Performance metrics

How is my parallel code performing and scaling?

Reusing this material

Performance metrics

- Fundamental measurement is the runtime *T*
  - we can vary the number of processes *P* or the problem size *N*
  - *N* measures the amount of computation, e.g. number of grid points

- Speed up
  - typically $S(N,P) < P$

$$S(N,P) = \frac{T(N,1)}{T(N,P)}$$

- Parallel efficiency
  - typically $E(N,P) < 1$

$$E(N,P) = \frac{S(N,P)}{P} = \frac{T(N,1)}{P\,T(N,P)}$$

- Serial efficiency
  - typically $E(N) <= 1$

$$E(N) = \frac{T_{best}(N)}{T(N,1)}$$

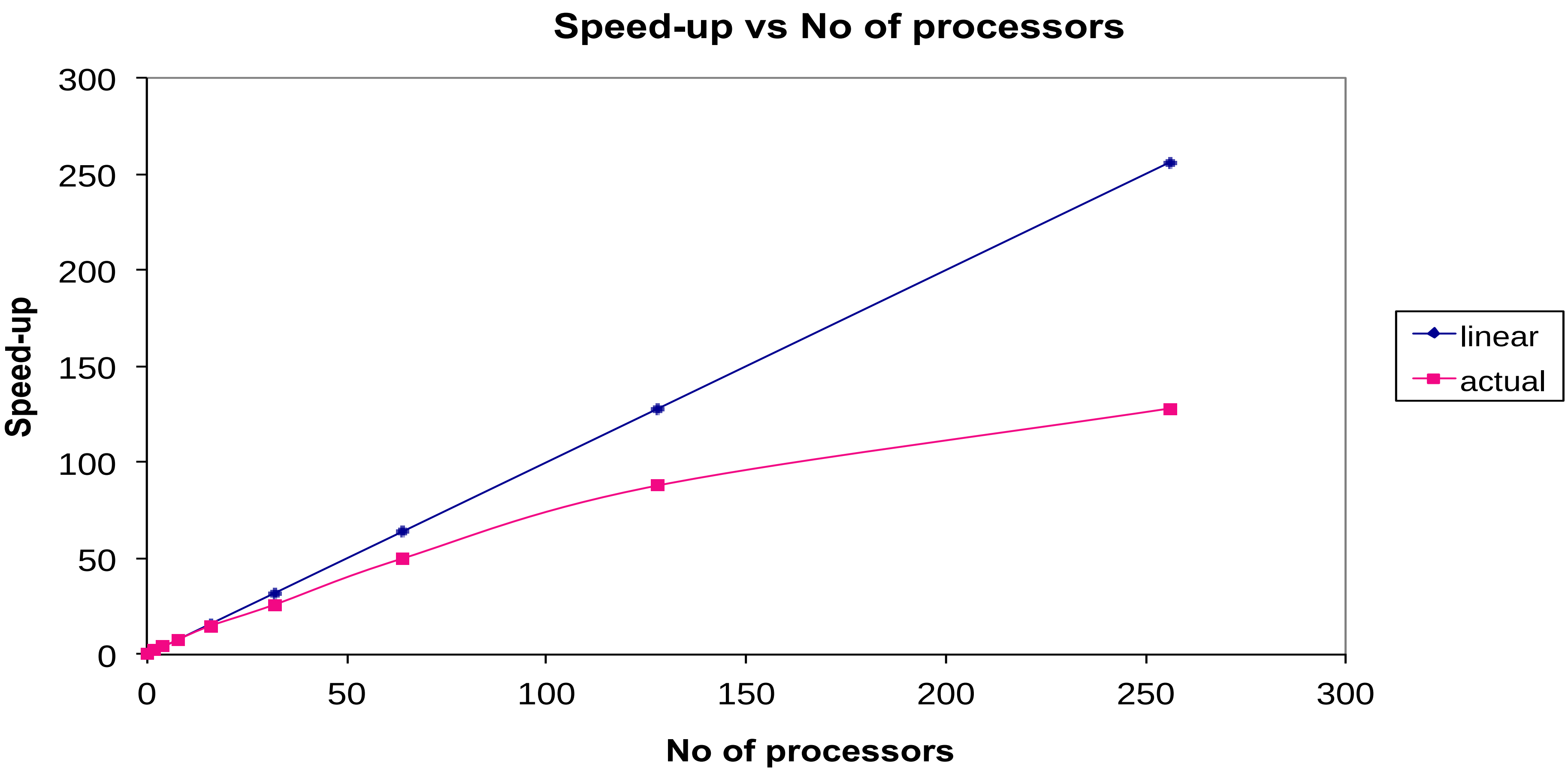
Scaling

- *Scaling* is how the performance of a parallel application changes as the number of processors is increased

