Scalability of weather prediction models

Sam Hatfield

samuel.hatfield@ecmwf.int

ECMWF

Lecture overview

- Work, parallelism, and wall clock time
- Problem size and computational complexity
- Limits to parallelism
- Scalability, speedup, and parallel efficiency
- Strong scalability and Amdahl’s law
- Weak scalability and Gustafson’s law

Some definitions to start

Work

The total "amount" of computation required to solve some computational problem
E.g. # floating-point operations

Worker

An independent computational agent who can operate on a subset of the total work in parallel with its "colleagues"
E.g. core, node, thread

Wall clock time

The total time a particular program running in a particular context takes to solve that same problem
Note: sometimes we care about energy consumption, not wall clock time

And one more...

Problem size

A single parameter of greatest interest that determines the overall size of a particular computational problem

Usually denoted by *n*
E.g. kilometric resolution, number of vertical levels, spectral truncation etc.

How does *work* depend on *problem size*?

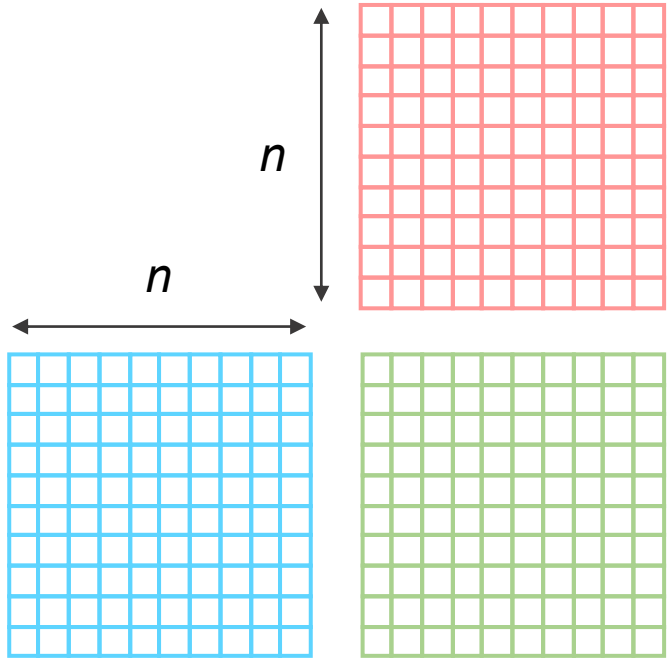
It depends on the problem!

E.g. naïve matrix multiplication of $n \times n$ matrices A = B * C:

Work = $2n^3 - n^2$ floating-point operations

We say this problem has *complexity* $O(n^3)$

It is difficult to derive expressions of complexity for real-world applications like NWP models



How does *wall clock time* depend on *work?*

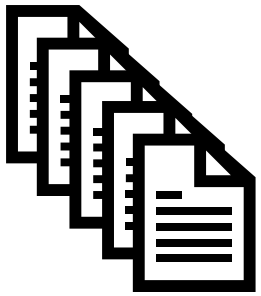
In general, in an ideal world:

$$\text{Wall clock time} = \frac{\text{Total amount of work}}{\text{Time taken to process one unit of work} \times \text{number of workers}}$$

E.g.: $$\text{Wall clock time} = \frac{\text{Number of FLOPs}}{\text{FLOPs/s} \times \text{number of nodes}}$$

Example problem

I am a teacher with **12 homework assignments**



I have **4 teaching assistants**

Each assistant takes **30 minutes to mark 1 assignment**

$$\text{Wall clock time} = \frac{12 \text{ assignments}}{2 \text{ assignments / hour} \times 4 \text{ assistants}} = 1.5 \text{ hours}$$

Why is this expression only "*ideal*" ?

In the real world, two main issues limit the benefits of parallelism:

1. Load imbalance

We may not be able to divide up work equally among the workers

2. Inherently serial parts (e.g. communication)

Not everything can be parallelised

Load imbalance

Suppose one assistant was close to burnout so could only mark max. 1 assignment...

