- <u>Parallelitation</u> (cont. from last time)

  - <u>Recap</u>:

    - 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

    ⇒ <u>same accuracy</u>, <u>shorter time</u>

  or:

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

    ⇒ <u>higher accuracy</u> (more MC samples), <u>same time</u>

  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 $\Leftrightarrow$ $T_n = \frac{T_1}{n}$ $\Leftrightarrow$ speedup factor is n

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

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

- 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]

- **Upper bound on speedup:**
  - 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