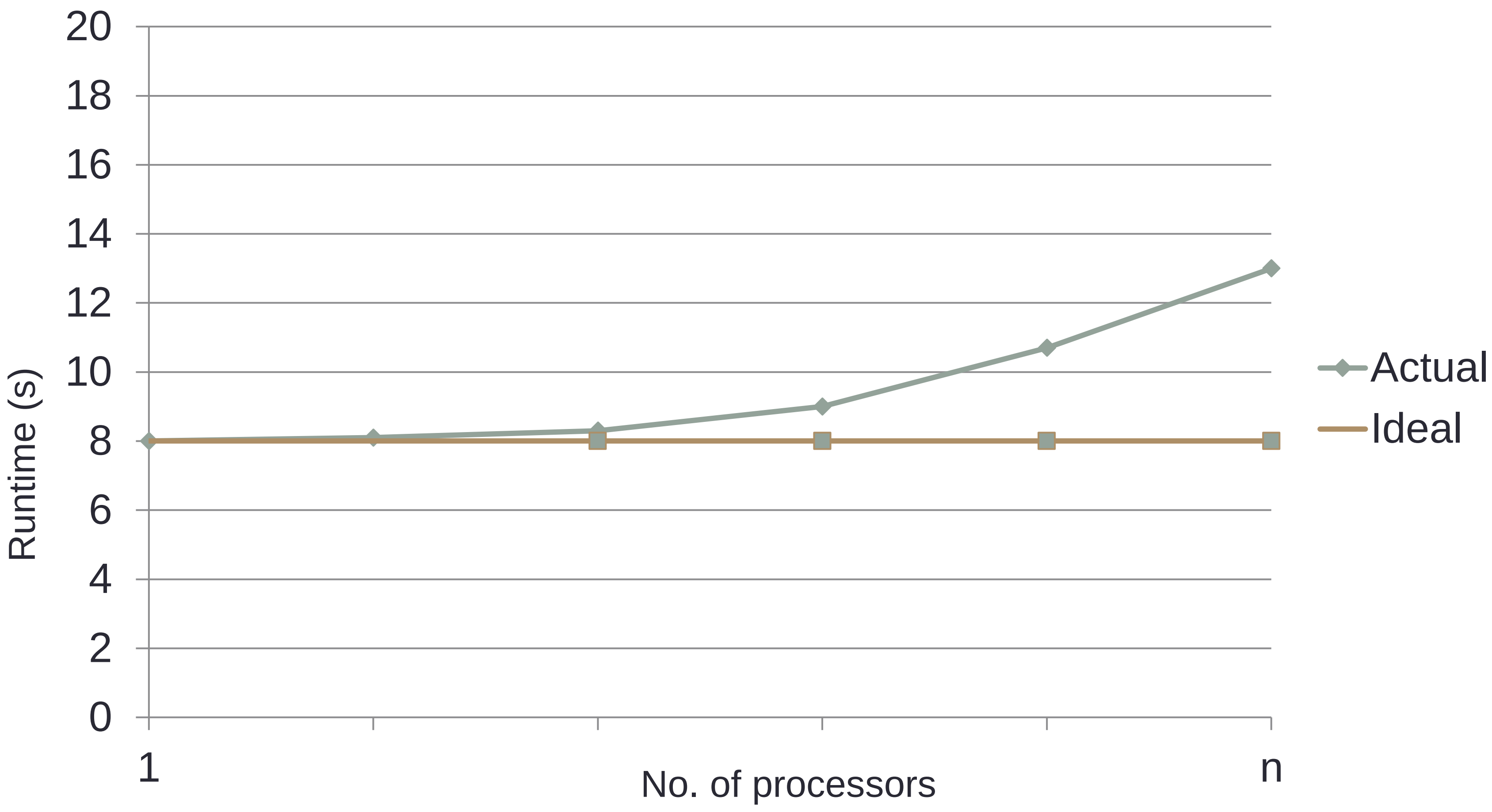
- There are two different types of scaling:
  - *Strong Scaling* – total problem size stays the same as the number of processors increases
  - *Weak Scaling* – the problem size increases at the same rate as the number of processors, keeping the amount of work per processor the same

- Strong scaling is generally more useful and more difficult to achieve than weak scaling

Strong scaling



Weak scaling



Modelling speedup

- A typical program has two categories of components
  - inherently serial sections: can’t be run in parallel
  - potentially parallel sections

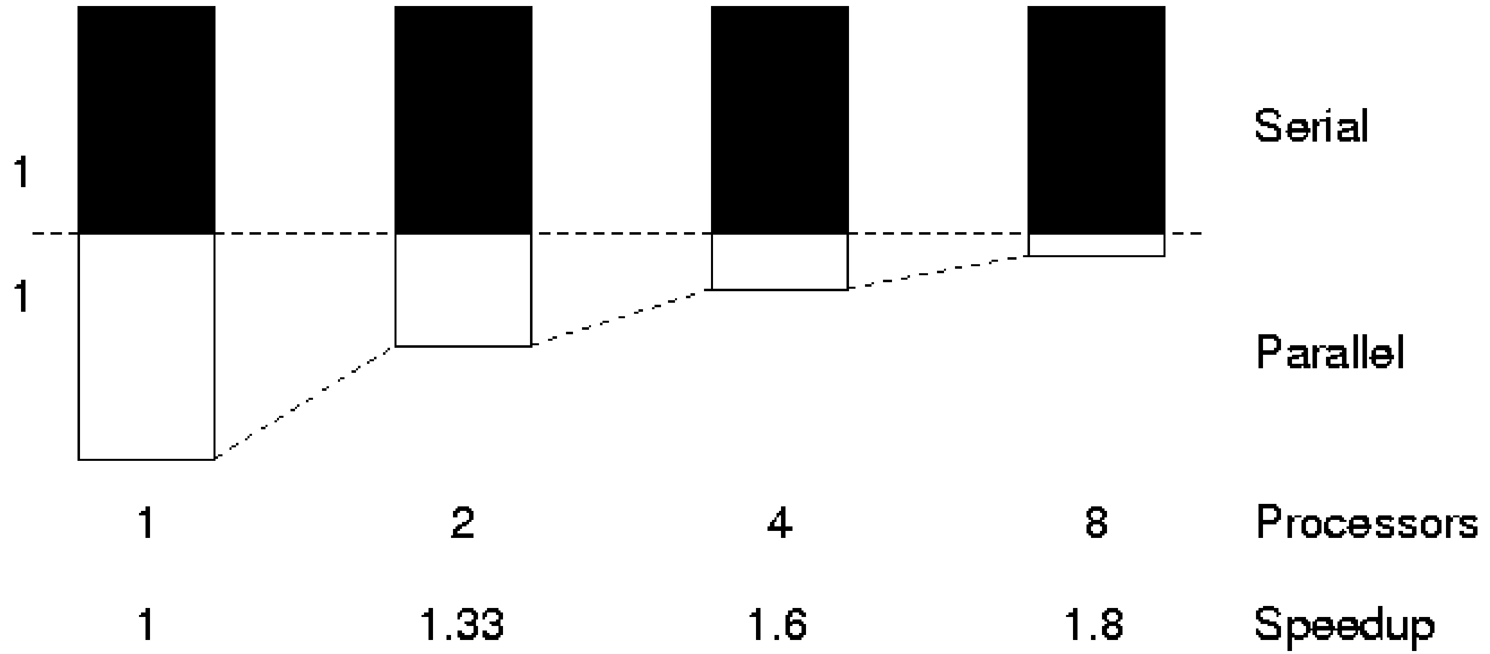
- Classic examples of serial
  - initialisation or file IO (in parallel just do it from a single process)

- In practice, “serial” includes all parallel overheads
  - any operation that does not benefit from parallelisation
  - this includes reduction operations, halo swapping, ...

