Measuring Parallel Performance

How well does my application scale?

Partners

Funding

Horizon 2020
European Union Funding
for Research & Innovation

Reusing this material

Outline

- Performance Metrics
- Scalability
- Amdahl’s law
- Gustafson’s law
- Load Imbalance

Why care about parallel performance?

- Why do we run applications in parallel?
  - so we can get solutions more quickly
  - so we can solve larger, more complex problems
- If we use 10x as many cores, ideally
  - we'll get our solution 10x faster
  - we can solve a problem that is 10x bigger or more complex
  - unfortunately this is not always the case…
- Measuring parallel performance can help us understand
  - whether an application is making efficient use of many cores
  - what factors affect this
  - how best to use the application and the available HPC resources

Performance Metrics

- How do we quantify performance when running in parallel?

- Consider execution time *T(N,P)* measured whilst running on P “processors” (cores) with problem size/complexity N

- Speedup*:*
  - typically $S(N,P) < P$

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

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

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

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

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

Parallel Scaling

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

- Can investigate two types of scaling:
  - **Strong Scaling** (increasing *P*, **constant** *N*):
    problem size/complexity stays the same as the number of processors increases, decreasing the work per processor

  - **Weak Scaling** (increasing *P* , **increasing** *N*):
    problem size/complexity increases at the same rate as the number of processors, keeping the work per processor the same