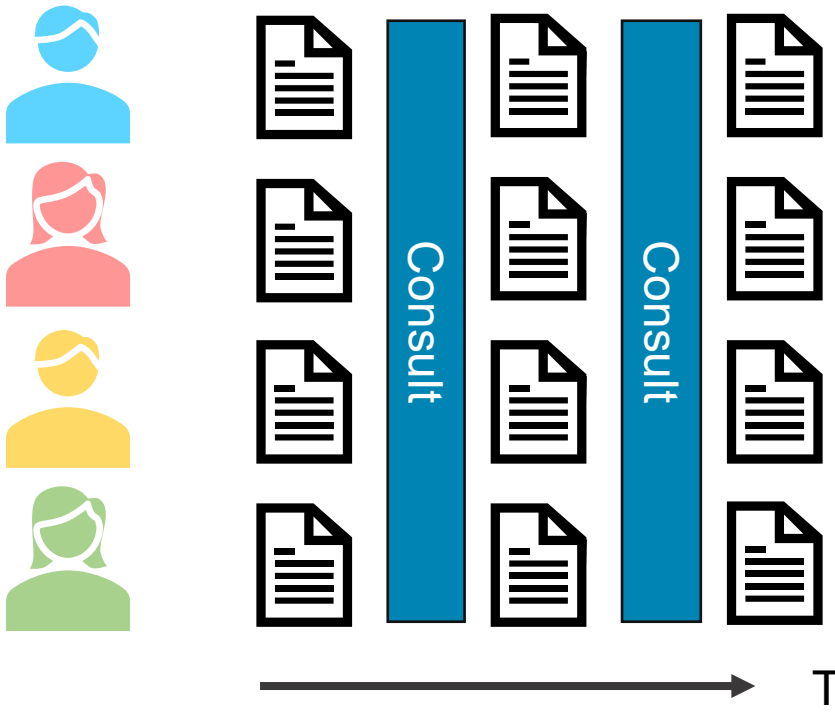
Total time 2 hours

We now have a load imbalance

The previous expression no longer holds

Serial

Suppose each assistant needs to consult all the others after every assignment



Serial parts are *bottlenecks*

The previous expression no longer holds

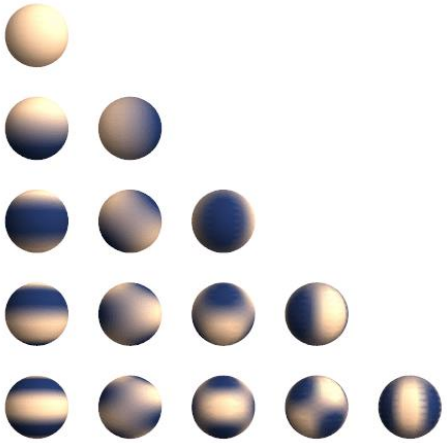
Scalability

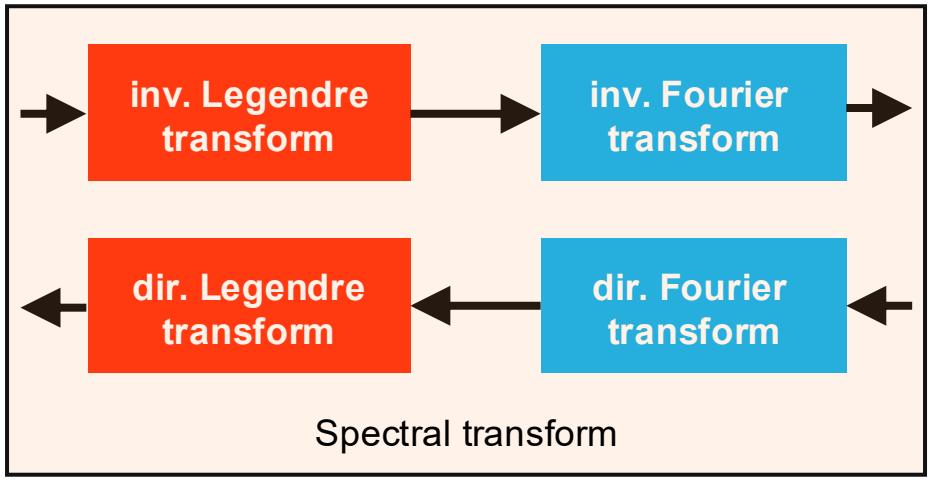
How does an application perform (*wall clock time*) as we change the allocation of parallel resources *(# workers)?*

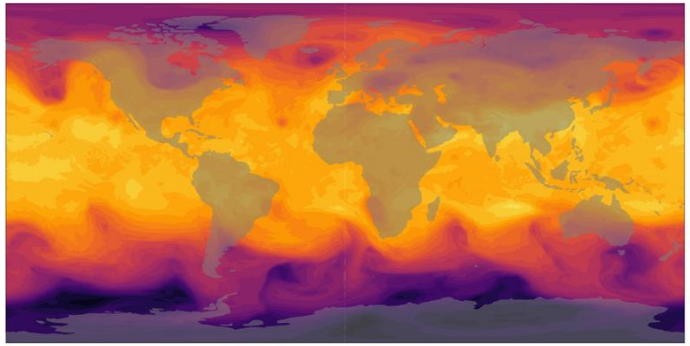
We can either have:

strong scalability *work* is fixed
or
weak scalability *work* increases with *# workers*

