PSC II: Computational scaling

Idea

Our algorithm and data structure choices are guided by, at first, some simple scaling laws.

Types

1960s idea

Strong scaling

fixed size problem
Add more workers to the problem.

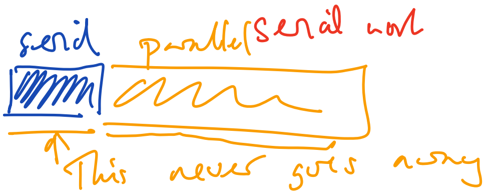
Gene Amdahl

$$T_p = \frac{\boxed{T_1}}{\boxed{P}} \quad \text{ideally.} \leftarrow \text{Hope}$$

Time on 1 process

processes

$$T_1 = \underbrace{fT_1}_{\text{serial work}} + \underbrace{(1-f)T_1}_{\text{parallel work.}}$$

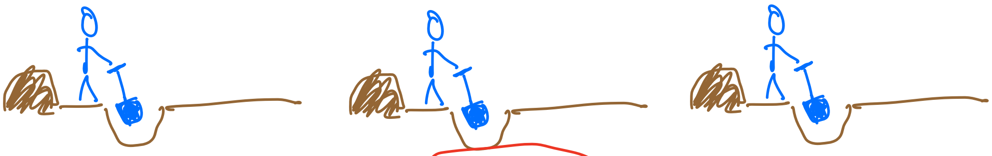


Amdahl says

$$T_p = \underbrace{fT_1}_{\text{serial fraction.}} + \frac{(1-f)T_1}{P}$$

If only 1% of code is serial then max speedup: $\frac{T_1}{T_p} = 100.$

Weak scaling   1990s   Fixed local size of problem.



$$T_p = \underbrace{\underbrace{T_1}_{\text{time to solve local problem}} + \overbrace{\underbrace{o(p)\,T_1}_{\text{grows slowly ``overhead''}}}^{}}_{\text{ideal scenario}}$$

overhead which is hopefully not too big.

Speedup: $$S_p = \frac{p\,T_1}{T_1 + O(p)T_1}$$

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

What does this o(p) term look like?

Best case is usually:

$$\underbrace{\alpha}_{\text{constant latency}} + \underbrace{\log p}_{\text{good algorithms use tree-based reductions.}}$$

Speedup and efficiency

Amdahl & Gustafson "laws"

strong scaling

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

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

$$= \frac{T_1}{T_1\left[f + \frac{1-f}{p}\right]}$$

$$= \frac{1}{f + \frac{1-f}{p}}$$

$$= \frac{p}{1-f+pf}$$

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

weak scaling

$$T_p = T_1 + o(p) T_1$$

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

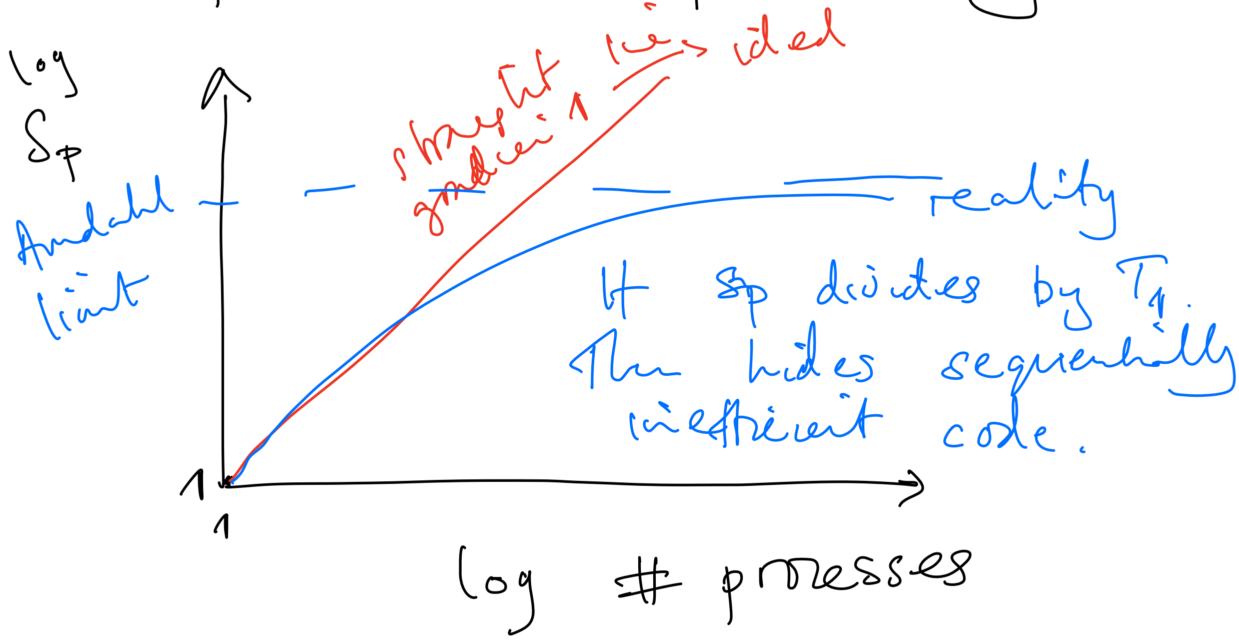
$$= \frac{p T_1}{T_1\left[1 + o(p)\right]}$$

