University of Leeds

Lecture 4: Theory of parallel perf

Previous lecture

In the last lecture we started to look at solving problems in parallel:

-
- `#pragma omp parallel for`
- **Mandelbrot set**, which has a **ne**
- Both **data parallel** problems ("**fo**
  loop are independent.
- Still difficult to achieve good performance for the Mandlebrot set.

This lecture

Now we will look at some general considerations for **parallel perf**

-
- Classic **models** for predicting paralle
  highlighting potential pitfalls.
- How these relate to **scaling**, *i.e.*
  the number of processors and the problem size.

Notation

For this lecture we will use the following notation:

s		
t_s	Serial execution time	ce "Optimal"
t_p	Parallel execution time	
f	Serial fraction	Amdahl, Gustafson-Barsis

What we are trying to achieve

Assume a problem can be solved by a **serial** algorithm in a time t_s.

- We assume this is **optimal**, *i.e.* cannot be improved (in serial).

In pra

- May be known, but take too long to implement.

Usually consider the serial algorithm which is 'eq parallel one.

- For instance, if developing a parallel bubblesort, would probably compare to **serial bubblesort** (rather than quicksort, mergesort, heapsort *etc.*).

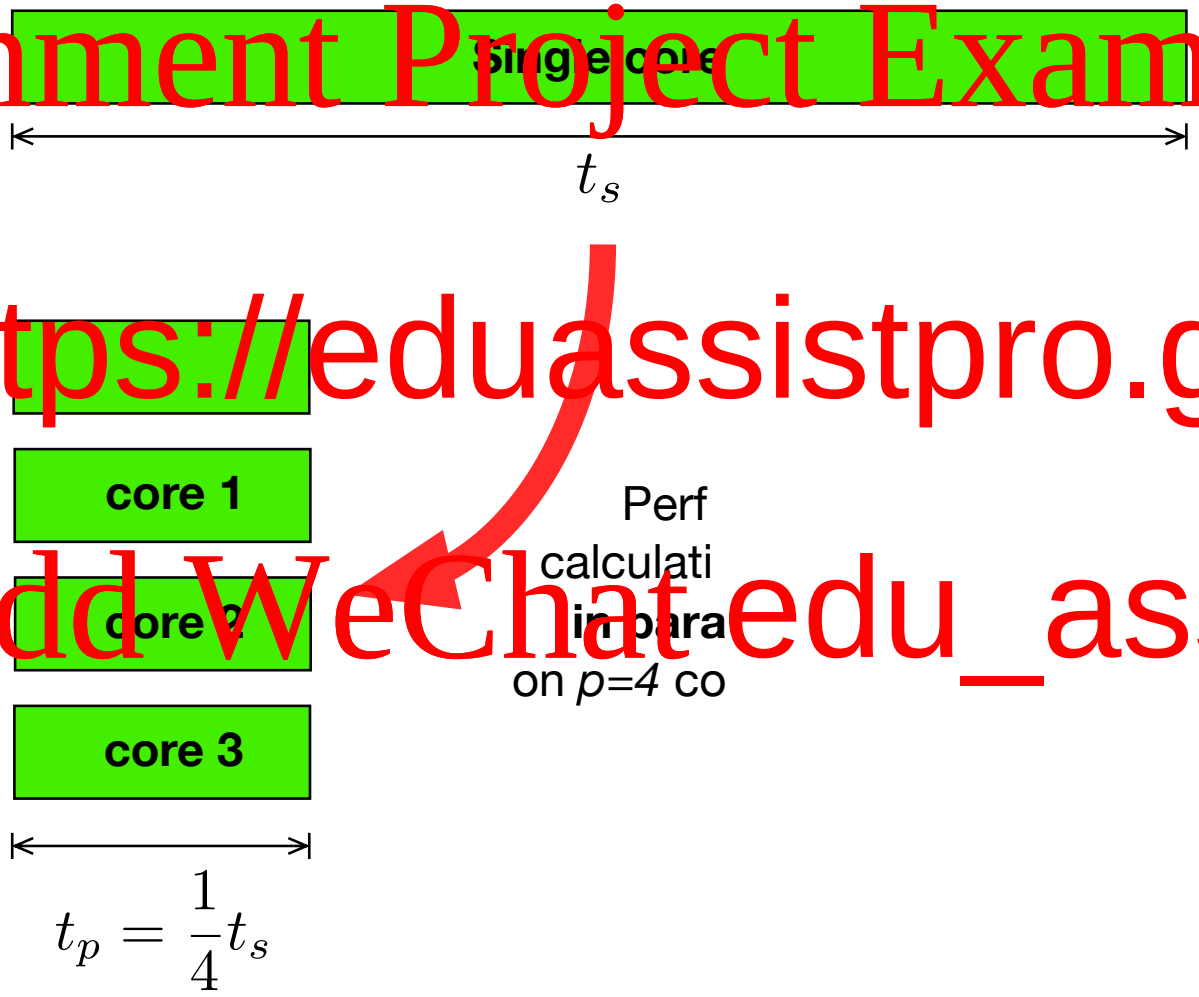
Parallel acceleration

One way to improve on the **serial** execution time t_s is to implement a **parallel** solution on **parallel** hardware.

Denote the (not necessarily optimal) parallel ex

- Measured in same units as t_s.
- On 'as similar as possible' hardware.
- Sometimes known as the **wall clock** time, as it is what 'a clock on the wall' would measure.

Simultaneous calculations (ideally)



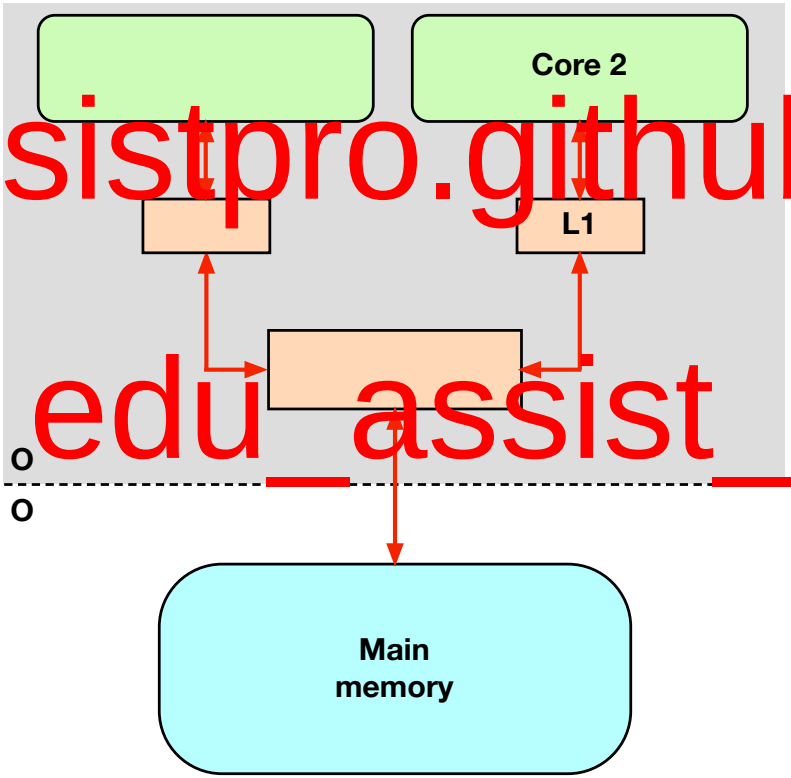
Perf
calculati
in para
on p=4 co

Multi-core memory cache

Recall from Lecture 2 that using multiple cores can make better use of **memory cache**.

Fewer

- C
c
required by another core.
- Depending on how data is arranged in memory and accessed, a parallel code may result in **fewer cache misses** overall that the equivalent serial algorithm.



Challenges to parallel performance

These potential benefits must be offset by the many challenges to achieving good parallel performance.