Our scalability guinea pig: ecTrans







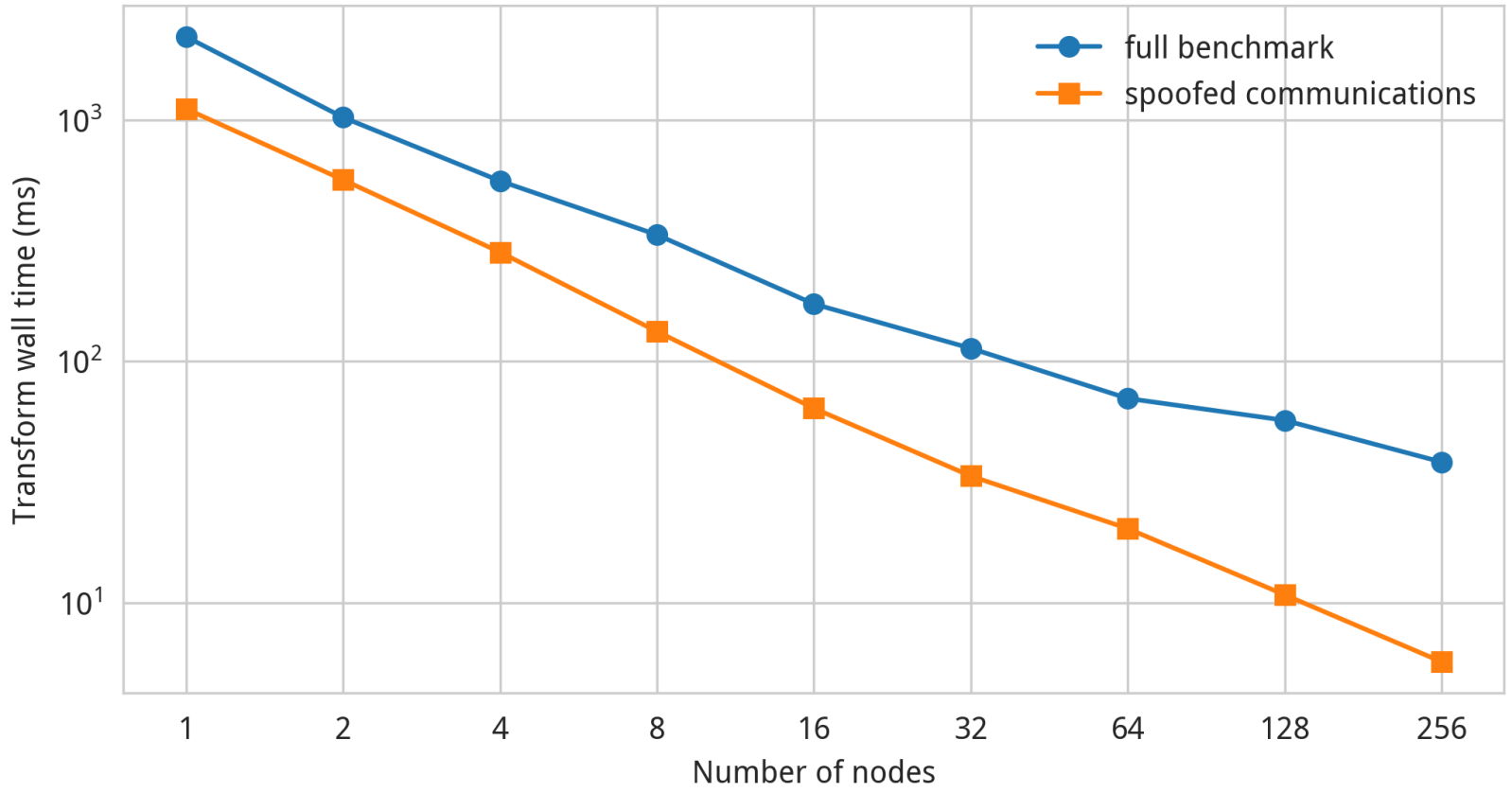
Spectral space, used by:
- Time-stepping
- Horizontal diffusion
- Linear terms