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

Say $o(p)$ is $\log p$

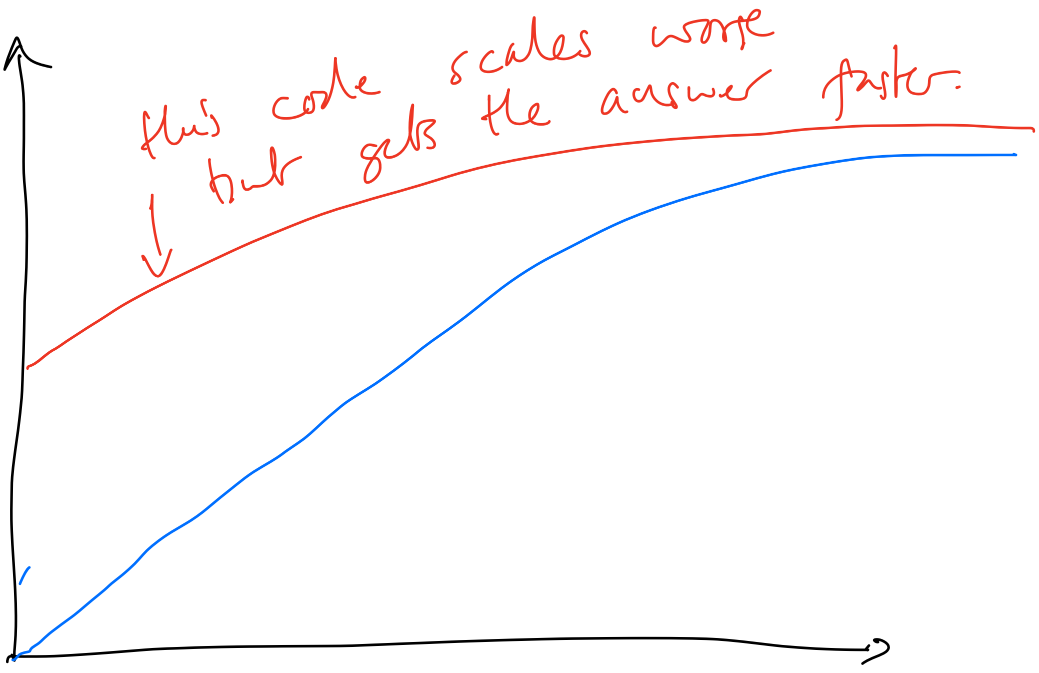
$$\lim_{p \to \infty} \frac{p}{1+\log p} = \frac{p}{\log p}$$

What does it look like on a plot / how to present things?



If Sp divides by T_1. Then hides sequentially inefficient code.

When comparing solutions of same problem with different algorithms prefer to normalise to the slowest one.



this code scales worse but gets the answer faster.

Define efficiency η_p as how close to ideal we are:

$$\eta_p = \frac{T_1}{\boxed{pT_p}}$$

strong scaling.

← total resource we used.

⇒ ideal $pT_p = T_1$ (no serial fraction).