The serial section of code

"The performance improvement to be gained by parallelisation is limited by the proportion of the code which is serial"

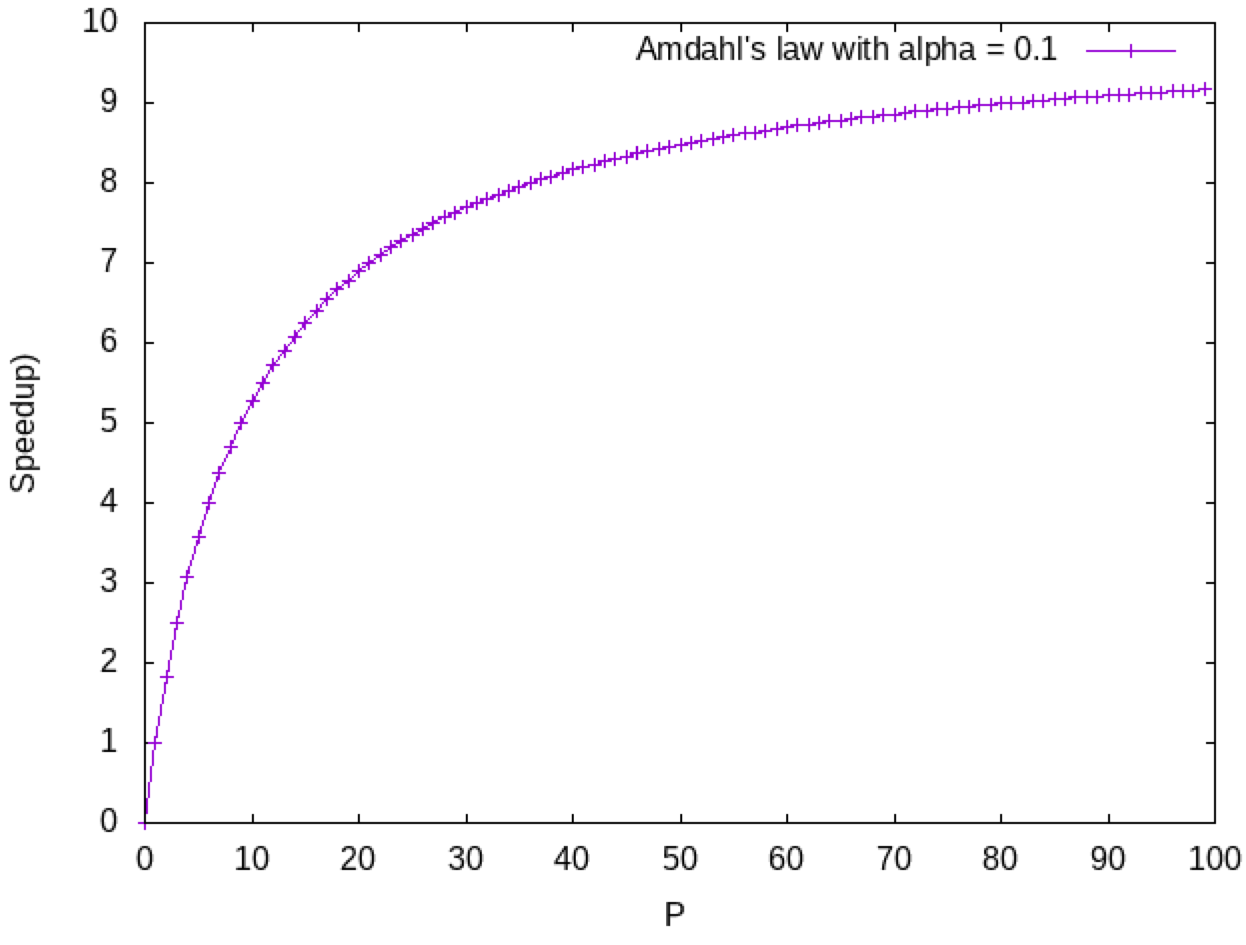
Gene Amdahl, 1967



Amdahl's law

- Assume that a fraction, α, is completely serial
- Parallel runtime $$T(N,P) = \alpha\, T(N,1) + \frac{(1-\alpha)\, T(N,1)}{P}$$
  - assuming parallel part is 100% efficient
- Parallel speedup $$S(N,P) = \frac{T(N,1)}{T(N,P)} = \frac{P}{\alpha P + (1-\alpha)}$$
- We are fundamentally limited by the serial fraction
  - For $\alpha = 0$, $S = P$ as expected (i.e. *efficiency* = 100%)
  - Otherwise, speedup limited by $1/\alpha$ for any P
    - for $\alpha = 0.1$; 1/0.1 = 10 therefore 10 times maximum speed up
    - for $\alpha = 0.1$; $S(N, 16) = 6.4$, $S(N, 1024) = 9.9$

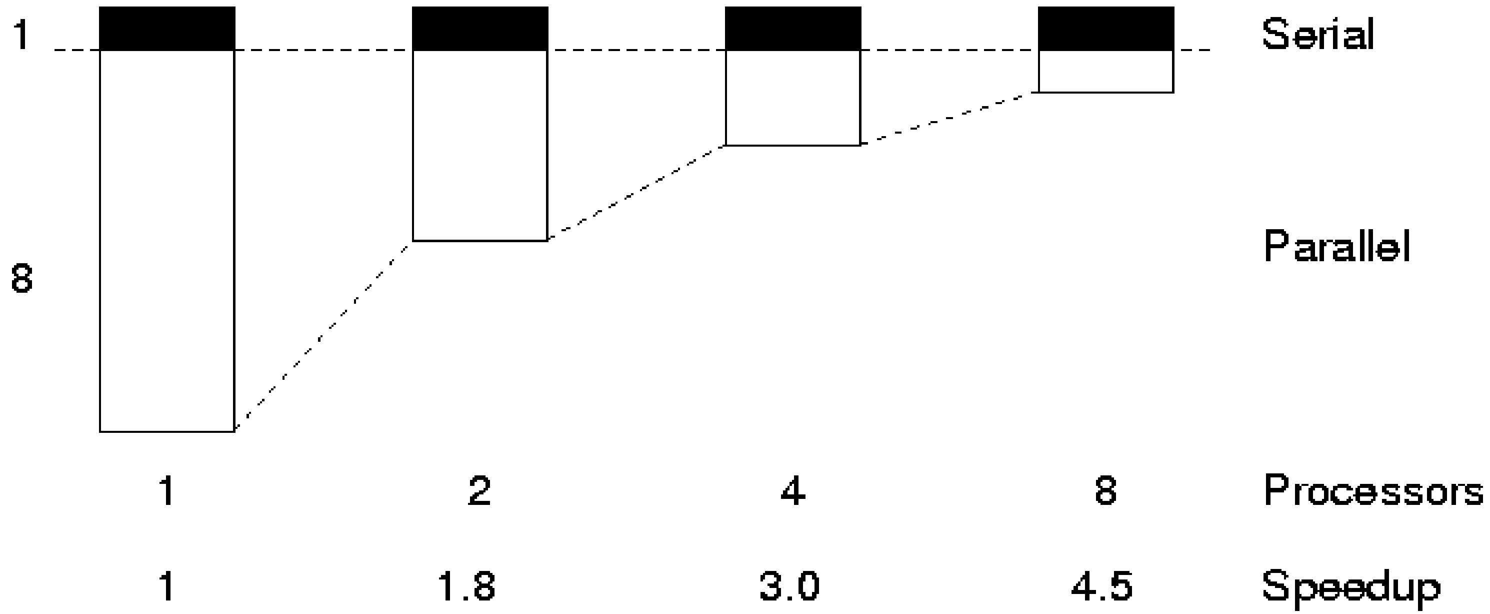
Example



Amdahl's law with alpha = 0.1
Speedup)
P
0
1
2
3
4
5
6
7
8
9
10
0
10
20
30
40
50
60
70
80
90
100