github.com/ecmwf-ifs/ectrans

ecTrans

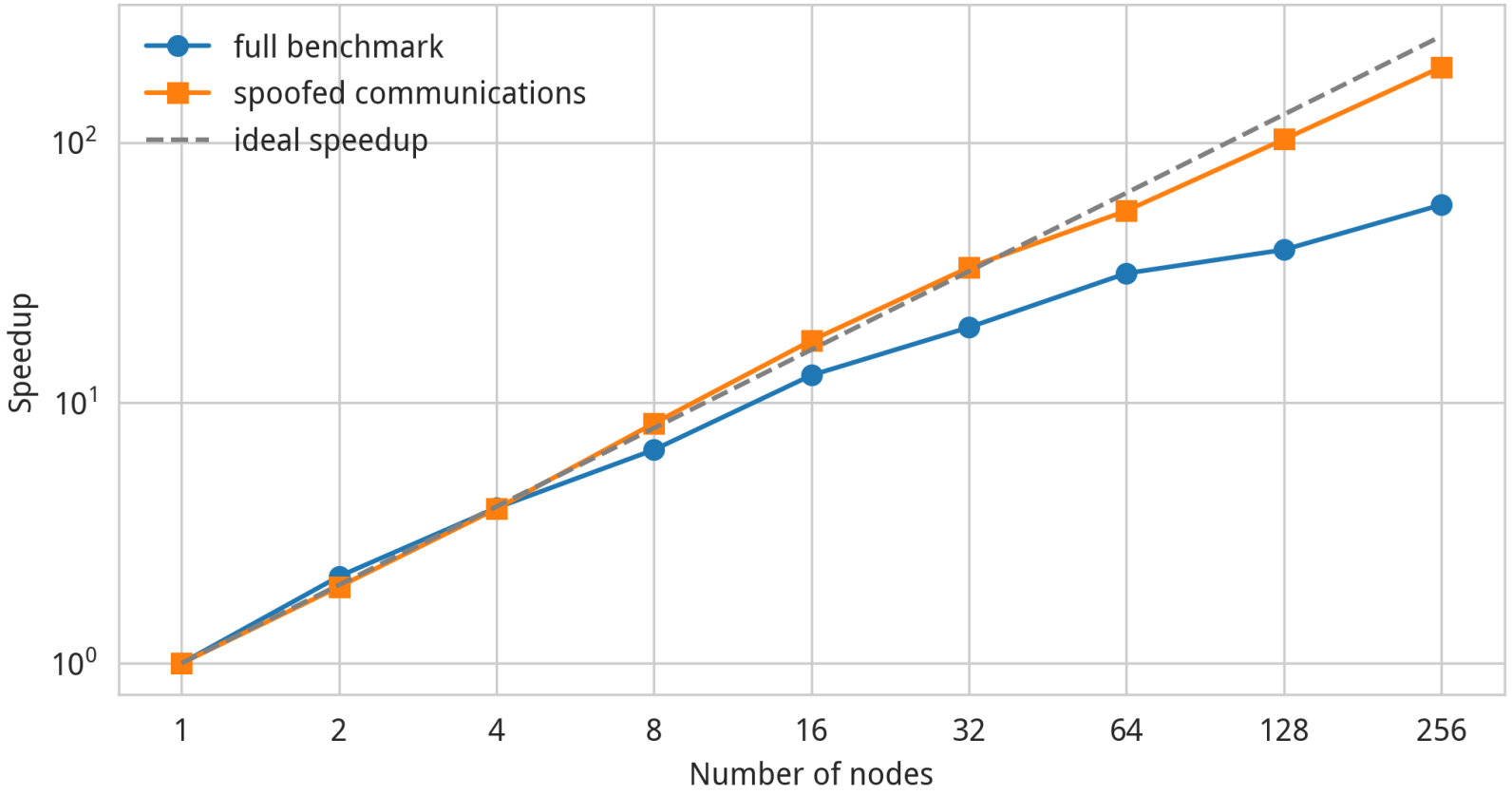
Grid point space, used by:
- Physical parametrisations
- Nonlinear terms

Scalability of ecTrans at TCO999 (~10 km)



