Last time
- First steps with MPI

Today
- Computational scaling
  → how do my code performance in parallel?
  Strong, weak, machine scaling.
  (60s ← Strong, 90s → Weak, 2010s thing → machine)
- Data decomposition choices. → Vectors.

Strong scaling

$$S_p = \frac{T_1}{T_p}$$

(T_1 ← time on one process; T_p ← on p processes.)

speedup

Problem size is fixed.

Hope $S_p > 1$ when $p > 1$.

1 person can build a wall in 1 week. How long will 4 people take? $T_4 = \frac{T_1}{4}$ → ideal linear scaling

Gene Amdahl

parallel part.

$$T_p = f T_1 + \boxed{(1-f)\frac{T_1}{p}}$$

serial fraction | "parallel fraction"

$f T_1$ $\quad$ $(1-f) T_1$

Amdahl's Law:

$$\lim_{p \to \infty} T_p = f T_1$$

$$\Leftrightarrow \lim_{p \to \infty} S_p = \frac{1}{f}$$

So, if 1% of the code is serial, best speedup is?

$$S_\infty = \frac{1}{0.01} = 100.$$

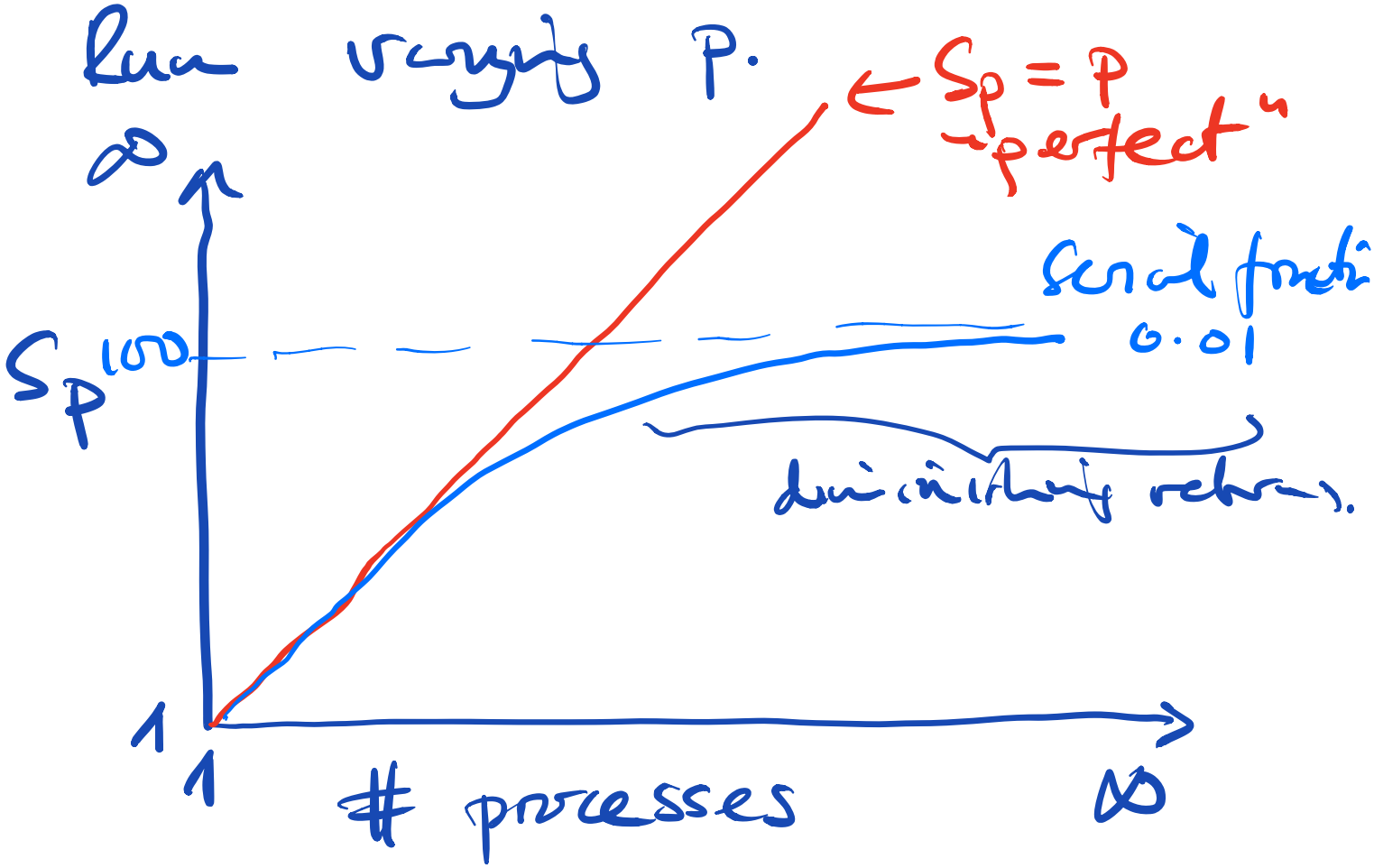
Consequence

→ Need to stamp out as much of the serial code as possible.

Presenting speedup data:

Problem size N: ← fix.

Run varying P.



Note: dividing at by T_1, so can hide inefficient implementation.

Can also plot efficiency.

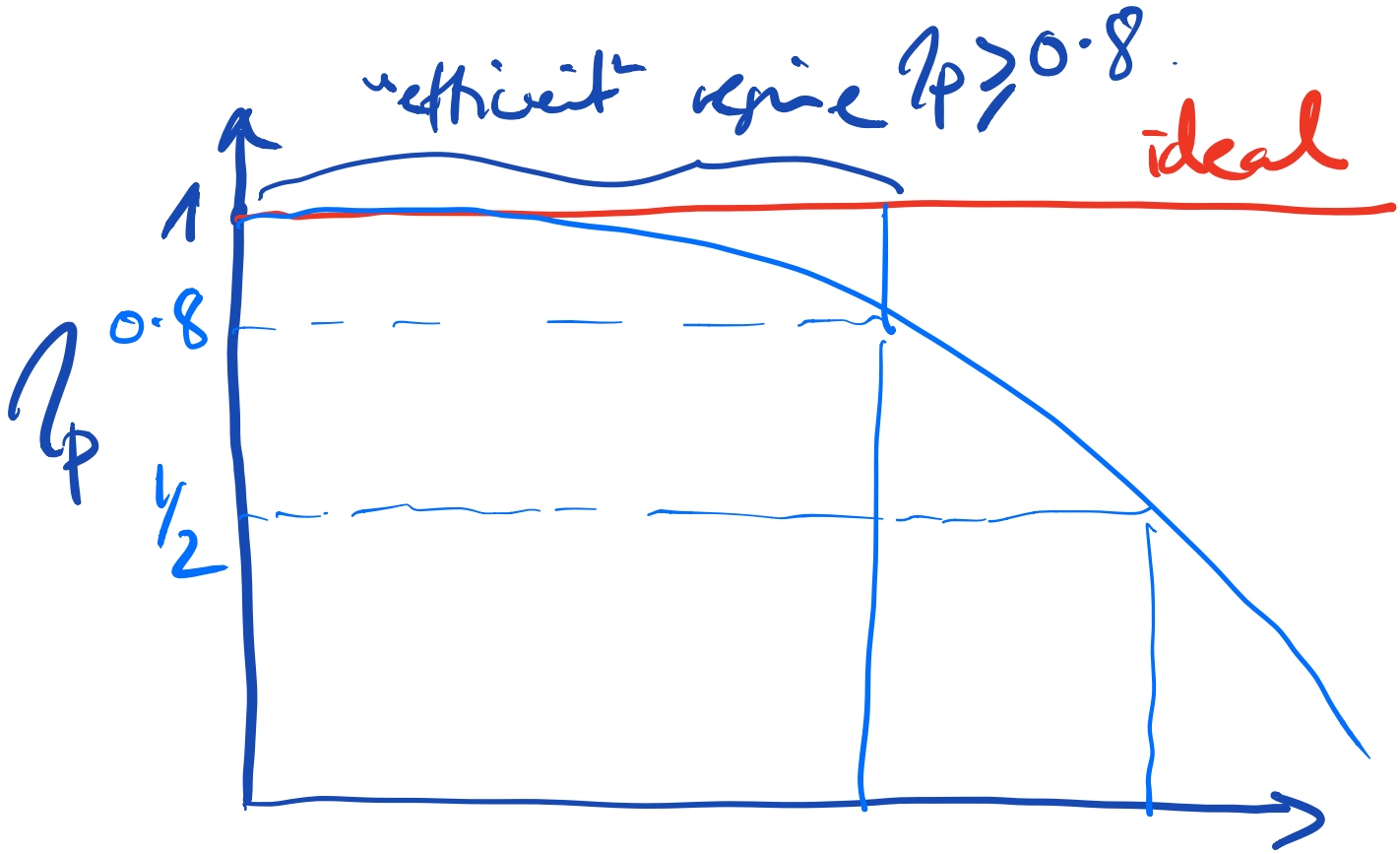
$\in [0, 1)$

$$\eta_p = \frac{T_1}{p\,T_p}$$

← total resource we used on p processes.

