Performance metrics

How is my parallel code performing and scaling?

Reusing this material

Performance metrics

- A typical program has two categories of components
  - Inherently sequential sections: can't be run in parallel
  - Potentially parallel sections

- 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 $E(N) <= 1$

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

*where N is the size of the problem and P the number of processors*

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
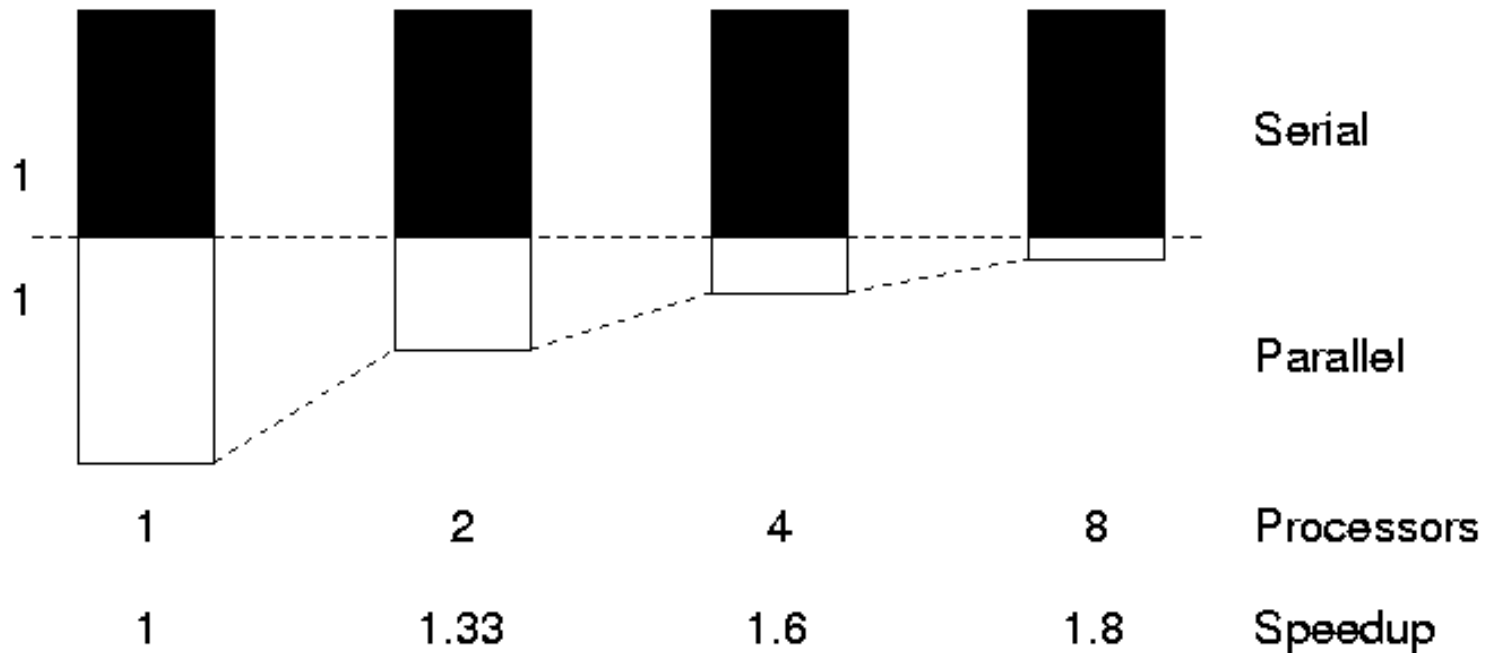