Gustafson's Law

- We need larger problems for larger numbers of CPUs



- Whilst we are still limited by the serial fraction, it becomes less important

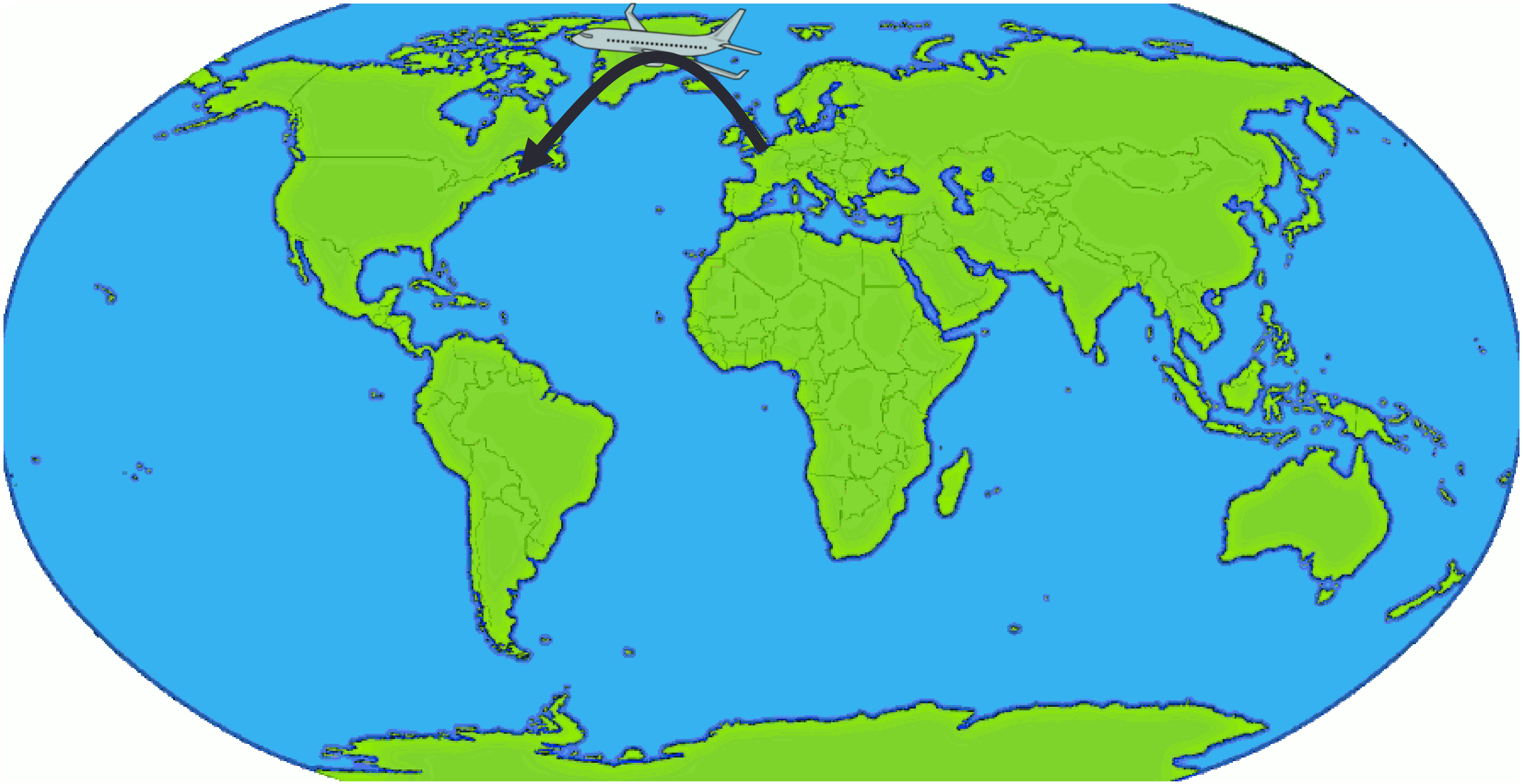
Utilising Large Parallel Machines

- Assume parallel part scales with *N*, serial part constant
  - i.e. parallel part is O(*N*), serial is O(1)
  - time
$$\begin{aligned} T(N,P) &= T_{serial}(N,P) + T_{parallel}(N,P) \\ &= \alpha T(1,1) + \frac{(1-\alpha)NT(1,1)}{P} \end{aligned}$$
  - speedup
$$S(N,P) = \frac{T(N,1)}{T(N,P)} = \frac{\alpha + (1-\alpha)N}{\alpha + (1-\alpha)\frac{N}{P}}$$
- Scale problem size with *P*, i.e. set *N* = *P* (weak scaling)
  - speedup
$$S(P,P) = \alpha + (1-\alpha)P$$
  - efficiency
$$E(P,P) = \frac{\alpha}{P} + (1-\alpha)$$

Gustafson's Law

- If you can increase the amount of work done by each process / task, the serial component will not dominate
  - increase the problem size *N* to maintain scaling
  - this can be in terms of
    - increasing the overall problem size
    - adding extra complexity
- For instance, $a = 0.1$: $S(N, 16) = 6.4$, $S(N, 1024) = 9.9$
  - using strong scaling:
    - $S(16\,N, 16) = 14.5$ $E(16\,N, 16) = 0.91$
    - $S(1024\,N, 1024) = 921.7$ $E(1024\,N, 1024) = 0.9001$