→ exercise

What is η_p as a function of the serial fraction, f?



Often on log scales

Weak scaling

small, grows slowly.

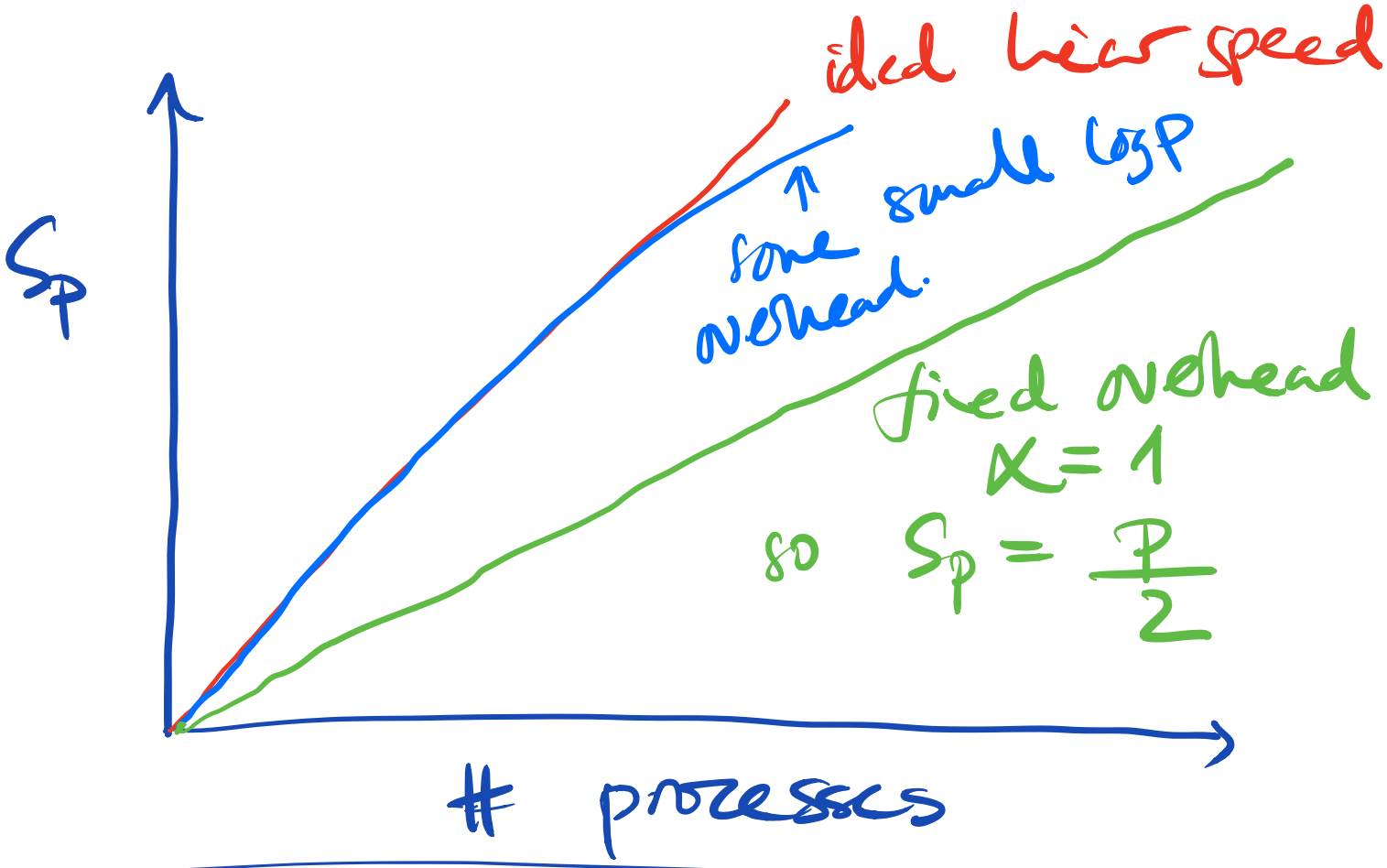
$$T_p = T_1 + \alpha(p) T_1$$

$$S_p = \frac{p T_1}{T_p}$$

~ Gustafson's law.

time to solve local problem.

