PMC Lecture 01

Gianmaria Romano

23 September 2025

Course generalities

General Info – Part II

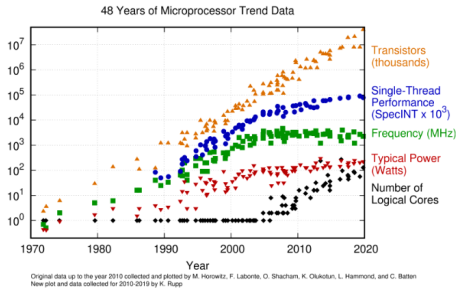
- **Email:** desensi@di.uniroma1.it
- **Schedule:**
  - Tuesday from 17:00 till 19:00 (Aula Magna RM111)
  - Wednesday from 14:00 till 17:00 (Aula 11, Via Scarpa)
    - **ATTENTION: Tomorrow, 203 Marco Polo**
- **Question Time:** You can book question time slots by emailing me (we will agree on a day/time/place).
- **Books:**
  - ***An Introduction to Parallel Programming*, 2nd ed., Pacheco & Malensek**
  - **Two additional books for the GPU part (you can find details on Moodle)**
  - **Extra resources will be linked on Moodle**
- Interactions/questions during lectures are welcome
- Hands-on sessions/lab during some of the three hours slots (bring your laptop)
- **Exam:** Project + oral exam (grade is the average of the two)
  - **Possibility of extra points for intermediate evaluations (TBD)**

Chapter 1

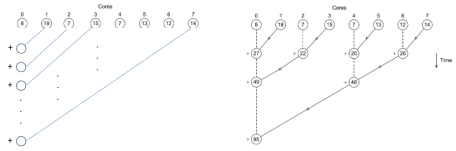
Parallel computing

1.1 The rise of parallel systems

From 1986 to 2023, microprocessor performance has increased by 50% per year, although, due to physical limitations, this increase has slowed down since 2003, leading to the "abandonment" of powerful monolithic systems in favour of placing multiple processors on a single circuit.



However, since sequential programs cannot benefit from multicore systems, it can come in handy to convert these codes into parallel constructs, but, since this approach tends to be inconvenient, it is more convenient to devise brand new algorithms that can effectively exploit parallelism.
Example: Suppose you want to compute the sum of n integers: by employing $p << n$ cores, an easy way of parallelizing this task consists in having each core compute the sum of $\frac{n}{p}$ numbers and, when all the partial sums are ready, a "master" core will deal with providing the global sum of the numbers.
However, since this implementation leaves most of the work to the master core, it is also possible to opt for a tree reduction that, in order to increase activity, computes pairwise partial sums.



1.2 Common parallelism strategies

In order to improve coordination and synchronization, parallel programs typically exploit task parallelism, which consists in partitioning tasks among cores, or data parallelism, which consists in partitioning data among cores so that each processor can work on its chunk of data.
Example: Suppose a professor with three assistants has to grade 300 exams with 15 questions each: applying data parallelism would have each assistant grade 100 exams, whereas task parallelism would have each assistant grade just some questions for each exam.

1.3 Writing parallel systems

Generally speaking, it is possible to write parallel programs by using some dedicated APIs provided by the C programming language.

1.3.1 Implementing memory structures

Regarding memory structures, a parallel system can implement a shared memory structure, where cores can access a common memory space, or a distributed memory structure, which provides a private memory for each core and thus coordinates activity by message passing over a dedicated network.

1.3.2 Implementing control units

Regarding control units, a parallel system can opt for a multiple-instruction multiple-data approach, where each core has its own control unit and can work independently of the other cores, or a single-instruction multiple-data approach, where cores share the same control unit and can either execute the same instructions or stay idle.