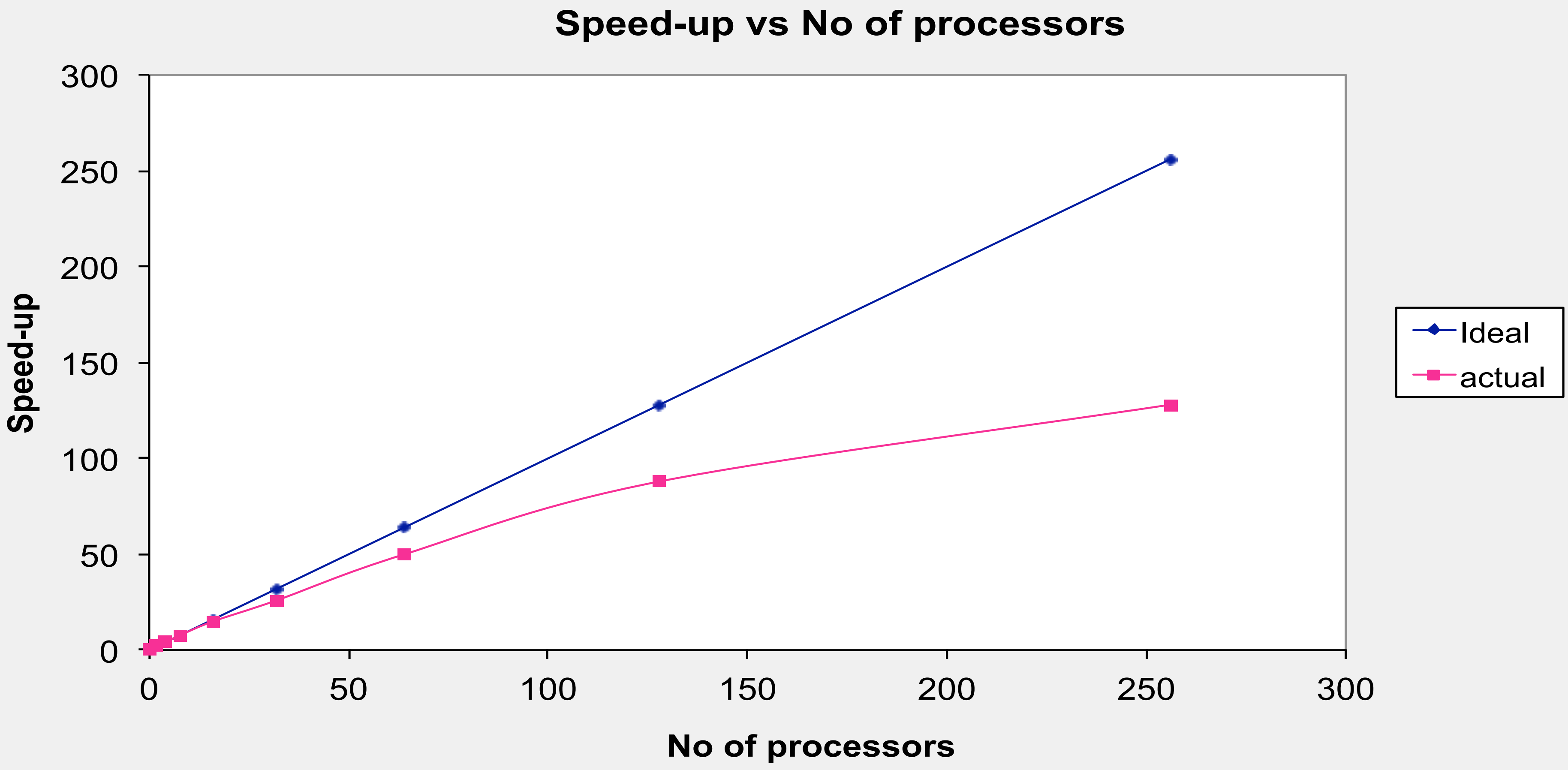
Parallel Scaling

- Ideal strong scaling: runtime keeps decreasing in direct proportion to the growing number of processor used