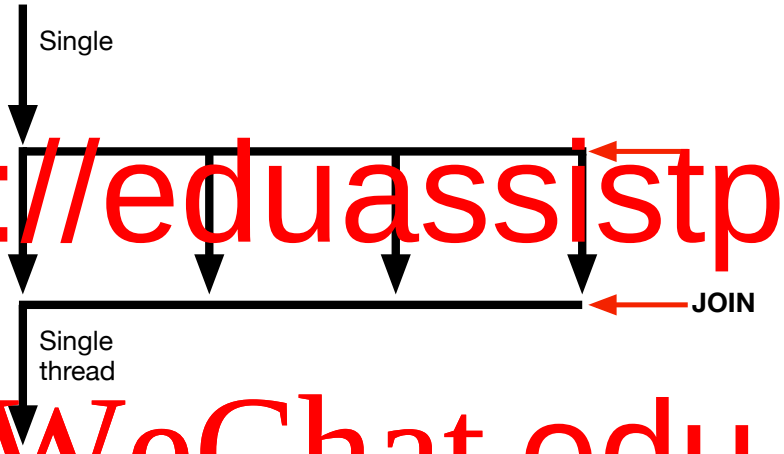
In Lec

- Hardware performance loss in maintaining **ency**
  when two cores repeatedly write to the same **ven**
  though they never read the other core's d

Over the coming lectures we will see two important, general challenges: **synchronisation** and **load balancing**.

Synchronisation

In the **fork-join** construct from last lecture, multiple threads complete before the main thread continues.



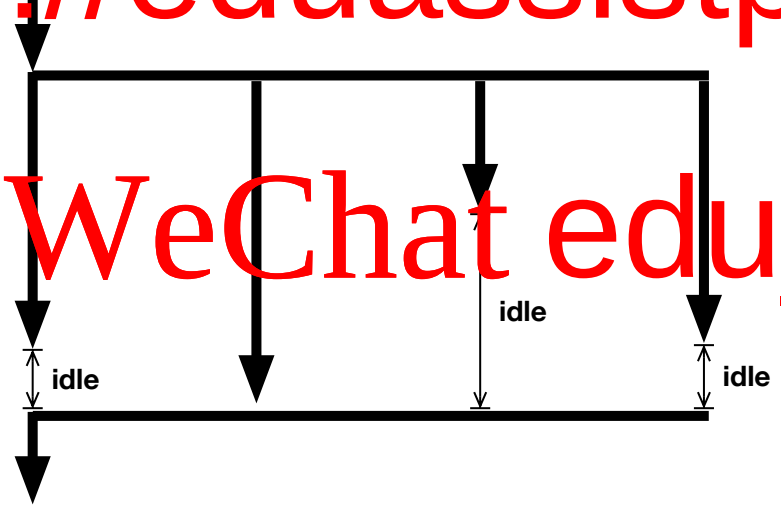
This **joining** requires resources.

- Main thread may repeatedly **probe** worker thread status.
- Alternatively, workers may **signal** their completion to main.
- An example of **synchronisation**.

Load balancing

A related issue is **load balancing**:

- What if the worker threads did not all finish **at the same time**?



This happens in the Mandelbrot set since each thread performs different numbers of calculations *[cf. last lecture; Lecture 13]*.

Parallel overheads

Even if these challenges could be overcome, there are inevitable **overheads**. For example:

- threads

- **Communication** between threads/processes not present in the serial equivalent.

- **Computation** not present in serial, e problem size between threads.

The impact may be small or large depending on parallel algorithm and hardware architecture.

Metrics for parallel performance

There are various measurements of parallel vs. serial performance.