Speed-up vs No of processors
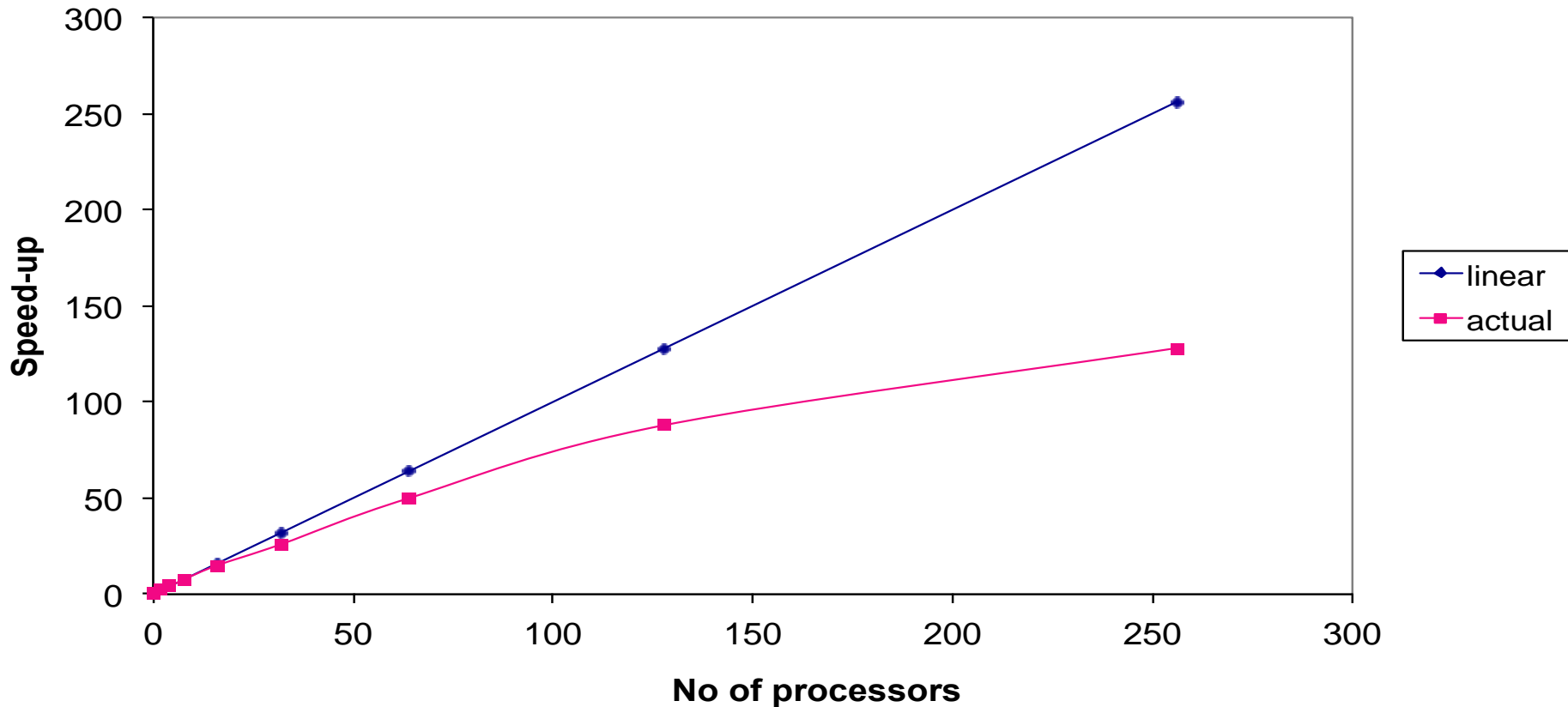
Weak scaling
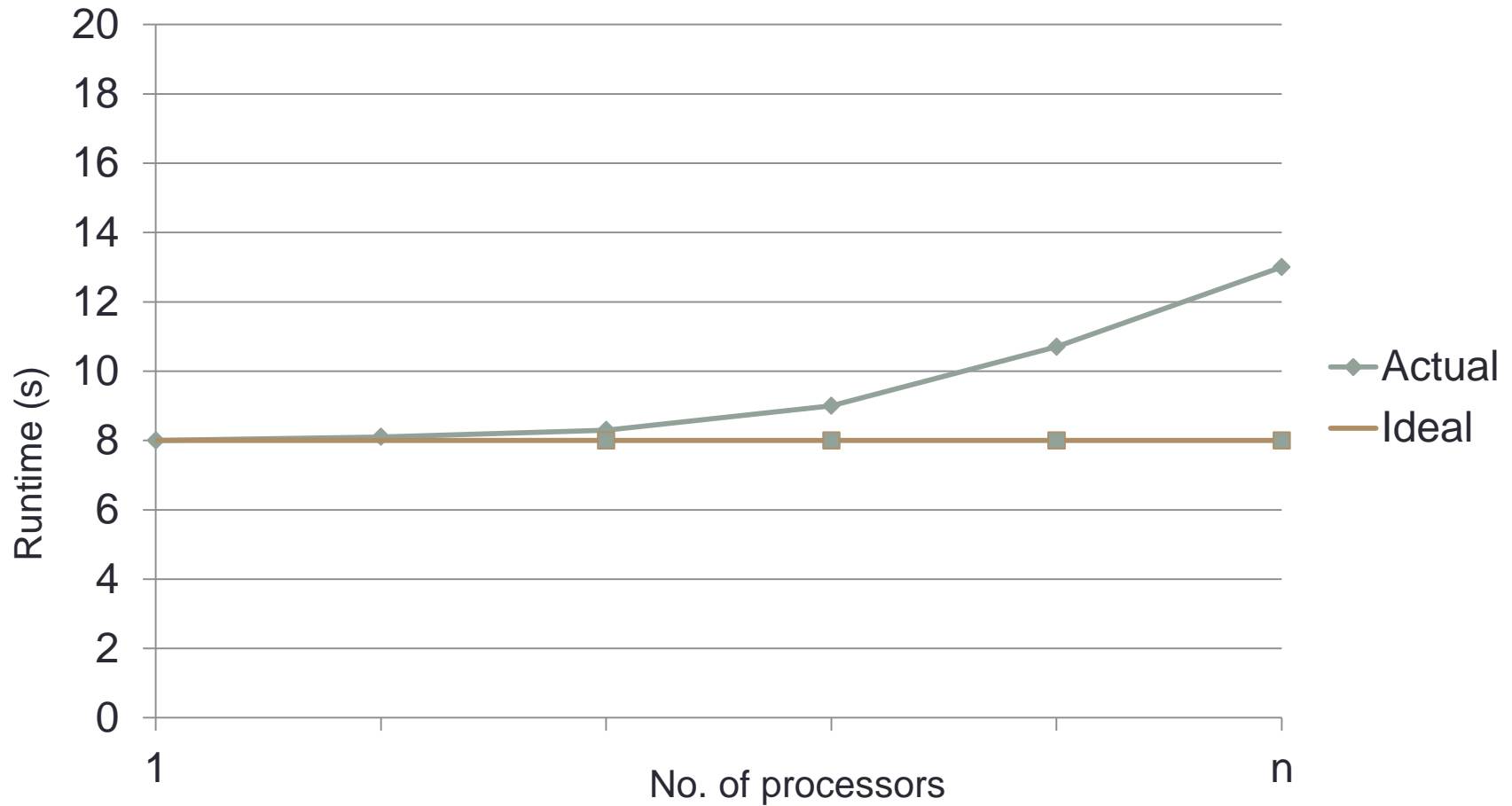
The serial section of code