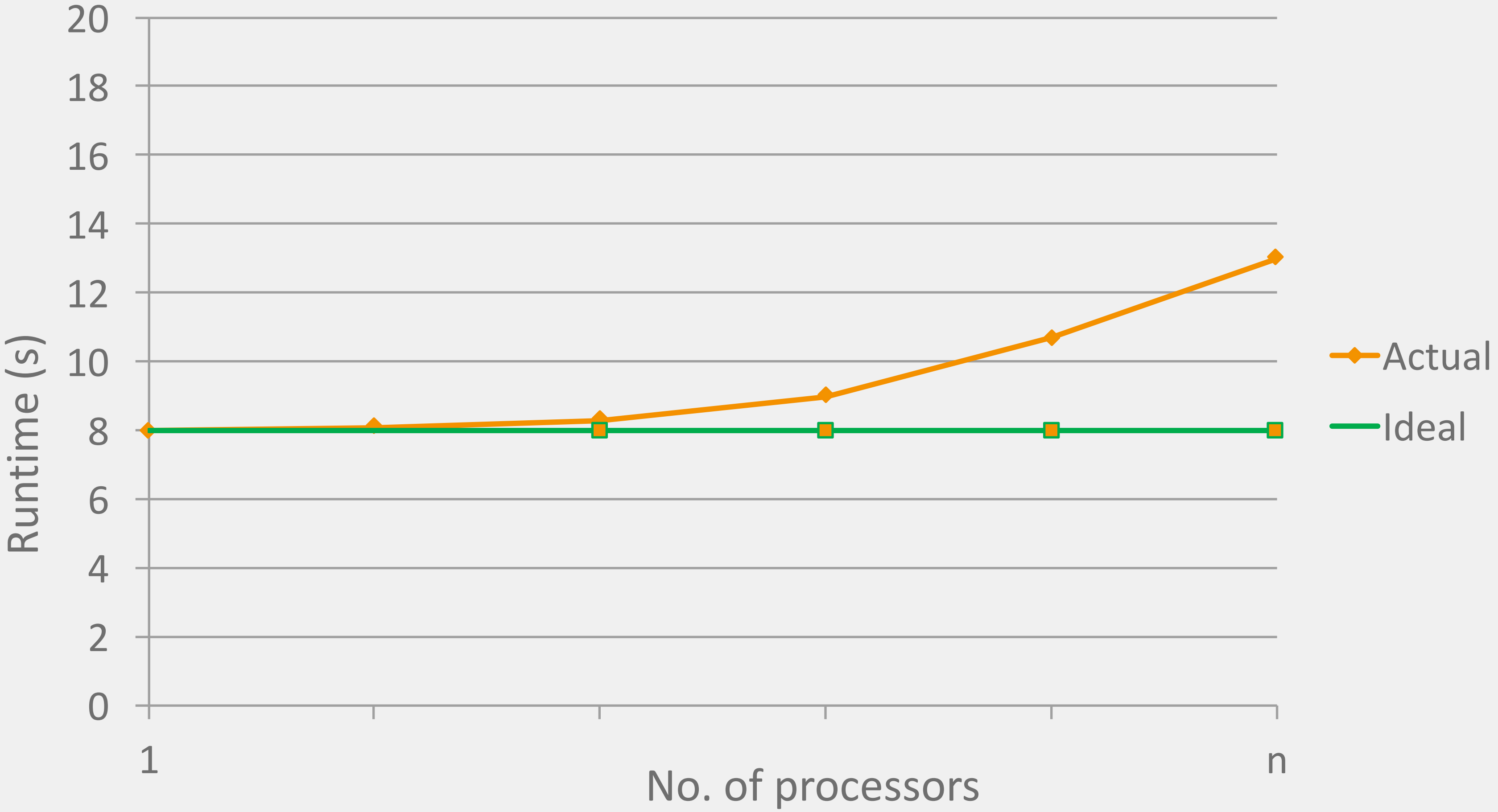
- Ideal weak scaling: runtime stays constant as the problem size gets bigger and bigger