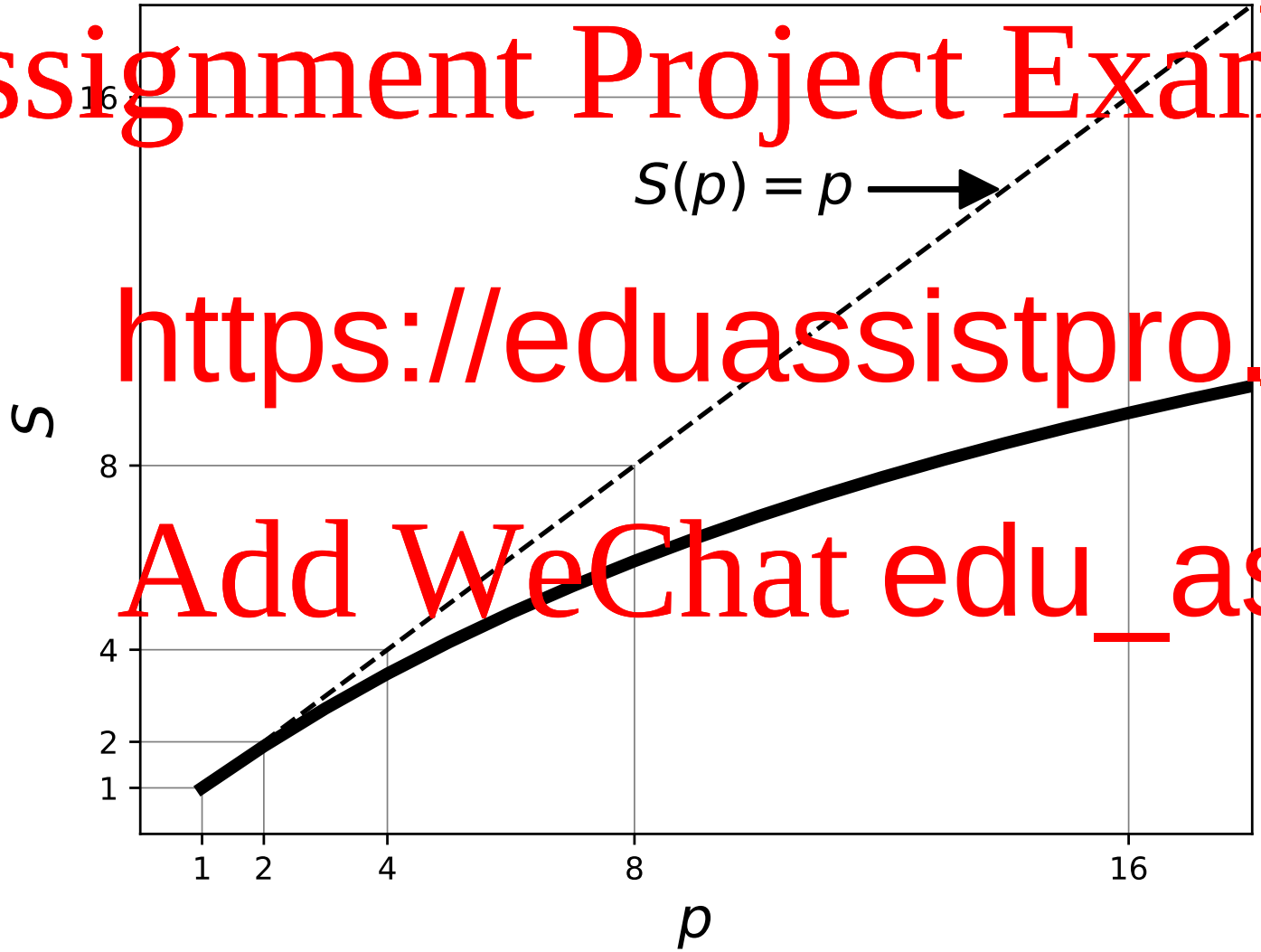
The m

- If the parallel version was p times f

$$t_p = \frac{1}{p} t_s \quad \Longrightarrow \quad S = \frac{}{\frac{1}{p} t_s} = p$$

- Rarely realised in practice due to **parallel overheads**.