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

Gene Amdahl, 1967

	1	2	4	8	Processors
	1	1.33	1.6	1.8	Speedup

# Amdahl's law

- A fraction, a, is completely serial

- Parallel runtime
$$T(N,P) = a\,T(N,1) + \frac{(1-a)\,T(N,1)}{P}$$
  - Assuming parallel part is 100% efficient

- Parallel speedup
$$S(N,P) = \frac{T(N,1)}{T(N,P)} = \frac{P}{aP + (1-a)}$$

- We are fundamentally limited by the serial fraction
  - For $a = 0$, $S = P$ as expected (i.e. *efficiency* = 100%)
  - Otherwise, speedup limited by $1/a$ for any P
    - *For* $a = 0.1$; 1/0.1 = 10 therefore 10 times maximum speed up
    - *For* $a = 0.1$; S(N, 16) = 6.4, S(N, 1024) = 9.9

Gustafson's Law

- We need larger problems for larger numbers of CPUs
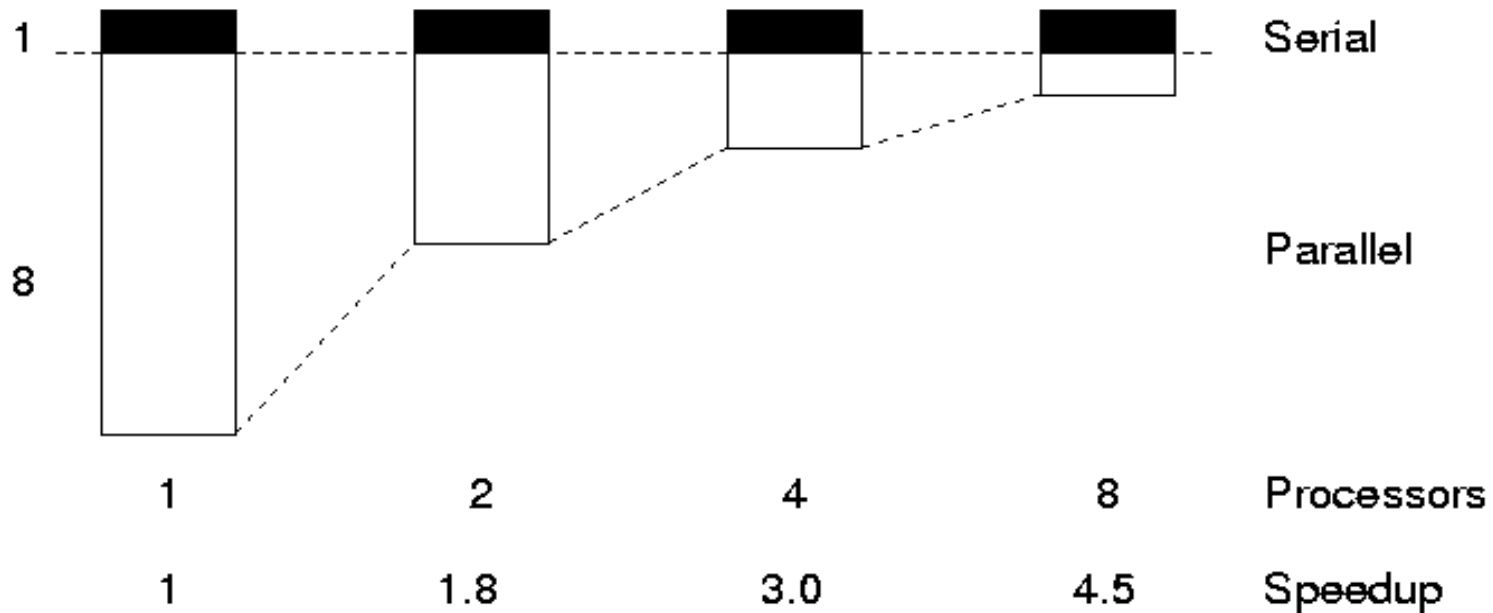
- Whilst we are still limited by the serial fraction, it becomes less important