- Good strong scaling is generally more relevant for most scientific problems, but more difficult to achieve than good weak scaling

Typical strong scaling behaviour



Typical weak scaling behaviour



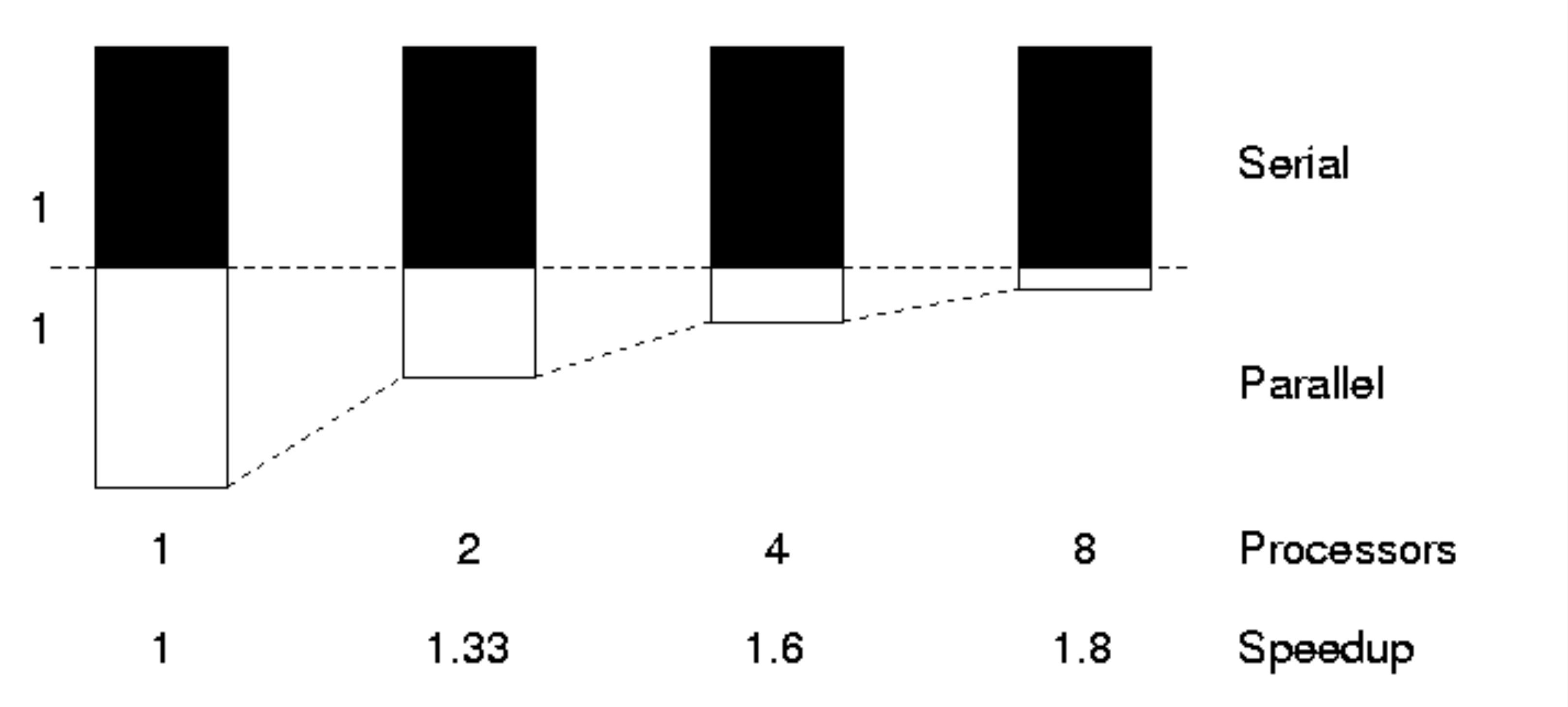
Limits to scaling – the serial fraction

Amdahl’s Law

Amdahl's Law - illustrated

"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 - proof

- Consider a typical program, which has:
  - Sections of code that are inherently serial so can't be run in parallel
  - Sections of code that could potentially run in parallel
- Suppose serial code accounts for a fraction α of the program's runtime
- Assume the potentially parallel part could be made to run with 100% parallel efficiency, then:
- Hypothetical runtime in parallel = $T(N,P) = \alpha T(N,1) + \frac{(1-\alpha)T(N,1)}{P}$
- Hypothetical speedup = $S(N,P) = \frac{T(N,1)}{T(N,P)} = \frac{P}{\alpha P + (1-\alpha)}$

Amdahl's Law - proof

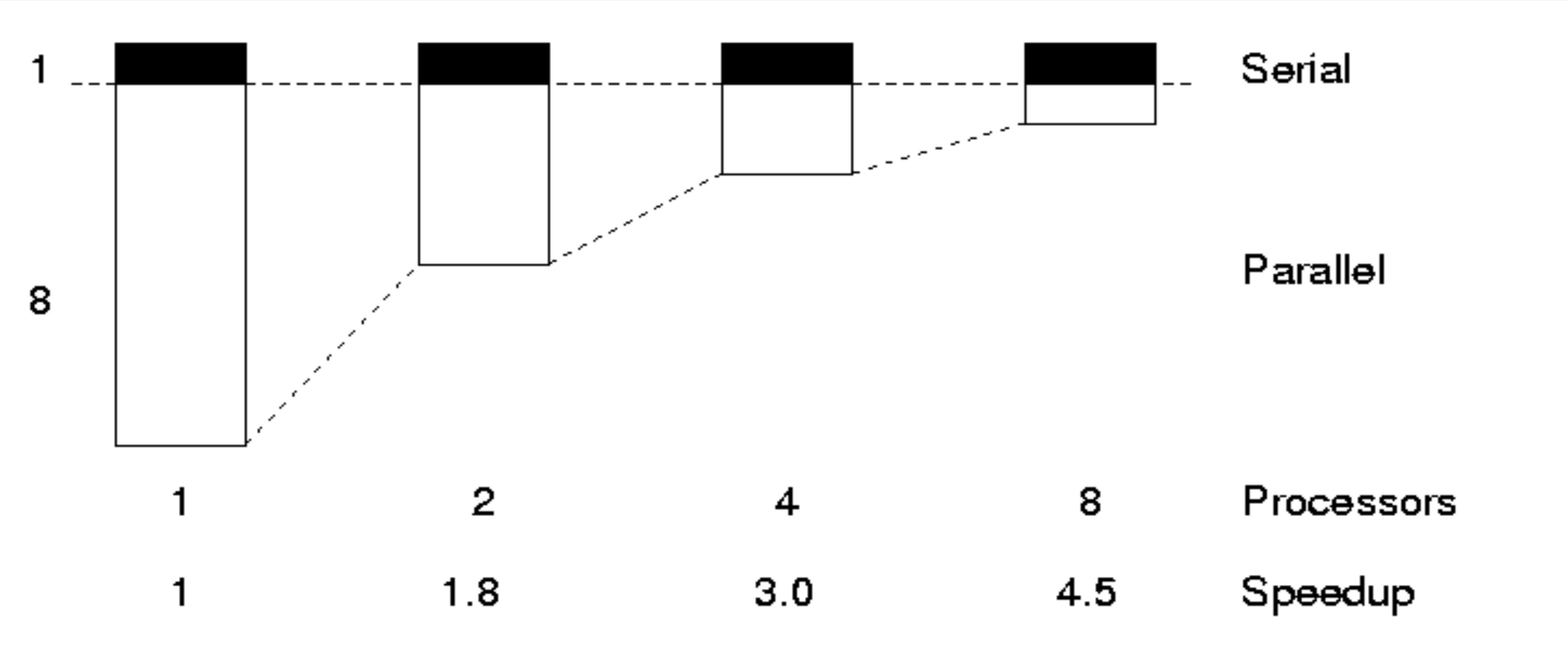
- Hypothetical speedup = $S(N,P) = \dfrac{P}{\alpha P + (1-\alpha)}$
- What does this mean?
- Speedup fundamentally limited by the serial fraction
  - Speedup will always be less than $1/\alpha$ no matter how large P
- E.g. for α = 0.1:
  - hypothetical speedup on 16 processors = *S(N,16)* = 6.4
  - hypothetical speedup on 1024 processors = *S(N,1024)* = 9.9
  - ...
  - maximum theoretical speed up is 10.0