Speedup example

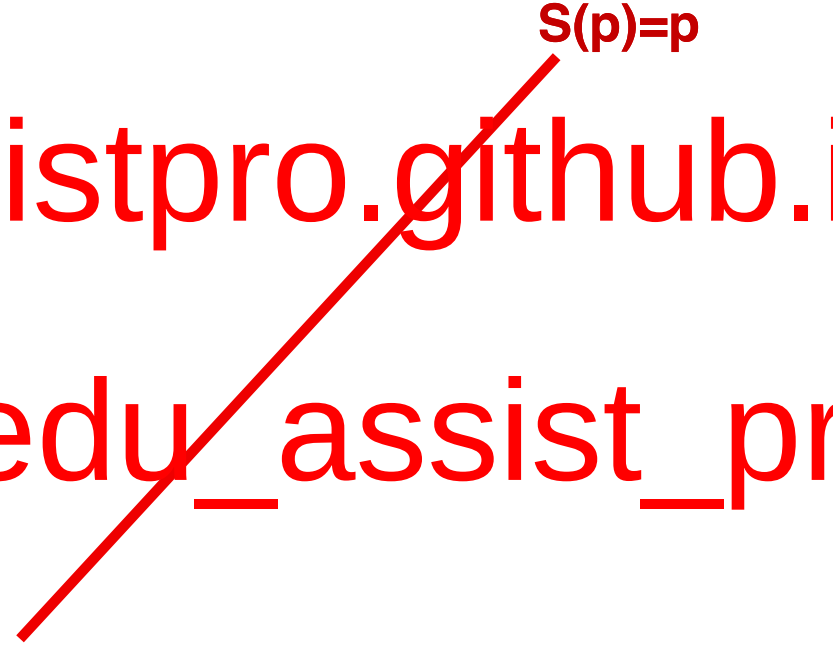


Superlinear speedup

$S(p) > p$ is possible.

- Usually due to memory
  c

This is



Example (right): Benchmark computational fluid dynamics algorithm.

However, this is rare - most commonly see $S(p) < p$.

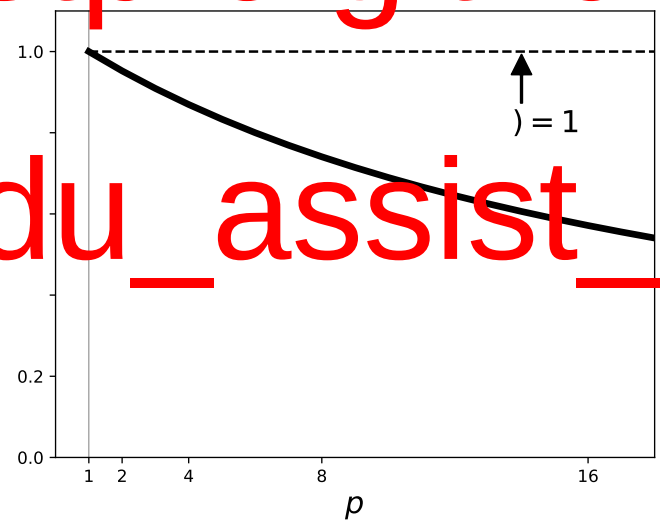
From *Parallel Programming in OpenMP*, Chandra *et al.* (Academic, 2001).

Efficiency

Another common parallel performance metric is the **efficiency** E:

$$\frac{t_s}{\quad} \quad \frac{S}{\quad}$$

- ‘
  corresponds to $E = 1$.
- Often expressed as a percentage: $E = 1 = 100\%$.
- Typically $E < 1$ due to parallel overheads.
- Superlinear speedup gives $E > 1$.



Models for parallel performance

Desirable to theoretically predict t_p for parallel algorithms.

- Select the 'best' without development and testing.
-

Chal

- Need to include *e.g.* memory cache
- Involve many unknown parameters requ
- Would need re-calibration for new hardw

However, even **simple** models can predict **trends**.

- **Parallel scaling**, which refers to the variation with p.

