Measuring Parallel Performance

How well does my application scale?

Partners

MANCHESTER 1824 The University of Manchester
Ian Harrow Consulting
archer

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 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*):

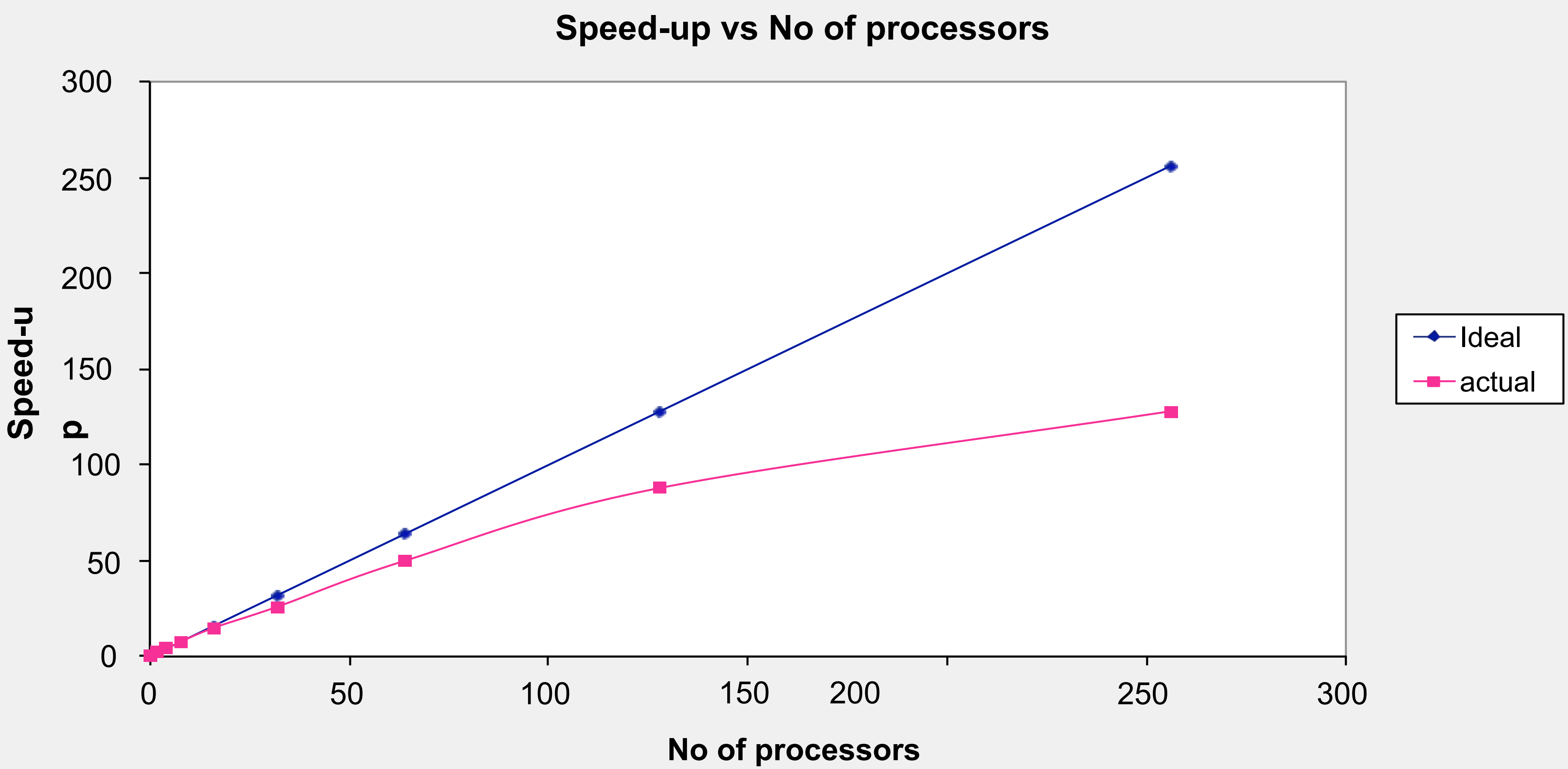
  - **Weak Scaling** (increasing *P* , **increasing** *N*):