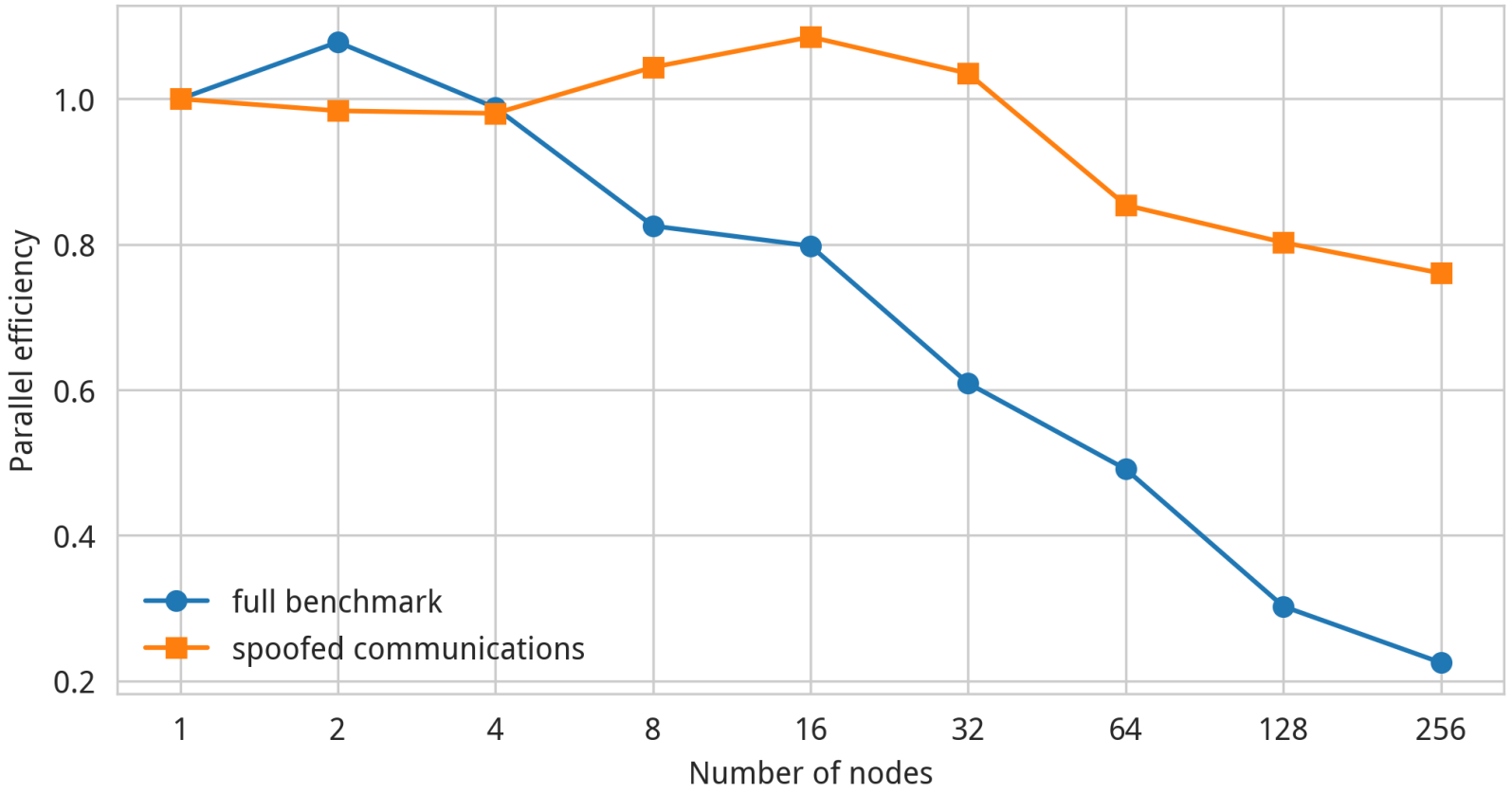
"spoofed communications" = all communications are skipped (receive buffers set to all zeros)

Speedup



$$S(n) = \frac{\text{Walltime(1 node)}}{\text{Walltime}(n \text{ nodes})}$$

Parallel efficiency



$$P(n) = \frac{S(n)}{n}$$

In general, high-performance applications are composed of **parallelisable** parts and **serial** parts

$$\text{Total wall time} = \text{Time spent in parallelisable part} + \text{Time spent in serial part}$$