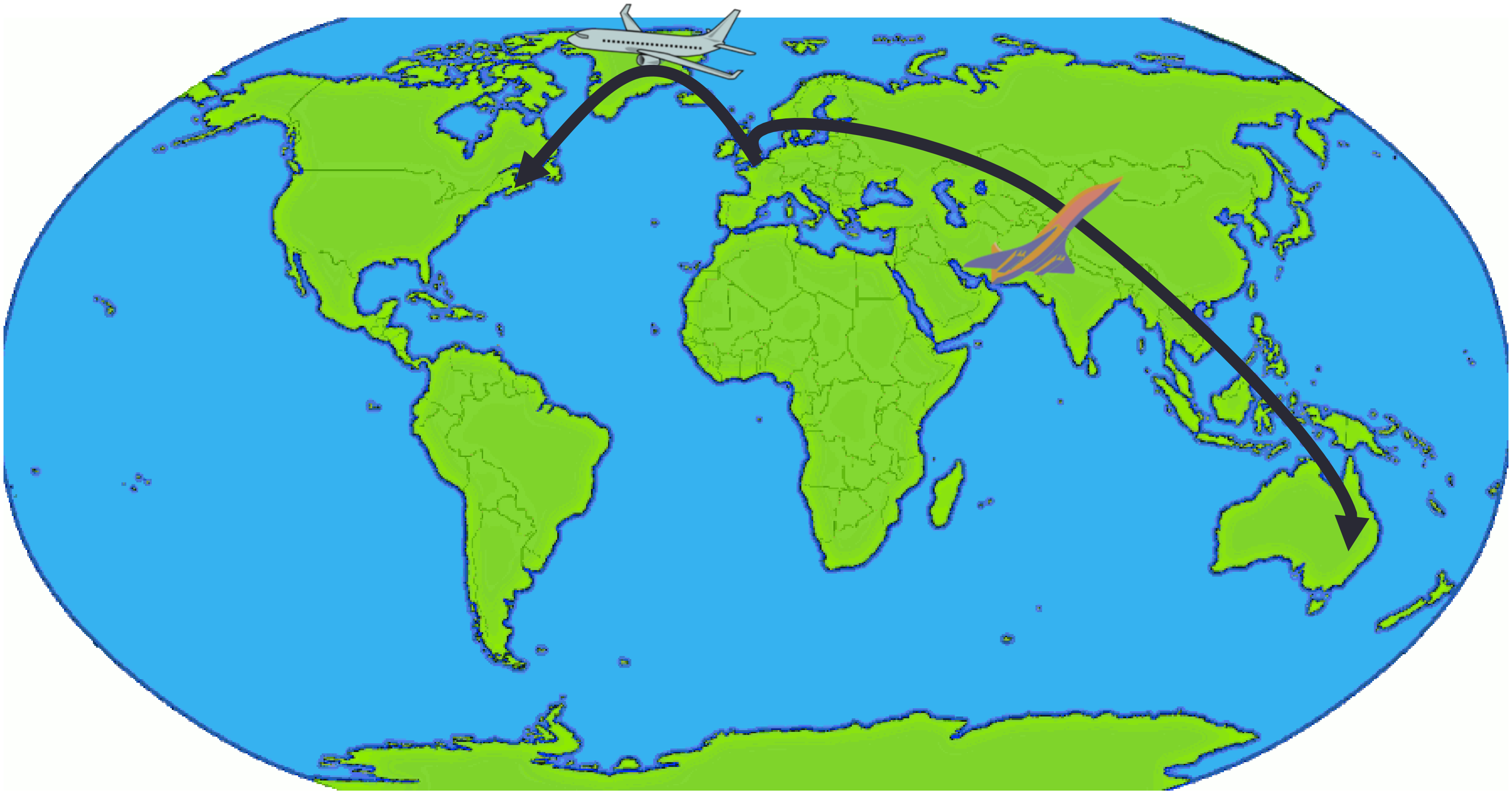
Analogy: Flying London to New York



Buckingham Palace to Empire State

- By Jumbo Jet
  - distance: 5600 km; speed: 700 kph
  - time: 8 hours ?
- No!
  - 1 hour by tube to Heathrow + 1 hour for check in etc.
  - 1 hour immigration + 1 hour taxi downtown
  - fixed overhead of 4 hours; total journey time: 4 + 8 = 12 hours
- Triple the flight speed with Concorde to 2100 kph
  - total journey time = 4 hours + 2 hours 40 mins = 6.7 hours
  - speedup of 1.8 not 3.0
- Amdahl's law!
  - a = 4/12 = 0.33; max speedup = 3 (i.e. 4 hours)

Flying London to Sydney



Buckingham Palace to Sydney Opera

- By Jumbo Jet
  - distance: 16800 km; speed: 700 kph; flight time; 24 hours
  - serial overhead **stays the same:** total time: 4 + 24 = 28 hours

- Triple the flight speed
  - total time = 4 hours + 8 hours = 12 hours
  - speedup = 2.3 (as opposed to 1.8 for New York)

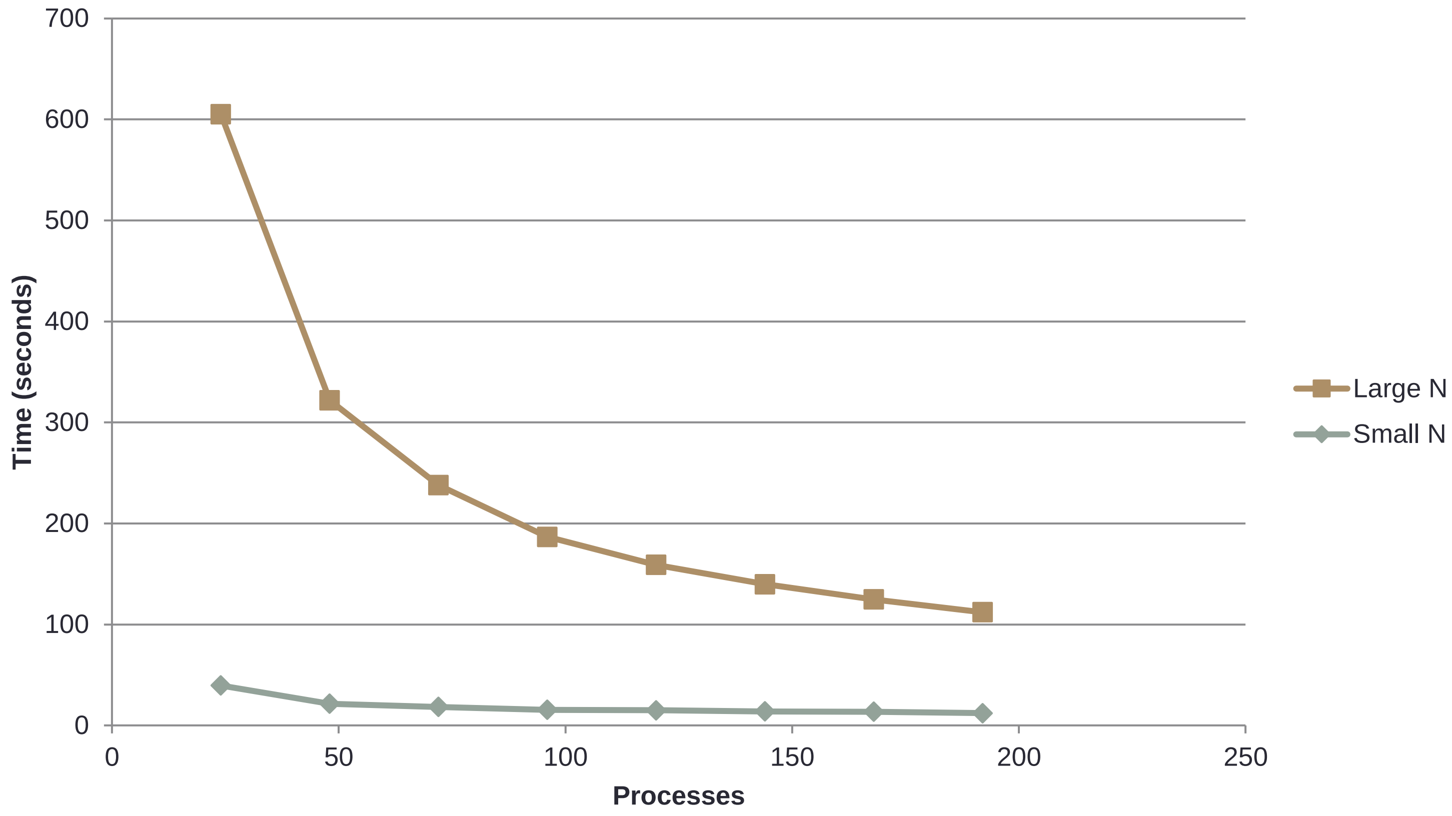
- Gustafson's law!
  - bigger problems scale better
  - increase **both** distance (i.e. *N*) **and** max speed (i.e. *P*) by three
  - maintain same balance: 4 "serial" + 8 "parallel"