Utilising Large Parallel Machines

- Assume parallel part is O(N), serial part is O(1)
  - time

$$T(N,P) = T_{serial}(N,P) + T_{parallel}(N,P)$$

$$= \alpha T(1,1) + \frac{(1-\alpha)NT(1,1)}{P}$$

  - speedup

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

- Scale problem size with CPUs, i.e. set $N = P$ (weak scaling)
  - speedup $\quad S(P,P) = a + (1-a)P$

  - efficiency $\quad E(P,P) = \frac{a}{P} + (1-a)$

Gustafson's Law

- If you can increase the amount of work done by each process/task then the serial component will not dominate
  - Increase the problem size to maintain scaling
  - This can be in terms of adding extra complexity or increasing the overall problem size.

$$S(N*P, P) = P - a(P-1)$$

  - Due to the scaling of N, effectively the serial fraction becomes $\dfrac{a}{P}$

- For instance, $a = 0.1$

$$S(16\,N, 16) = 14.5$$
$$S(1024\,N, 1024) = 921.7$$

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 version of traffic model on multiple nodes of ARCHER
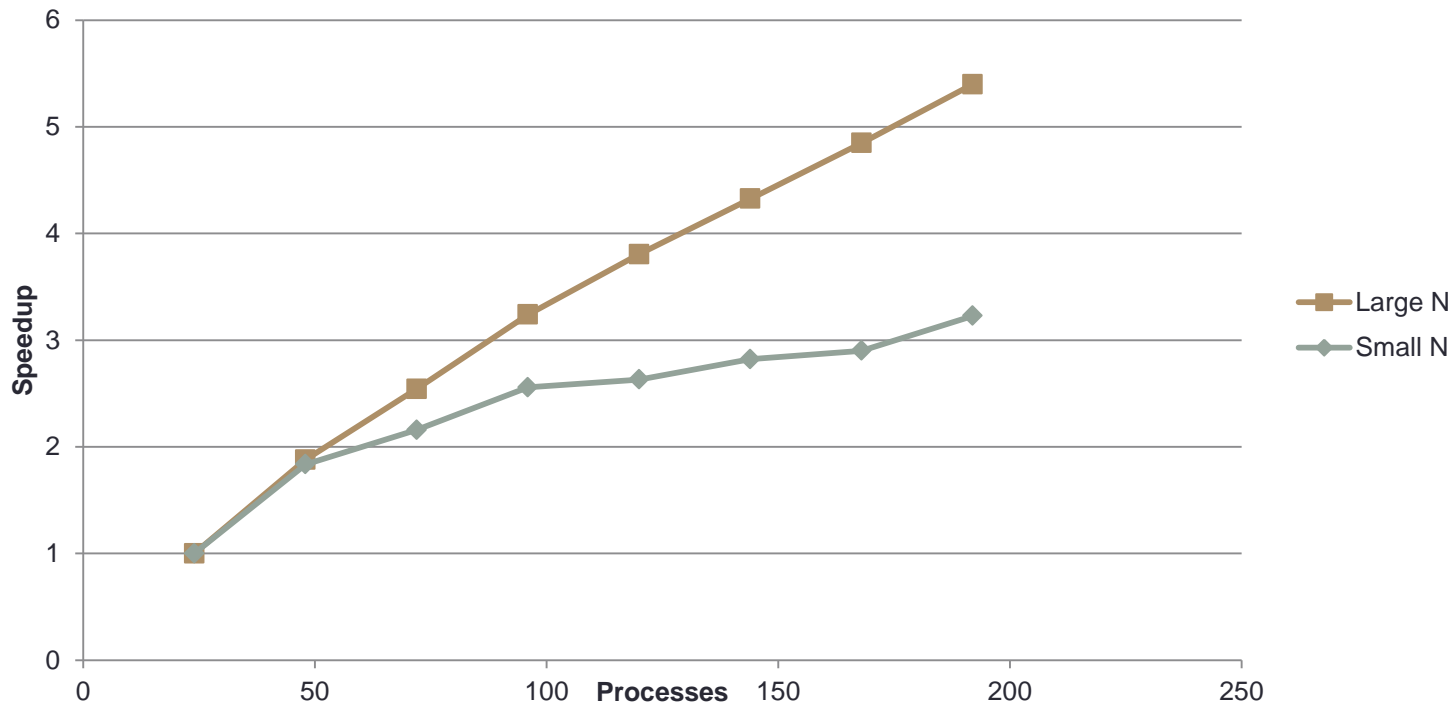
Hard to interpret small N data here
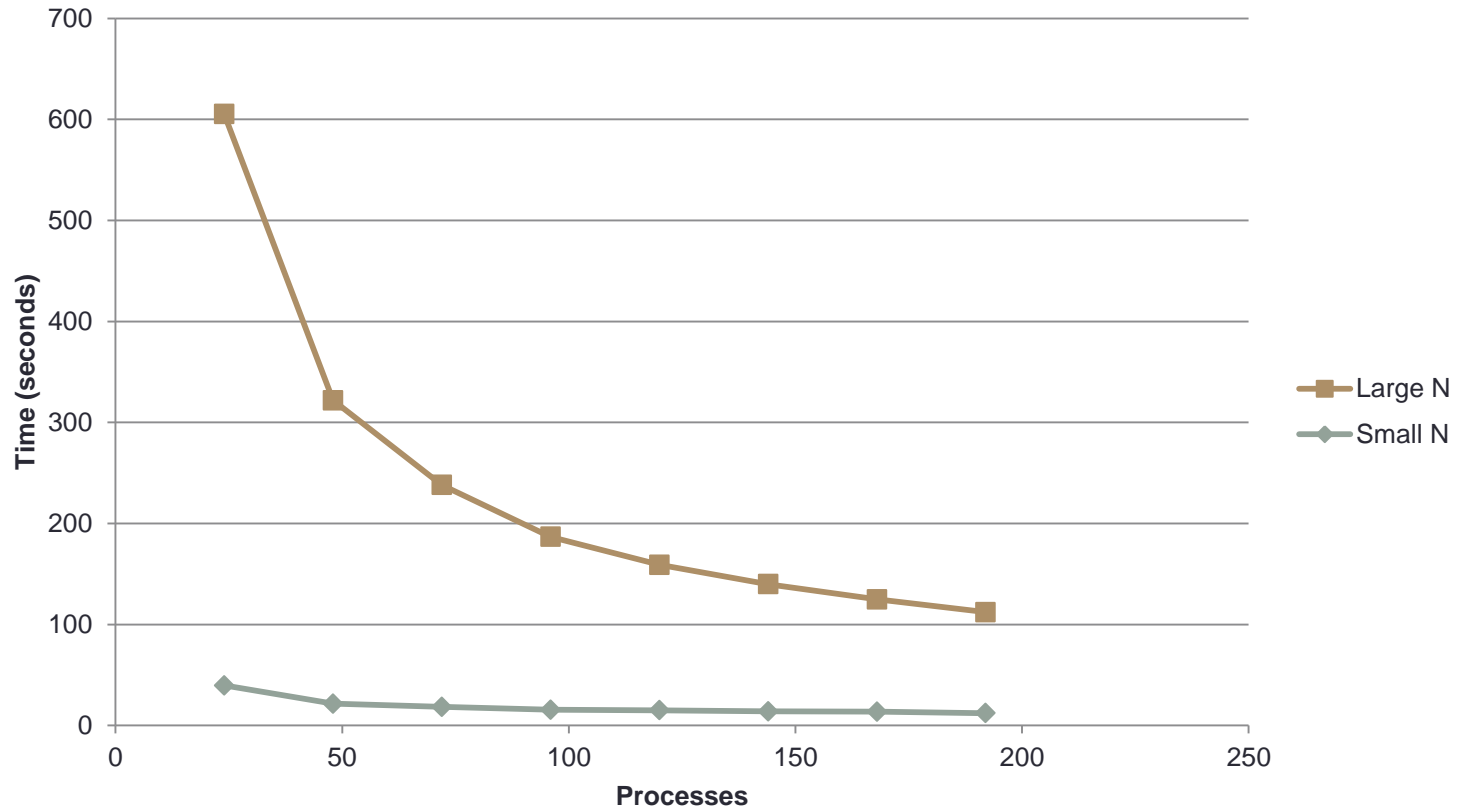
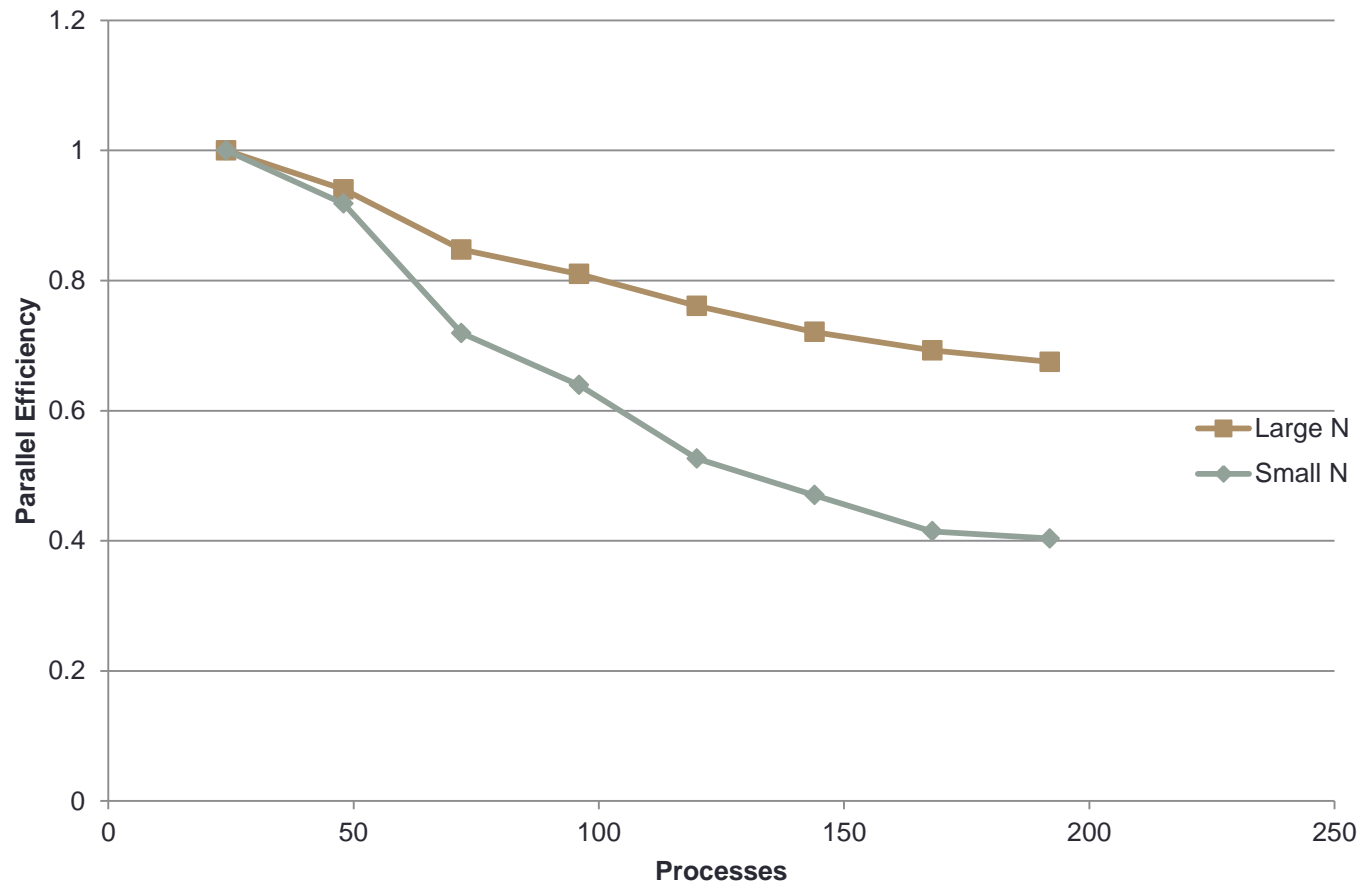# log/log can make trends in data too similar
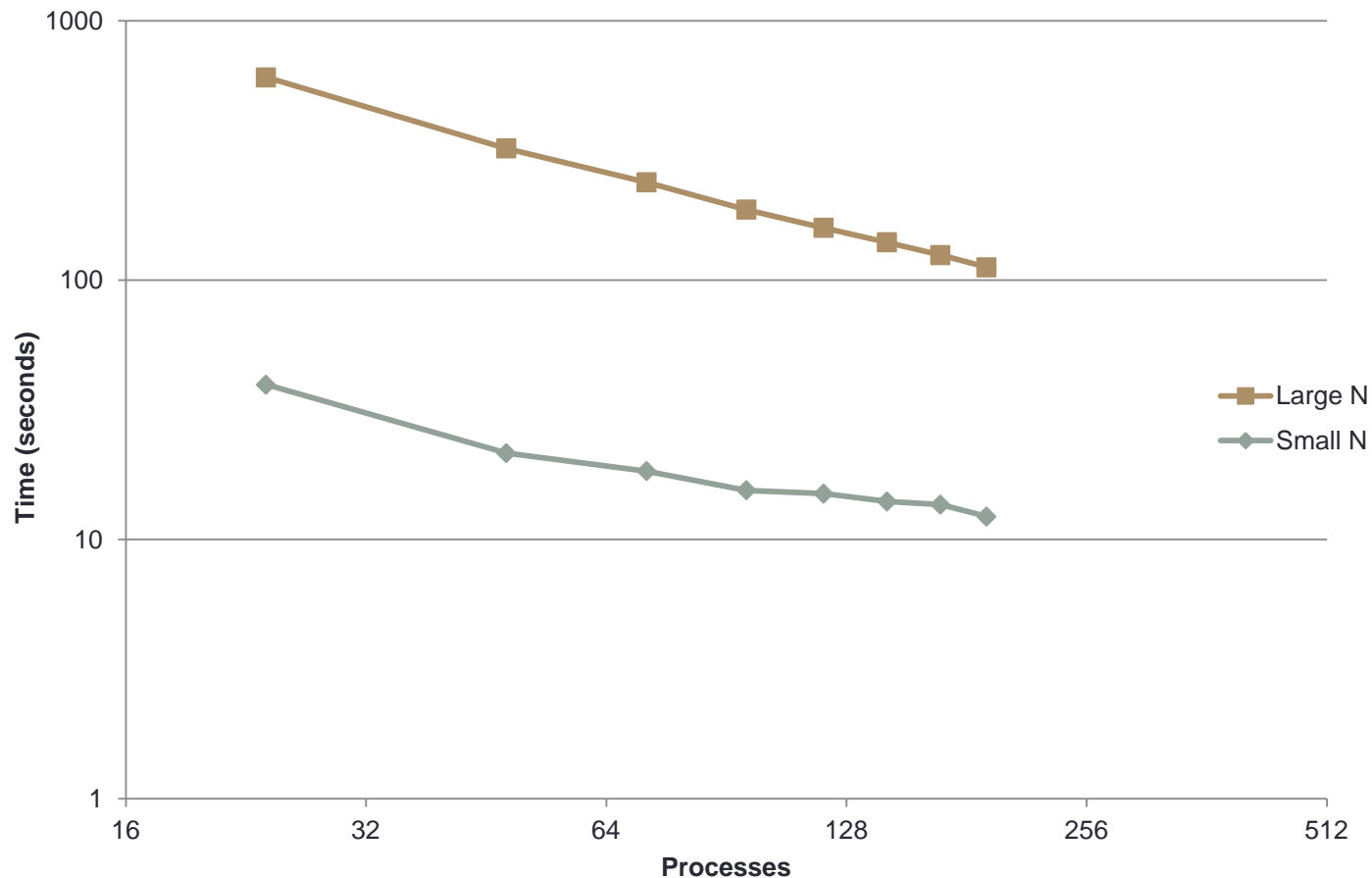
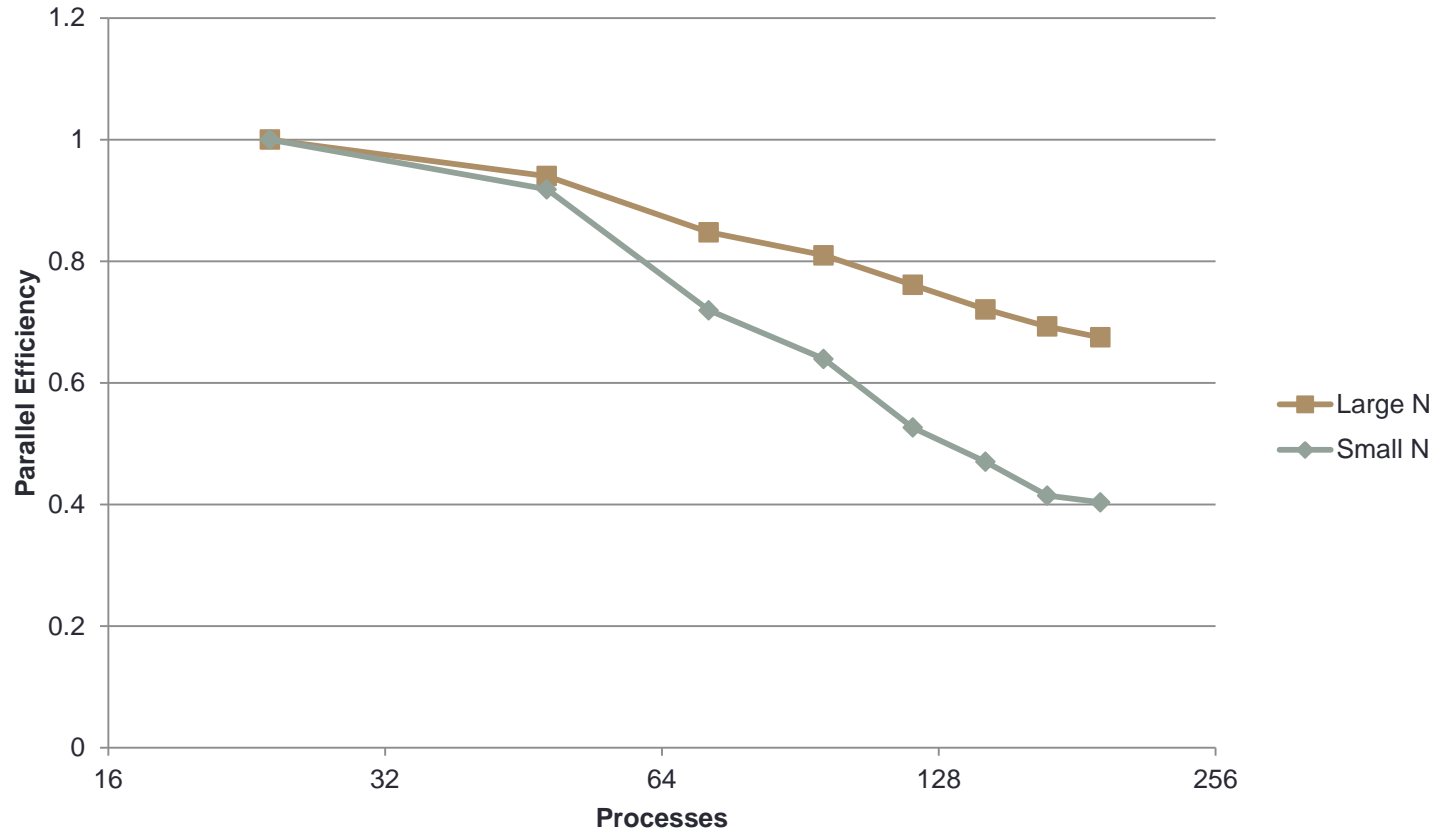# Normalised data easier to compare