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

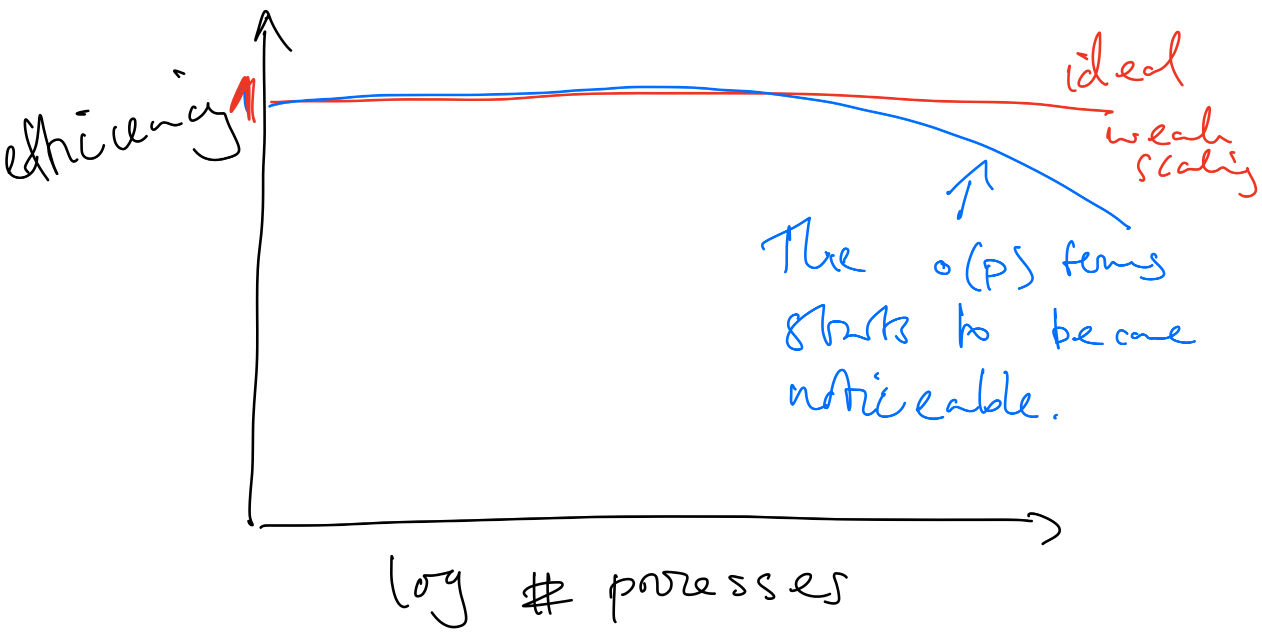
weak scaling

How much slower are we when we scale out?

⇒ ideal $T_p = T_1$

ie parallel overhead is zero.

Weak scaling



The o(p) terms starts to become noticeable.

What does "problem size" mean?

1 – Dense matrix-matrix

2 – Stationary PDE

3 – Time dep PDE

1) $C \Leftarrow C + A \times B \qquad A, B, C \in \mathbb{R}^{N \times N}$

Natural "size" is N.

→ weak scaling:

$N \to 2N$. matrices are $N^2 \to 4N^2$ entries.

Add 4 times as many processes
→ keeps local matrix size fixed.

But. N^3 work in algorithm.

⟹ work goes from $N^3 \to 8N^3$

⟹ need 8 times the process count.

But in this case each local problem gets smaller by $\sqrt{2}$.

⟹ "weak" scaling for D.M.M. doesn't really exist.

→ still need to strong scale.

Stationary PDEs.

Double dof-count $N \to 2N$
scalable solution method.
cost is $O(N \log N)$

$$\Rightarrow O(2N \log 2N)$$

$\Rightarrow$ adding twice the compute helps
and we can weak scale.

<u>time dep</u> : as we add resolution.
need more timesteps for
accuracy (even with implicit
methods)

– need more timesteps for
stability for explicit methods.

$\Rightarrow$ to get an answer on
a bigger problem in a fixed
time budget, need to <u>strong</u> scale.

Consequences for grid-based methods

Basic linear algebra operations.

"load balancing"