Plotting

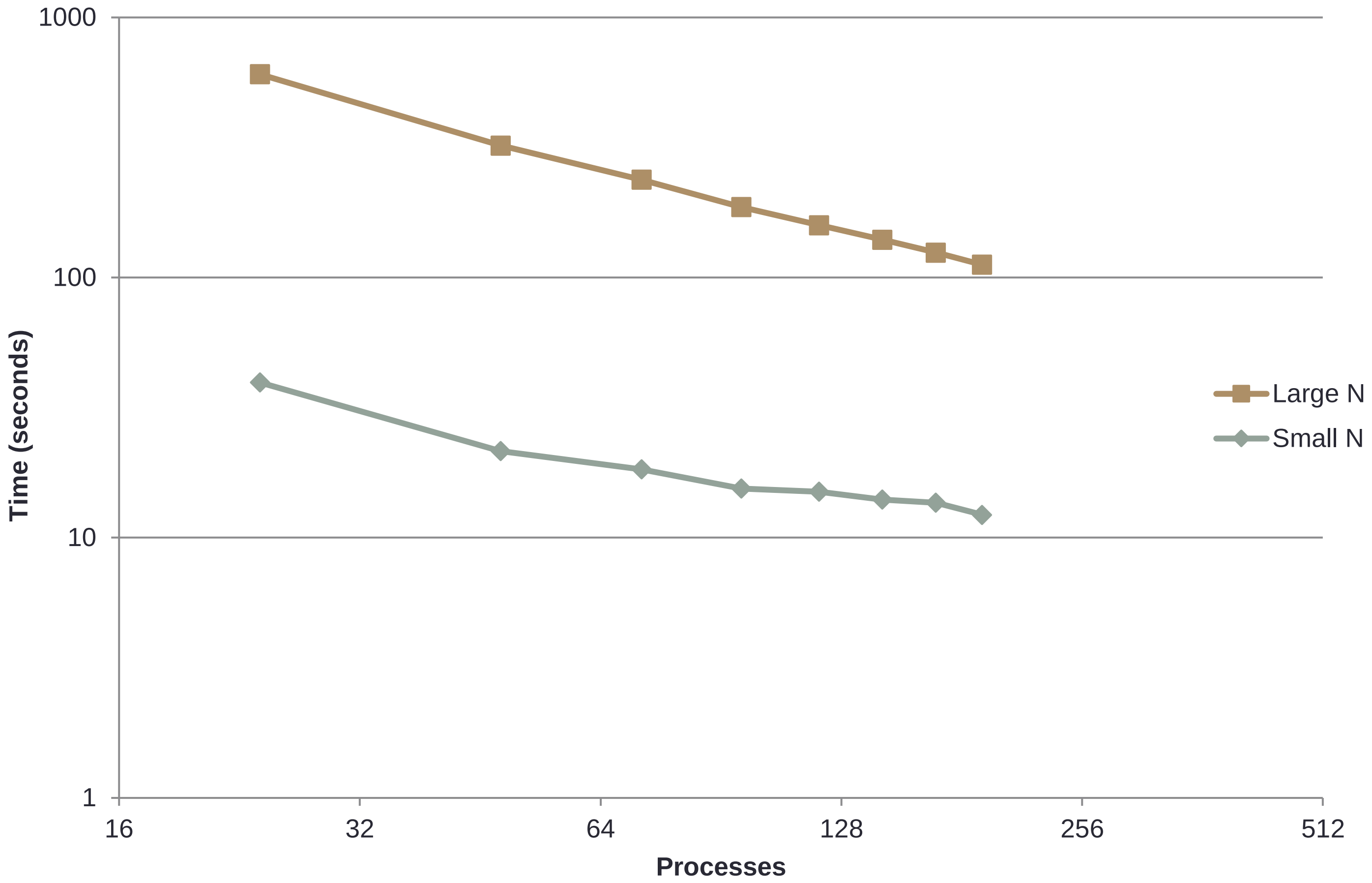
- Think carefully whenever you plot data
  - what am I trying to show with the graph?
  - is it easy to interpret?
  - can it be interpreted quantitatively?