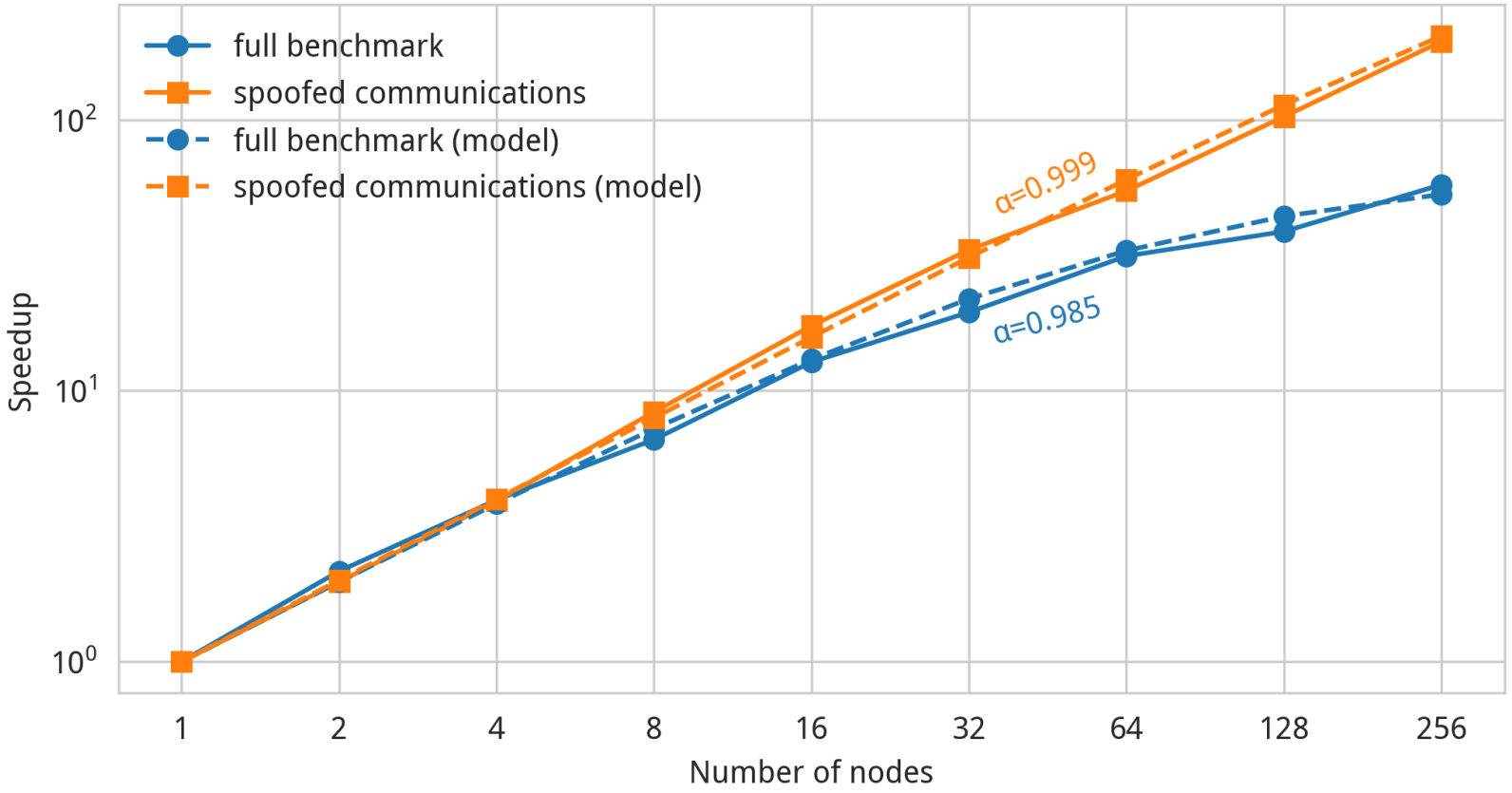
This part scales like 1 ÷ (number of nodes)

This part **doesn't scale at all**

Suppose that we measure performance on **1** node, and want to predict **speedup** from running on ***n*** nodes

$$S_{\text{theoretical}}(n) = \frac{\text{Wall time on 1 node}}{\text{Predicted wall time on } n \text{ nodes}} = \frac{T(1)}{T(n)} = \frac{T(1)}{\frac{\alpha T(1)}{n} + (1-\alpha)T(1)} = \frac{1}{\frac{\alpha}{n} + 1 - \alpha}$$

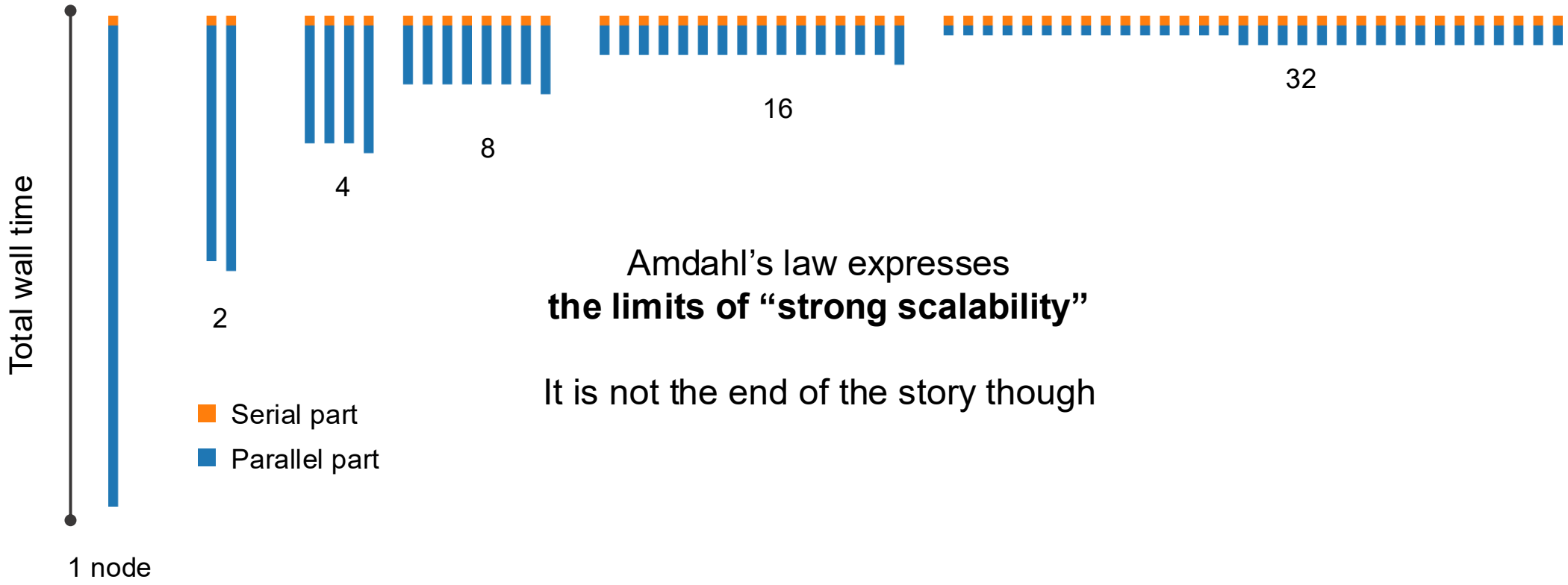
It depends on what ratio of time is spent in parallelisable parts ($\boldsymbol{\alpha}$) and serial parts ($\mathbf{1-\alpha}$)