Amdahl's law

Suppose a fraction f of t_s cannot be parallelised.

$$t_s = ft_s + (1-f)t_s$$

$$t_p \qquad ft_s + \frac{(1-f)t_s}{} \qquad f + \frac{-}{}$$

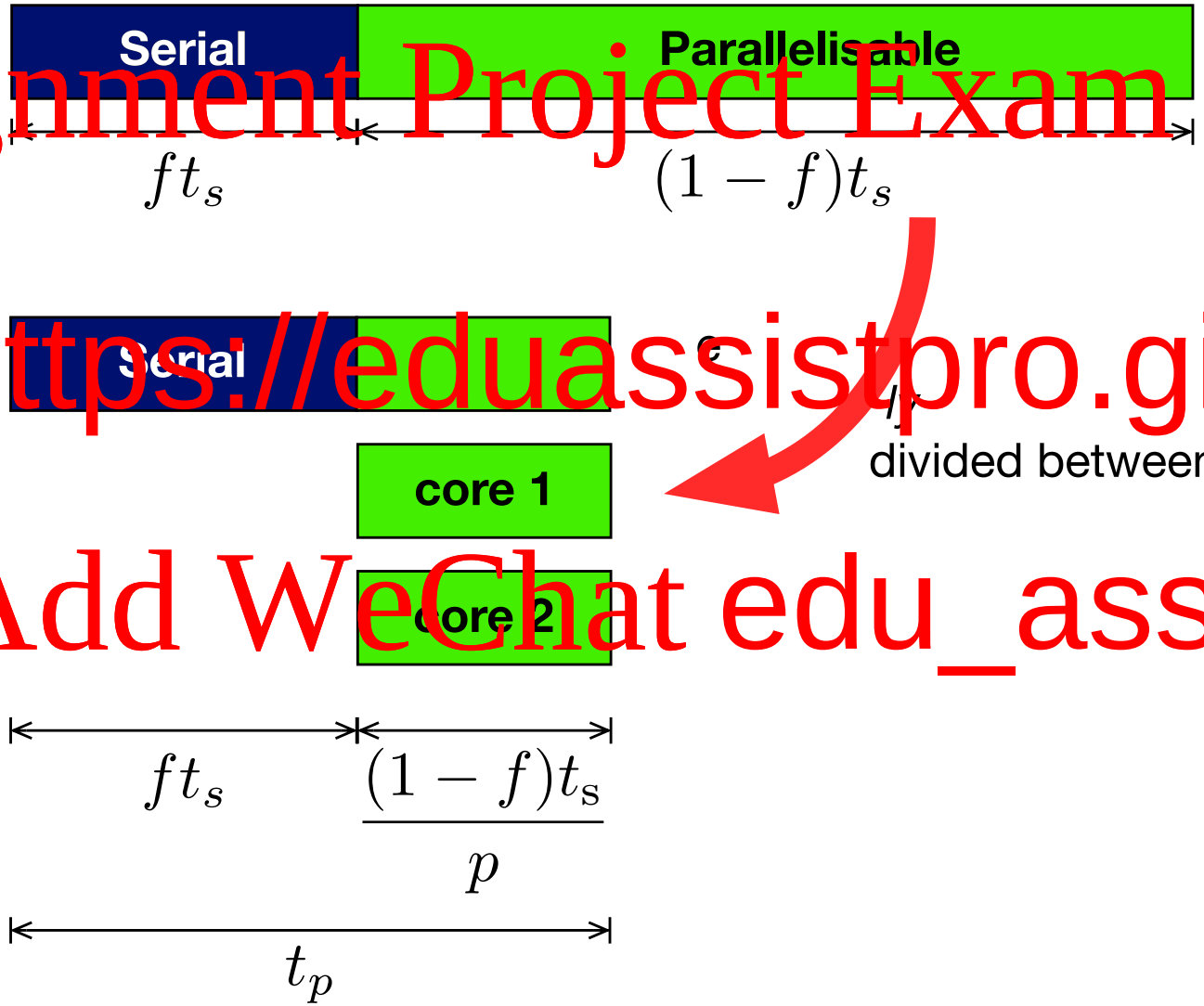
This is **Amdahl's law**[1] (1967).