- use single-node (24-core) performance as baseline here
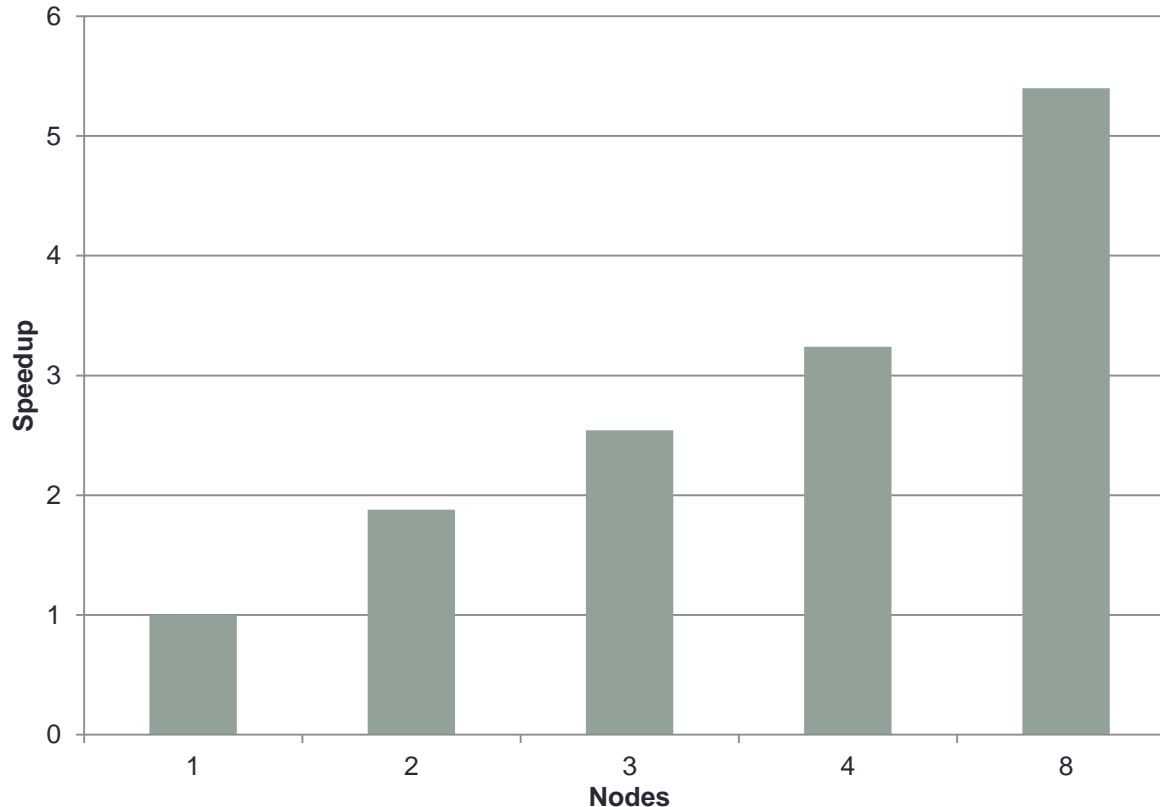
Efficiency plots can be useful too

log/linear useful if many points at small P

Don't just accept the default options

- In this bar 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 take advantage of

- Metrics exist to give you an indication of how well your code performs and scales
  - important to plot them appropriately