$$S_{\text{theoretical}}(n) = \frac{1}{\frac{\alpha}{n} + 1 - \alpha}$$

is called
Amdahl's Law

What's going on?
Diminishing returns!

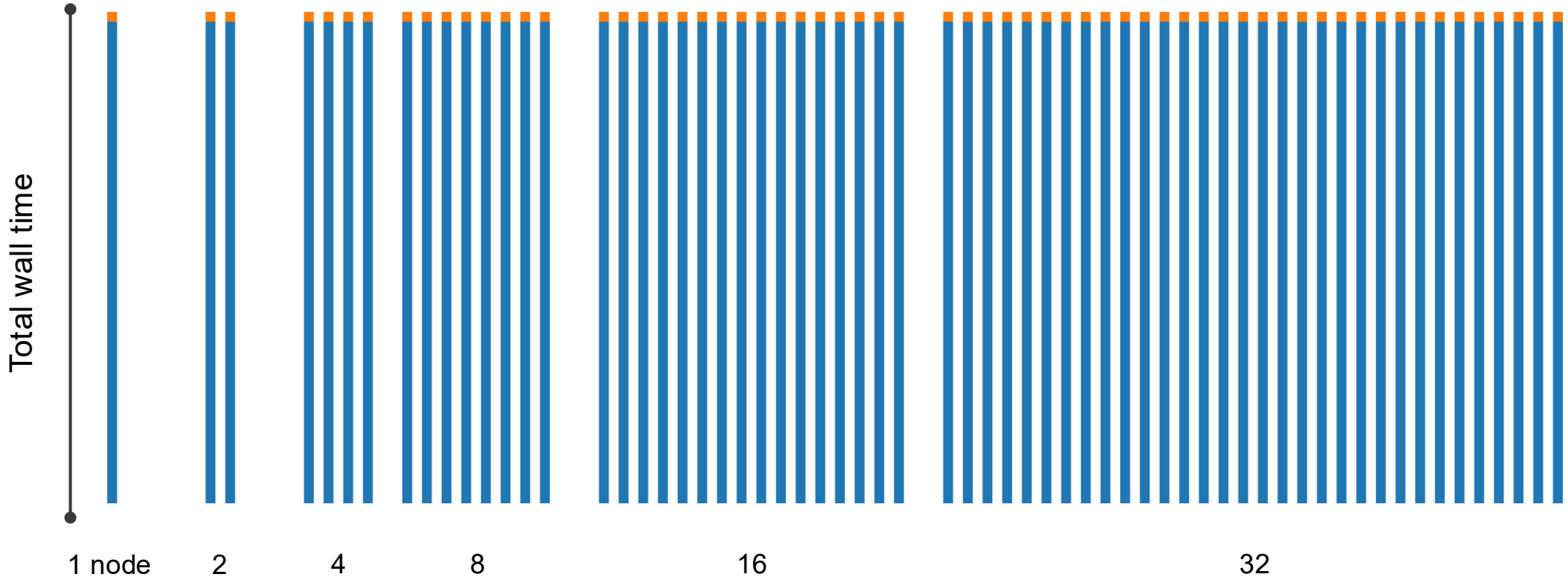


Amdahl's law expresses
the limits of "strong scalability"