For large p it predicts $S \leq \frac{1}{f}$ **regardless of** p.

- *e.g.* $f = 0.2$, maximum speedup of 5, **even for p**$=\infty$!

[1] Amdahl, *AFIPS Conference Proceedings* **30**, 483 (1967).

Schematic for Amdahl's law ($p = 3$)



Gustafson-Barsis law

However, Amdahl assumed t_s — and hence f — was fixed.

- Suppose instead n increases with p such that t_p is fixed.

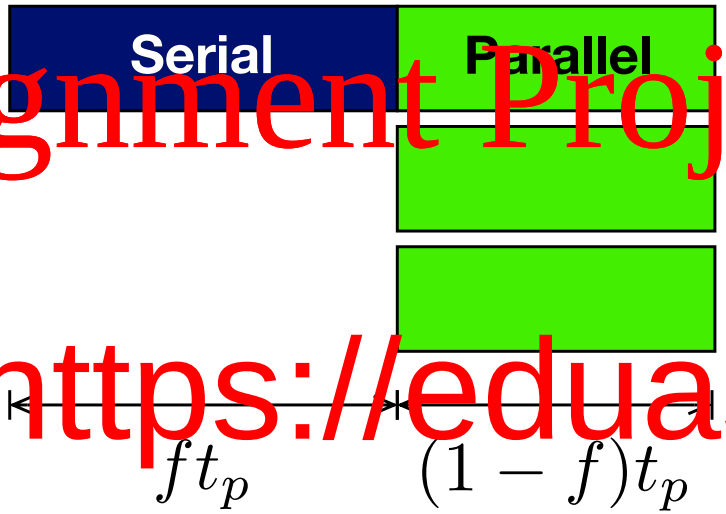
$$\implies S \leq f + p(1-f$$

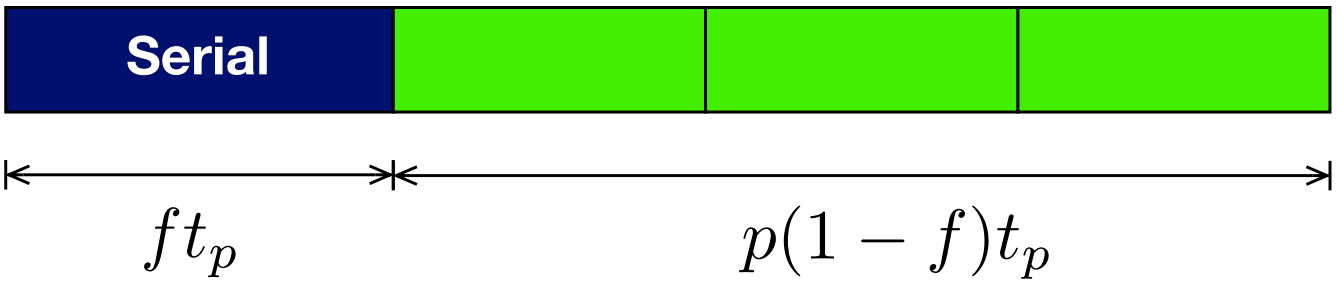
Now $S \leq (1-f)p$ for large p — no upper

This is the **Gustafson-Barsis law**, or just **Gustafson's law**[1].

