Parallel computing

- "History": From 1980's to ~2004, processor performance increased mainly due to frequency scaling (increased clock rate).

[Codes would run faster and faster without changes]

$$\text{Runtime} = \frac{\text{instructions}}{\text{program}} \times \frac{\text{cycles}}{\text{instruction}} \times \underbrace{\frac{\text{time}}{\text{cycle}}}_{\frac{1}{f_{proc.}}}$$

$$\left[f_{proc} = \frac{\#\text{cycles}}{\text{second}} \right]$$

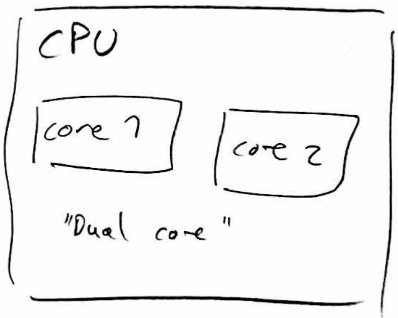
Power consumption in chip:

$$P \propto f$$

so increased f → increased P (as expected!)

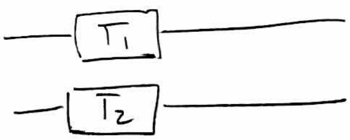
⇒ Problems w/ overheating etc.

From ~2004; performance increase mainly from shift to parallel comp., and in part multicore processors



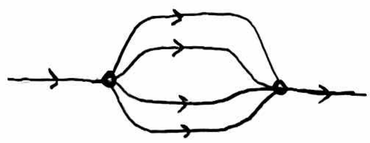
- Challenge: Requires changes on software side! Need to distribute tasks or data across threads or processes!

[Old-school parallelization: Give each student in a class their own ~~eq~~ eq to solve ☺]

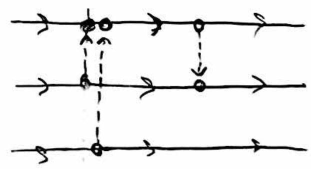


- Two main approches

1) - Shared memory
   - Threading
   - Single computer/node
   - Example: OpenMP, <thread>
   - Single process, can switch between one and multiple threads

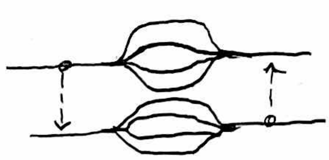


2) - Distributed memory
   - Message passing
   - Can be used on single computer/node or between computers/nodes
   - Example: MPI
   - Multiple indep. processes



- These approaches can be combined:

  - Multiple processes, each spawning multiple threads (sharing that process' memory)



[Mention GAMBIT]

- Parallelization comes with some overhead, from spawning threads, passing messages, etc. Only useful if $\Delta t_{task} > \Delta t_{overhead}$. [Also: comes with substantial room for mistakes and bugs ...]

[Go through code examples in
code_examples/omp-parallelitation]