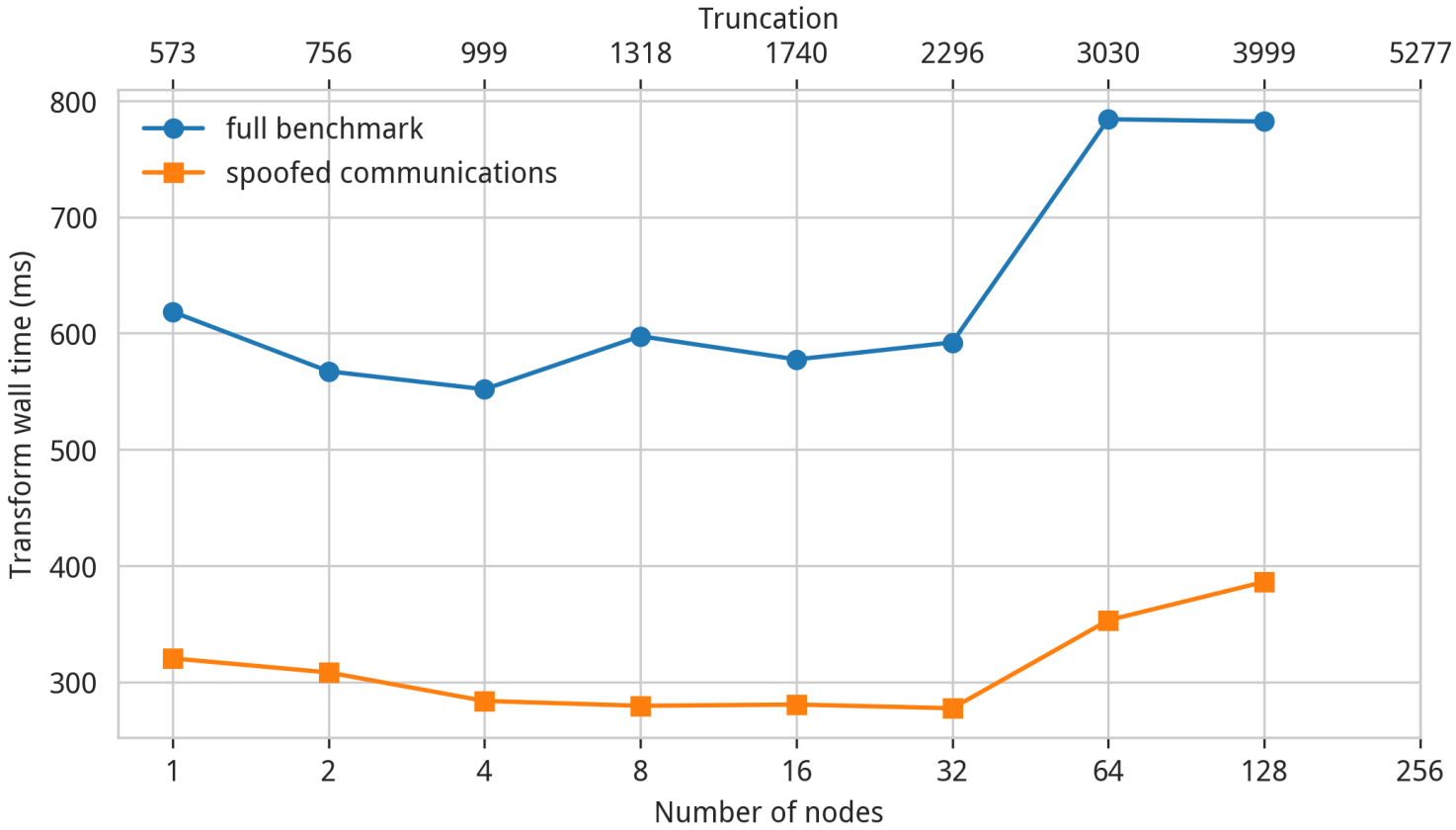
It is not the end of the story though

Solution:

Increase the problem size



Weak scalability of ecTrans



Here we assume that:

Work $\propto$ truncation$^{2.5}$

We can now derive *another* expression for speed-up:

Suppose that we measure performance on ***n*** nodes, and want to calculate the **speedup** compared with *if we had only run one **1** node*

$$S_{\text{theoretical}}(n) = \frac{\text{Predicted wall time on 1 node}}{\text{Wall time on } n \text{ nodes}} = \frac{T(1)}{T(n)} = \frac{\big(n\beta + (1-\beta)\big)T(n)}{T(n)} = n\beta + 1 - \beta$$

It depends on what ratio of time *for the parallelised case* is spent in parallelisable parts (***β***) and serial parts (**1 − *β***)

$$S_{\text{theoretical}}(n) = n\beta + 1 - \beta$$

is called **Gustafson's law**

It states that *the "speedup" that really matters is the speedup compared with running the same problem on only a single node*

and

If you can keep β constant by scaling the work in proportion with the # nodes, you can achieve serious meaningful speedups, despite inherent limits to parallelisability

Scalability summary

Strong scalability
Work is fixed
Governed by Amdahl's law
Difficult to achieve

Weak scalability
Work increases with # *workers*
Governed by Gustafson's law
Easi*er* to achieve

Sam Hatfield

samuel.hatfield@ecmwf.int