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]

$$\text{Runtime} = \frac{\text{instructions}}{\text{program}} \times \frac{\text{cycles}}{\text{instruction}} \times \frac{\text{time}}{\text{cycle}}$$

$$\frac{\text{time}}{\text{cycle}} = \frac{1}{f_{proc.}}$$

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

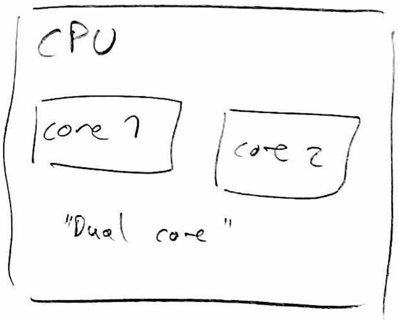
Power consumption in chip:

$$P \propto f$$

so increased f → increased P (as expected!)

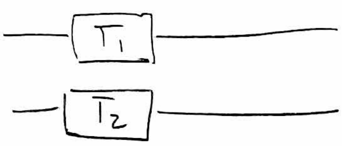
⇒ Problems w/ overheating etc.

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



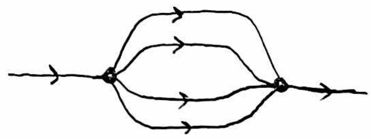
- Challenge: Requires changes on software side! Need to distribute tasks or data across threads or processes!

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

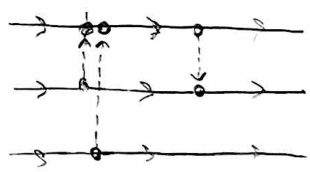


- Two main approches

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

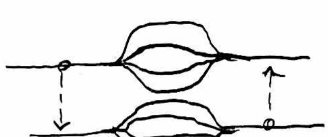


2) - Distributed memory
   - Message passing
   - Can be used on single computer / node or between computers / nodes
   - Example: MPI
   - Multiple indep. processes



- 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 ...]

- Parallelization (cont. from last time)

  - Recap:

    - Two approaches: Shared memory (e.g. OpenMP)
      Distributed memory (e.g. MPI)

- [Go through OpenMP examples in the Git repo]

- Different approaches to parallelize project 4:

  Alt 1) Parallelize loop over temperatures (simplest!)

  Alt 2) For each temp., use parallelization to run multiple MCMC chains (multiple "walkers")

  Can either:

  - Increase number of threads and decrease cycles per thread

  Each walker/chain needs burn-in

  ⇒ same accuracy, shorter time

  or:

  - Increase number of threads while keeping number of cycles per thread fixed

  ⇒ higher accuracy (more MC samples), same time

  Alt 3) For each temperature and each MC cycle, parallelize the "sweep" over the spin matrix

  - Most complicated! (Don't do this...)
  - Most overhead

- How to define speedup from parallelization

$$\text{Speedup} = \frac{\text{time with single thread / process}}{\text{time with } n \text{ threads/processes}} = \frac{T_1}{T_n}$$

- Ideal case: n threads $\Longleftrightarrow$ $T_n = \frac{T_1}{n}$ $\Longleftrightarrow$ speedup factor is n

  - In most cases we will not get ideal speedup
  - In rare cases we can get better than ideal speedup (e.g. through changes in memory access)

- Keep in mind: A complicated algorithm with less than ideal speedup from parallelization can still be a better choice than a simple algorithm with better (ideal?) parallelization speedup!

- Example: Find the maximum of a complicated, high-dim function through

  1) random sampling or grid scan (ideal speedup, "embarrasingly parallelizable")

  2) sophisticated optimization algorithm, e.g. differential evolution

  (needs communication and synchronization → less than ideal speedup)

[Show example from paper]

- <u>Upper bound on speedup:</u>
  - A task takes time T_1 on single thread/process
  - Fraction of time spent in perfectly parallelizable code: f
  - Non-parallelizable fraction: $1-f$

- Single thread/process:

$$T_1 = (1-f)T_1 + f\,T_1$$

- On n threads/processes:

$$T_n = (1-f)T_1 + f\frac{T_1}{n}$$

- Speedup:

$$\frac{T_1}{T_n} = \frac{T_1}{(1-f)T_1 + f\frac{T_1}{n}} = \frac{1}{(1-f) + \frac{f}{n}}$$

$$\boxed{\lim_{n \to \infty} \frac{T_1}{T_n} = \frac{1}{1-f}}$$

Amdahl's law

Example: If 99% of a task is parallelizable ($f = 0.99$) the maximum possible speedup factor is $\frac{1}{1-0.99} = \frac{1}{0.01} = 100$

Example 2: $f = 0.80 \Rightarrow$ max speedup is 5