Local (per process) problem size fixed.



Real applications scale problems up with weak scaling as then at fixed size do strong scaling.

$$\eta_p = \frac{T_1}{T_1 + o(p) t_1} = \frac{T_1}{1 + o(p)}$$

overhead fixed

$$\eta_p^{fix} = \frac{T_1}{1+\alpha} \quad ; \quad \varsigma_p^{fix} = p \frac{1}{1+\alpha}$$

$$\eta_p^{log} = \frac{T_1}{1+\log p} \quad ; \quad \varsigma_p^{log} = \frac{p}{1+\log p}$$

Typical setup for PDEs.

overhead : $\alpha + O(\log p)$

α : message latency

$O(\log p)$: reductions (combining data)

e.g. dot product $a \cdot b$ → bias → scale

"problem size fixed".

Dense matrix-matrix.

$R^{N \times N}$

$O(N^3)$ algorithm.

Weak scale.

If I double N.

& double P

→ & the time to solution

fixed?

$N \longrightarrow 2N$ · $P \longrightarrow 2P$

N^3 $\quad$ $8N^3$ / $\frac{N^3}{P} \longrightarrow \frac{8N^3}{2P} = 4N^3$

So to get constant T_p need to add 8x processes

$$N^3 \longrightarrow 8N^3$$

$$p \longrightarrow 8p$$

$$\frac{N^3}{p} \longrightarrow \frac{N^3}{p} \quad \checkmark$$

But local matrix size.

$$N \times N \longrightarrow \frac{N}{\sqrt{8}} \times \frac{N}{\sqrt{8}} \; ?$$

so local problem is smaller

$\Rightarrow$ strong scaling.

Machine / algorithmic scaling

Range of sizes, Time to sol

plot T vs $\frac{N}{T}$ → want flat.

→ Add link to paper in notes.

Vectors in parallel.

$x \in \mathbb{R}^N$

pointwise

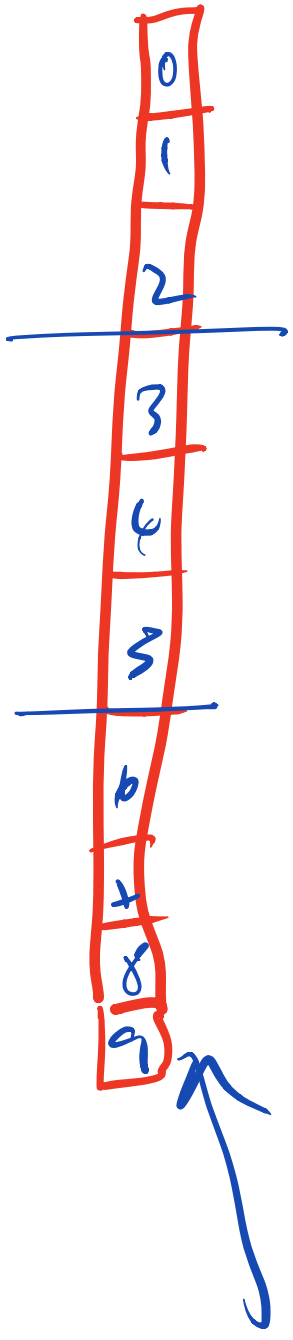
$y \leftarrow \alpha x + y$ $\quad y \in \mathbb{R}^N$, $\alpha \in \mathbb{R}$

$y \leftarrow 1/y$

⋮

collective. computing norms / dot products.

$\beta \leftarrow a \cdot b$ $\quad \beta \in \mathbb{R}$, $a, b \in \mathbb{R}^N$



portrix updates.
Image we have
3 processes.

Aim to evenly distribute
total length N
across the p processes.

Scalability limit for
portrix cases from
uneven distr of
local work.

⇒ "load imbalance"

3, 3, 4

DST products / grids
next time.