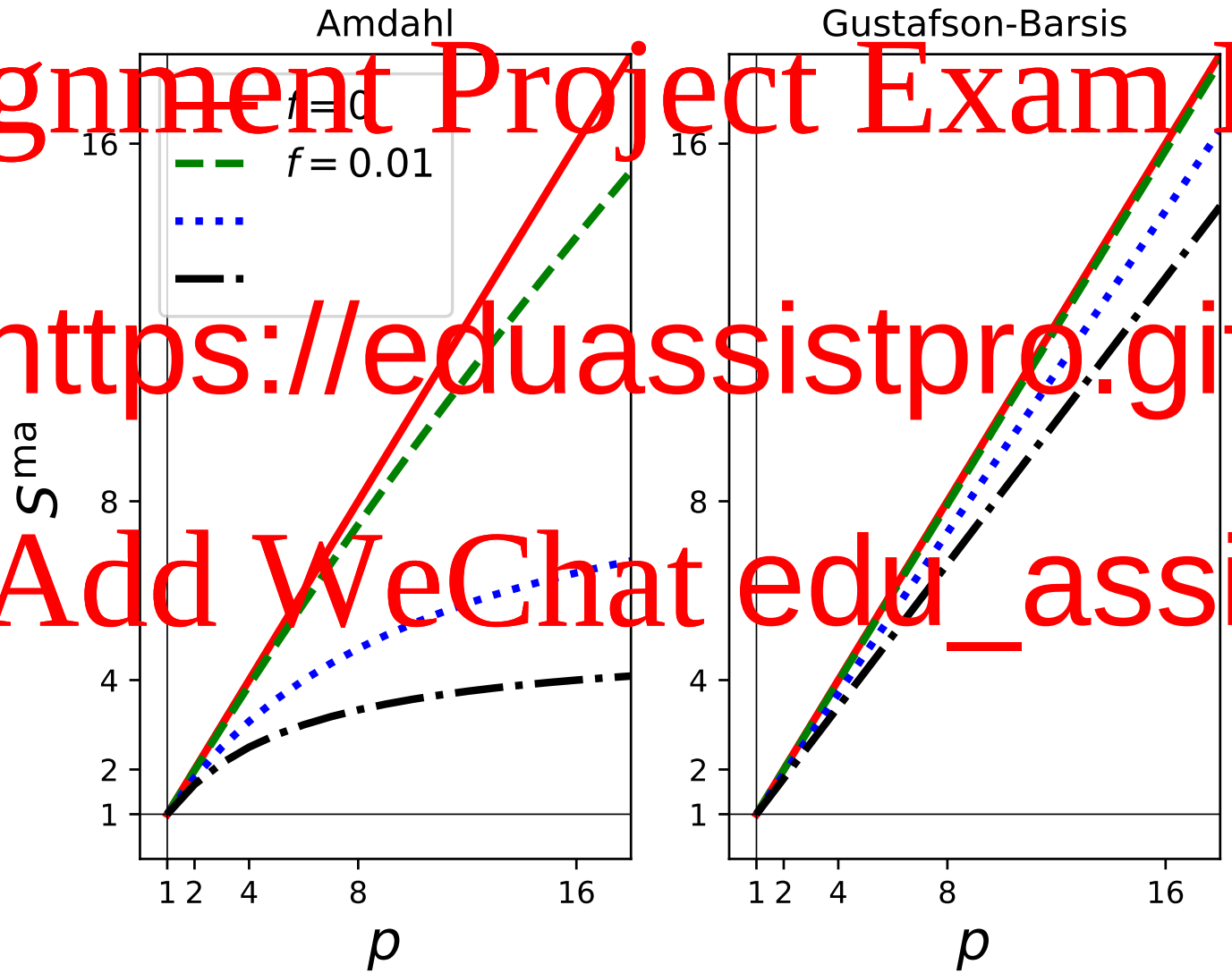
[1] Gustafson, *Comm. ACM* **31**, 532 (1988).

Schematic for Gustafson-Barsis law (p=3)





Amdahl *versus* Gustafson-Barsis



Weak *versus* strong scaling

The differences are encapsulated in **weak** *versus* **strong** scaling:

Strong scaling: Increasing p with n fixed.

Weak scaling: Increasing n with p.

- **Gustafson-Barsis law**.
- Have freedom to vary n.
- *e.g.* higher resolution meshes for scientific/engineering applications; more/larger layers in neural networks.

Summary and next lecture

Today we have looked at **parallel performance**:

- Two common metrics: **speedup** and **efficiency**.
- **arallel**
- **Gustafson-Barsis** law.
- Correspond to **strong** and **weak**

Next time we will look more closely at **data dependencies** in parallel loops.