- Default plotting options are rarely what you want
  - default colours can be hard to read (e.g. yellow on white)
  - default axis limits may not be sensible
  - ...
- Test data
  - MPI traffic model on multiple (24-core) nodes of ARCHER

Hard to interpret small N data here

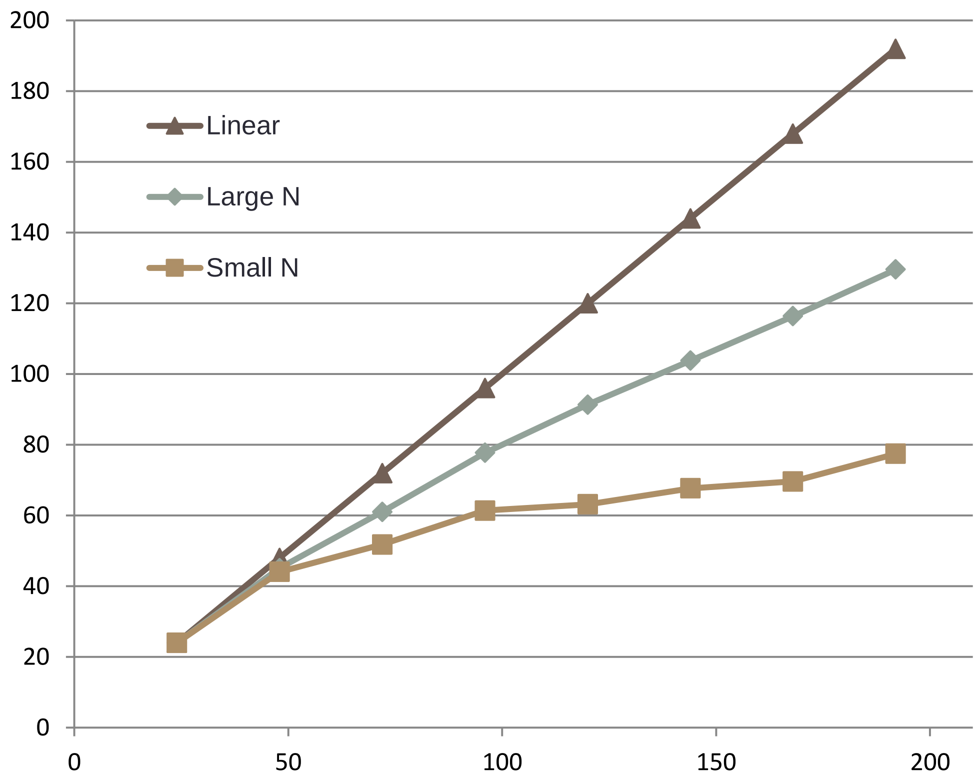


log/log can make trends in data too similar

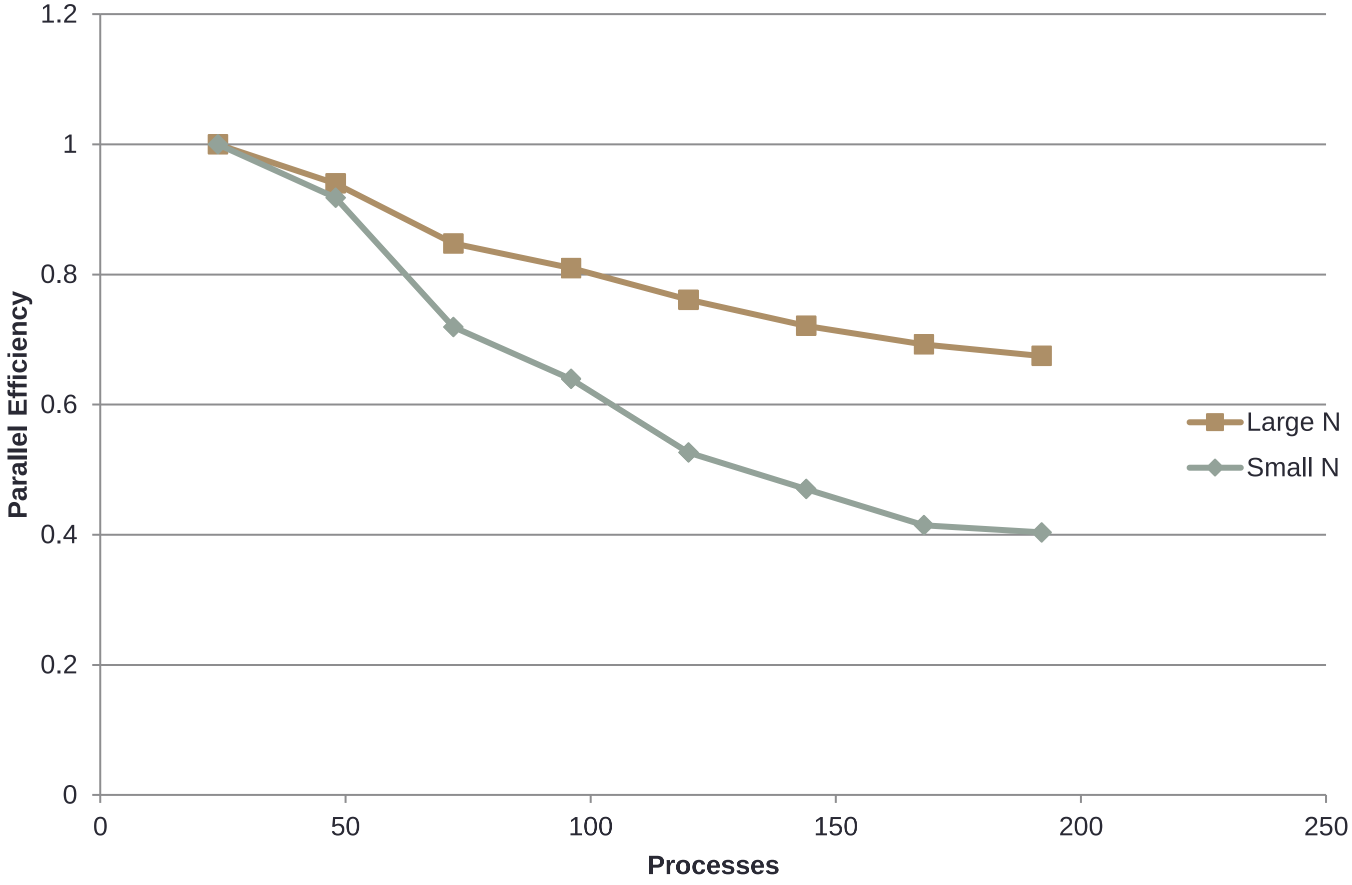


Normalised data easier to compare

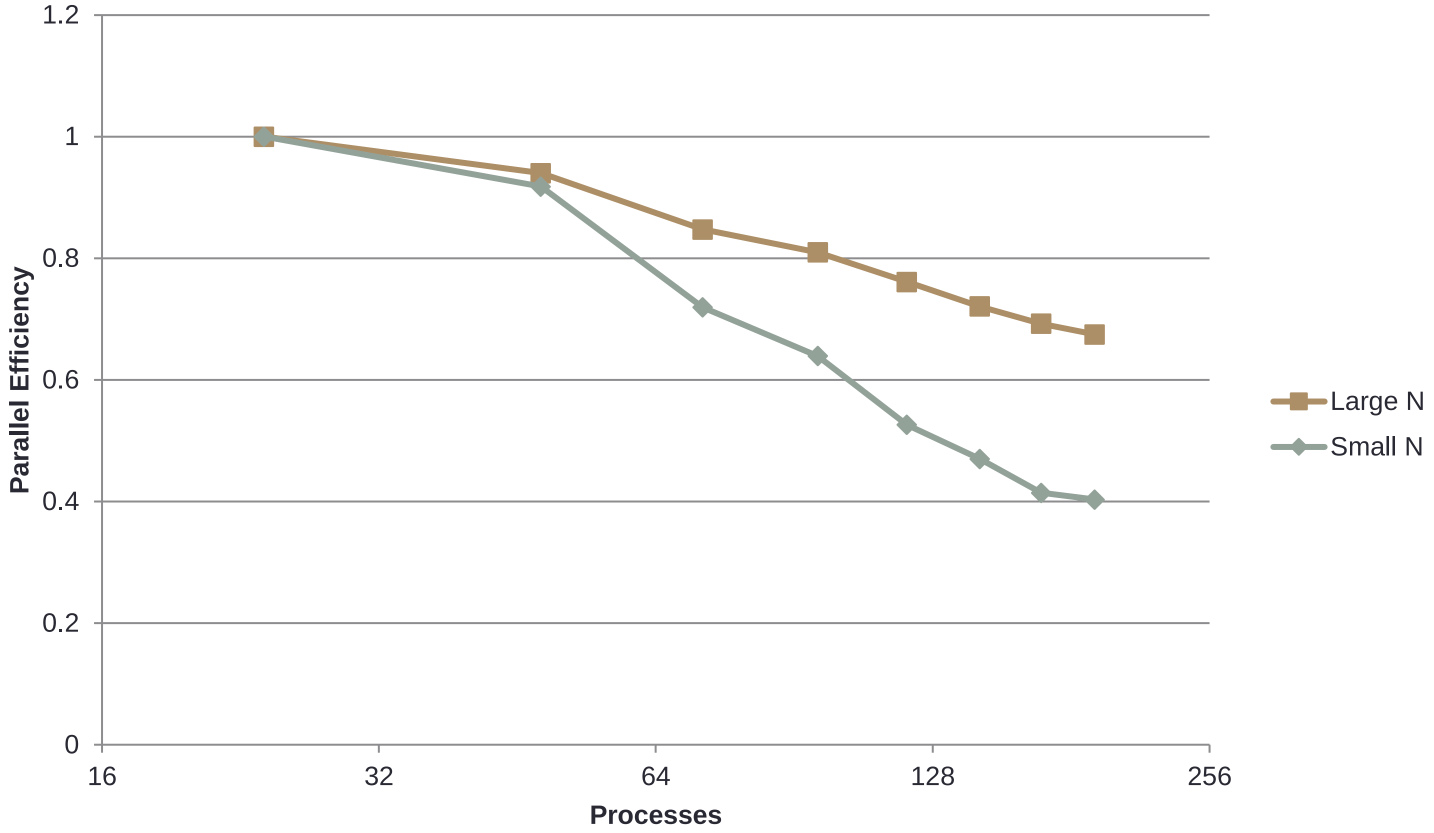
- use single-node (24-core) performance as baseline here



Efficiency plots can be useful too