Limits to scaling – problem size

Gustafson’s Law

Gustafson’s Law - illustrated

We need larger problems for larger numbers of processors



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

Gustafson's Law - proof

- Assume parallel contribution to runtime is proportional to *N*, and serial contribution independent of *N*

- Then total runtime on *P* processors =

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

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

- And total runtime on 1 processor =

$$T(N,1) \ = \alpha T(1,1) + (1-\alpha)\ N\ \mathrm{T}(1,1)$$

Gustafson's Law - proof

- Hence speedup = $S(N,P) = \dfrac{T(N,1)}{T(N,P)} = \dfrac{\alpha + (1-\alpha)N}{\alpha + (1-\alpha)\frac{N}{P}}$

- If we scale problem size with number of processors, i.e. set $N = P$ (weak scaling), then:
  - speedup $S(P,P) = \alpha + (1- \alpha) P$
  - efficiency $E(P,P) = \alpha /P + (1- \alpha)$

- What does this mean?

Gustafson's Law – consequence

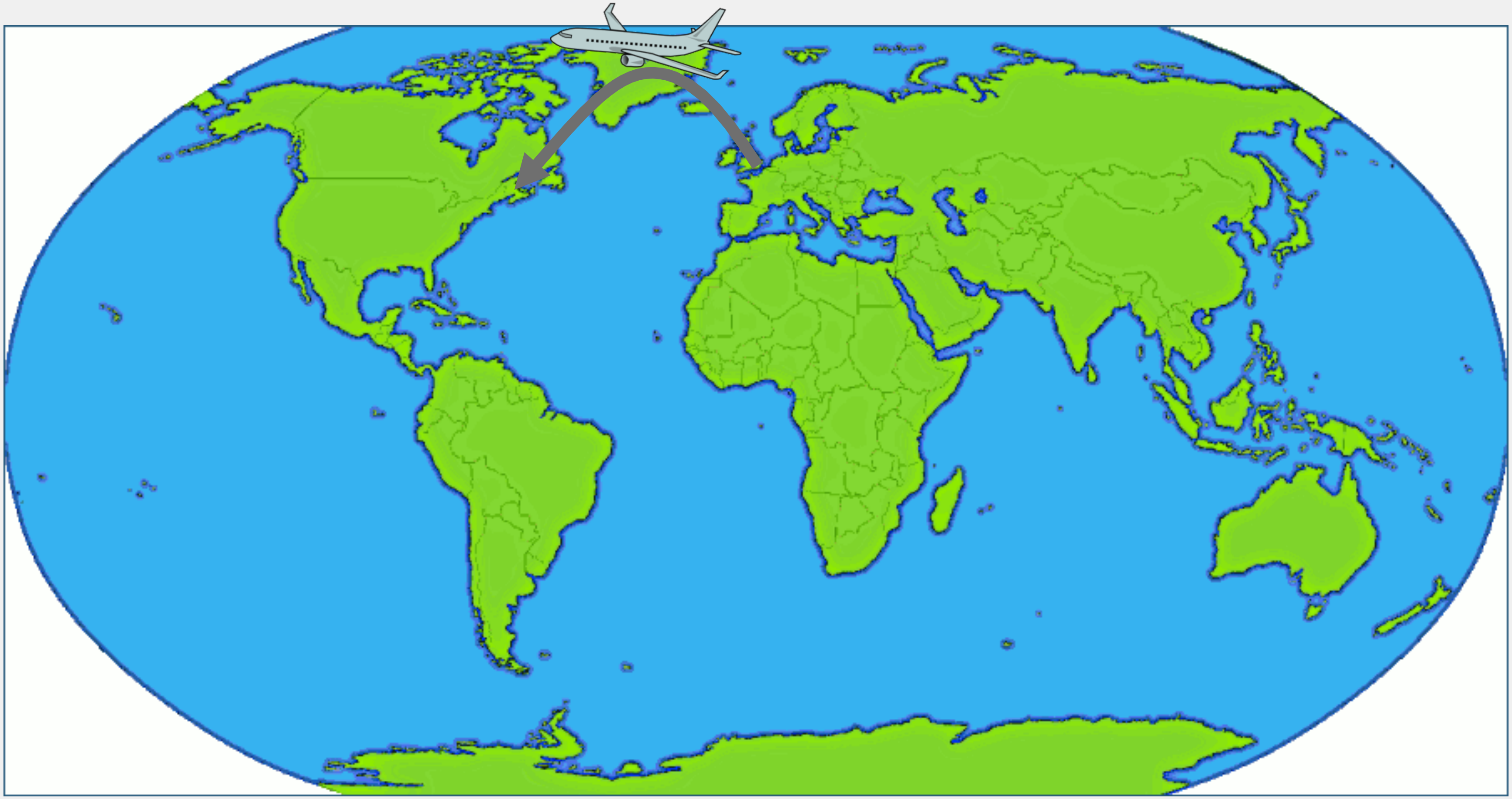
Efficient Use of Large Parallel Machines

- If you increase the amount of work done by each parallel task then the serial component will not dominate
  - Increase the problem size to maintain scaling
  - Can do this by adding extra complexity or increasing the overall problem size

Number of processors	Strong scaling (Amdahl's law)	Weak scaling (Gustafson's law)
16	6.4	14.5
1024	9.9	921.7

Due to the scaling of N, the serial fraction effectively becomes α/P

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! $\alpha = 4/12 = 0.33$; 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"

Load Imbalance

- These laws all assumed all processors are equally busy
  - what happens if some run out of work?
- Specific case
  - four people pack boxes with cans of soup: 1 minute per box

Person	Anna	Paul	David	Helen	Total
# boxes	6	1	3	2	12

  - takes 6 minutes as everyone is waiting for Anna to finish!
  - if we gave everyone same number of boxes, would take 3 minutes

- Scalability isn’t everything
  - make the best use of the processors at hand before increasing the number of processors

Quantifying Load Imbalance

- Define Load Imbalance Factor

  $$LIF = maximum\ load\ /\ average\ load$$

  - for perfectly balanced problems $LIF = 1.0$, as expected
  - in general, $LIF > 1.0$
  - LIF tells you how much faster your calculation could be with balanced load

- Box packing
  - $LIF = 6/3 = 2$
  - initial time = 6 minutes
  - best time = 6/2 = 3 minutes

Summary

- Key performance metric is execution time
- Good scaling is important, as the better a code scales the larger a machine it can make efficient use of and the faster you'll solve your problem
  - can consider weak and strong scaling
  - in practice, overheads limit the scalability of real parallel programs
  - Amdahl's law models these in terms of serial and parallel fractions
  - larger problems generally scale better: Gustafson's law
- Load balance is also a crucial factor
- Metrics exist to give you an indication of how well your code performs and scales