC, Application Specific Integrated Circuits).
However, it is possible to *re-program* the very connections between gates by using a different architecture - that of FPGA (Field-Programmable Gate Arrays).