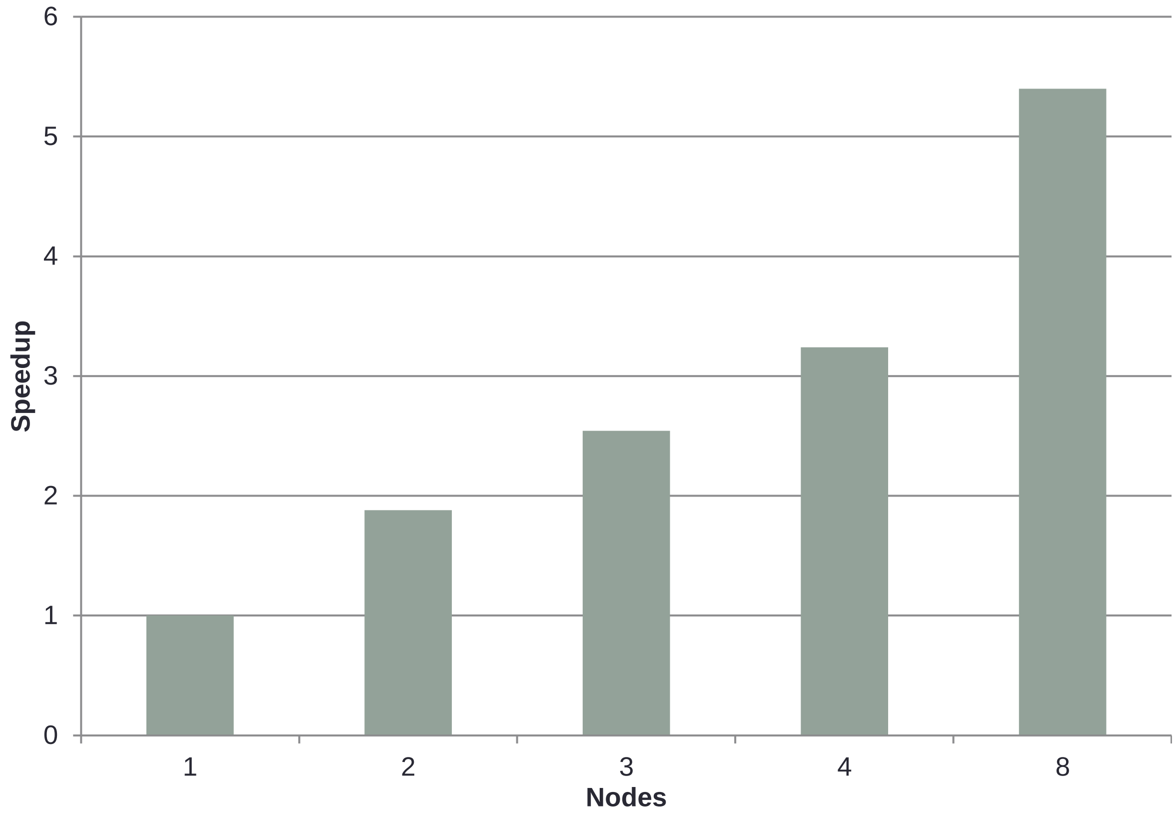


log/lin efficiency if many points at small *P*



Don’t just accept the default options

- In this chart the x-axis doesn’t have a meaningful scale



Summary

- A variety of considerations when parallelising code
  - serial sections
  - communications overheads
  - load balance
  - ...
- Scaling is important
  - the better a code scales the larger machine it can use efficiently
- Quantitative metrics exist to give you an indication of how well your code performs and scales
  - important to plot them appropriately