Strong Scaling

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

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

Typical strong scaling behaviour



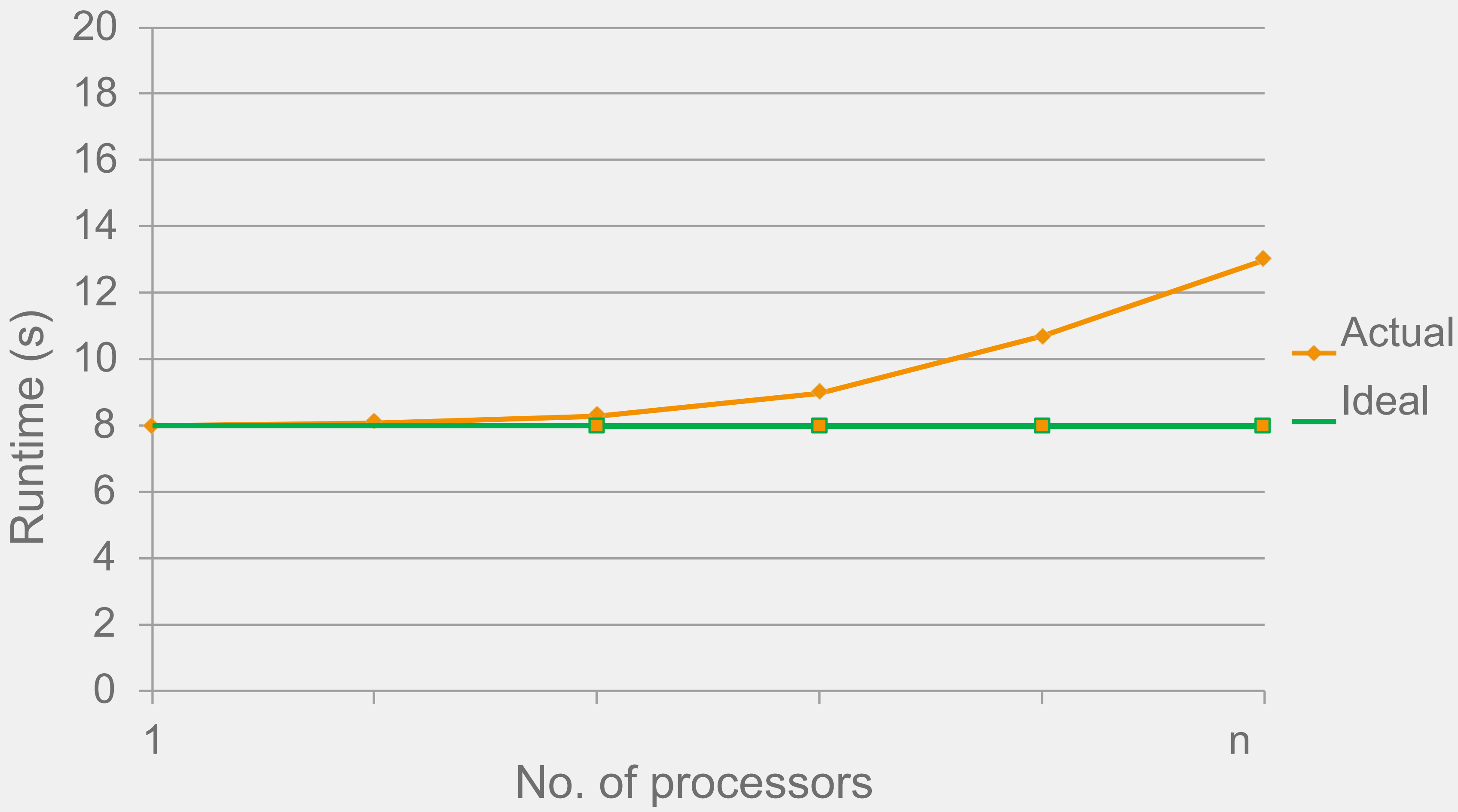
Weak Scaling

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

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

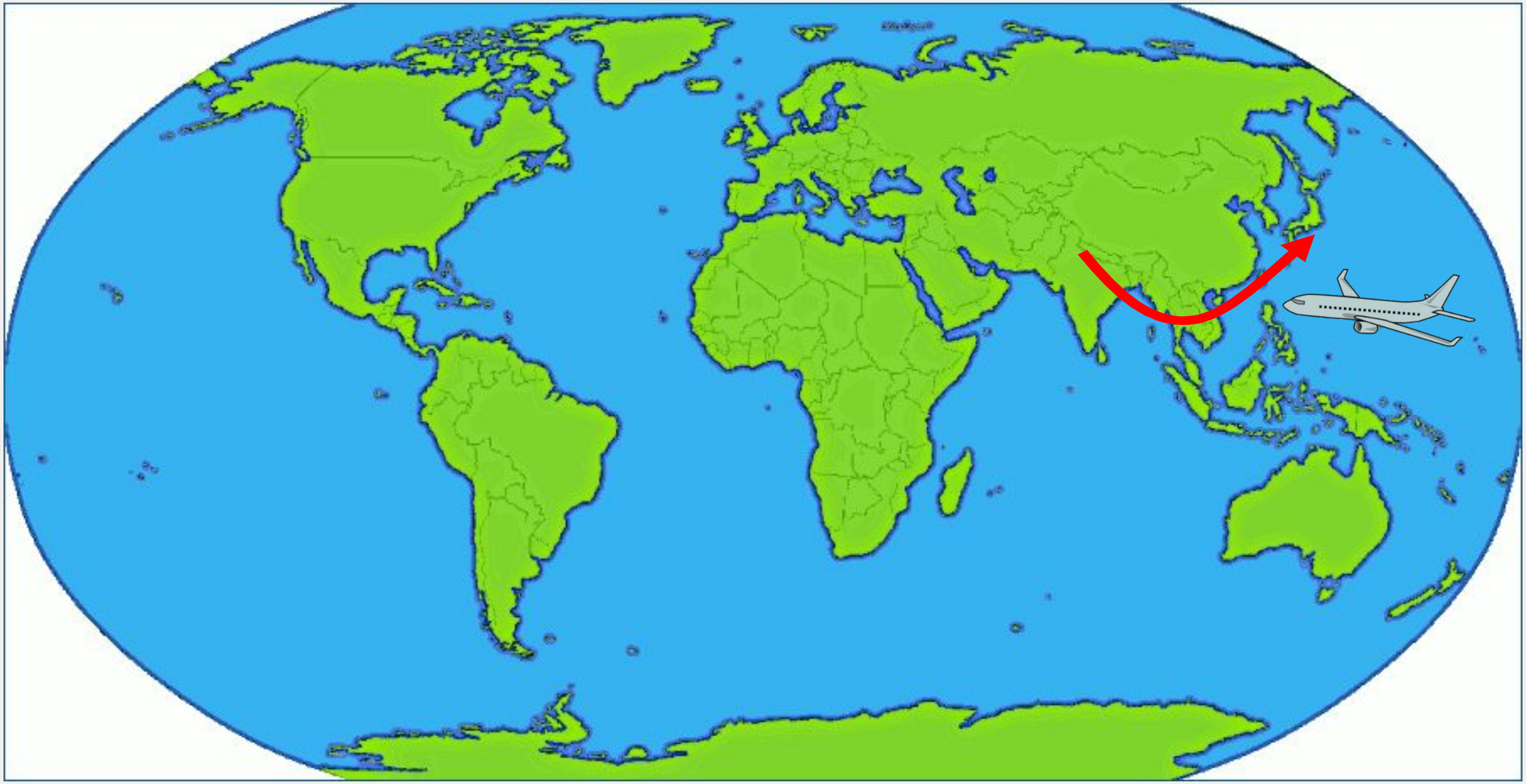
Typical weak scaling behaviour



Limits to scaling – the serial fraction

Amdahl’s Law

Analogy: Flying Delhi to Japan



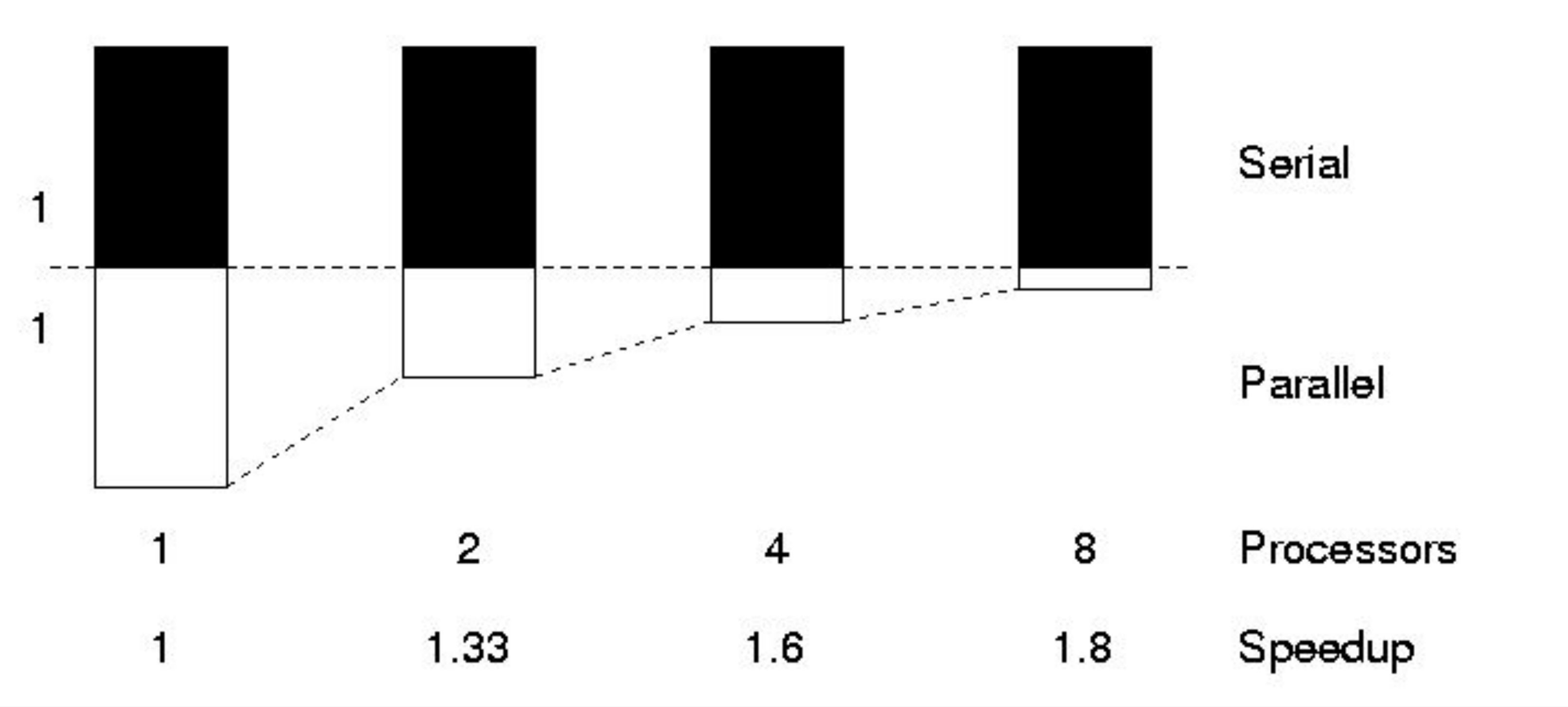
Delhi (Home) to Japan (Hotel)

- By Jumbo Jet
  - distance: 5600 km; speed: 700 kph
  - time: 8 hours ?
- No!
  - 1 hour by Cab to Delhi Airport + 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
- Max speedup = 3 (i.e. 4 hours)

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

- 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

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

- 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

- What does this mean?

- Speedup fundamentally limited by the serial fraction

Delhi (Home) to Japan (Hotel)

- By Jumbo Jet
  - distance: 5600 km; speed: 700 kph
  - time: 8 hours ?
- No!
  - 1 hour by Cab to Delhi Airport + 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

What is the maximum speed up?

What is alpha?

Exercise-1

Consider the following snippet of the program.

```
for ( i =0; i<n ; i++)
A[i]=B[i]+C[i]

Parallel  for ( i =0; i<n ; i++)
A[i]=B[i]+C[i]

for ( i =0; i<n ; i++)
A[i]=B[i]+C[i]
```

Assume each of the 3 for loops takes same time on the sequential execution.

If the above snippet is executed on a parallel machine that has P processors, then what is the maximum speed up we can achieve?

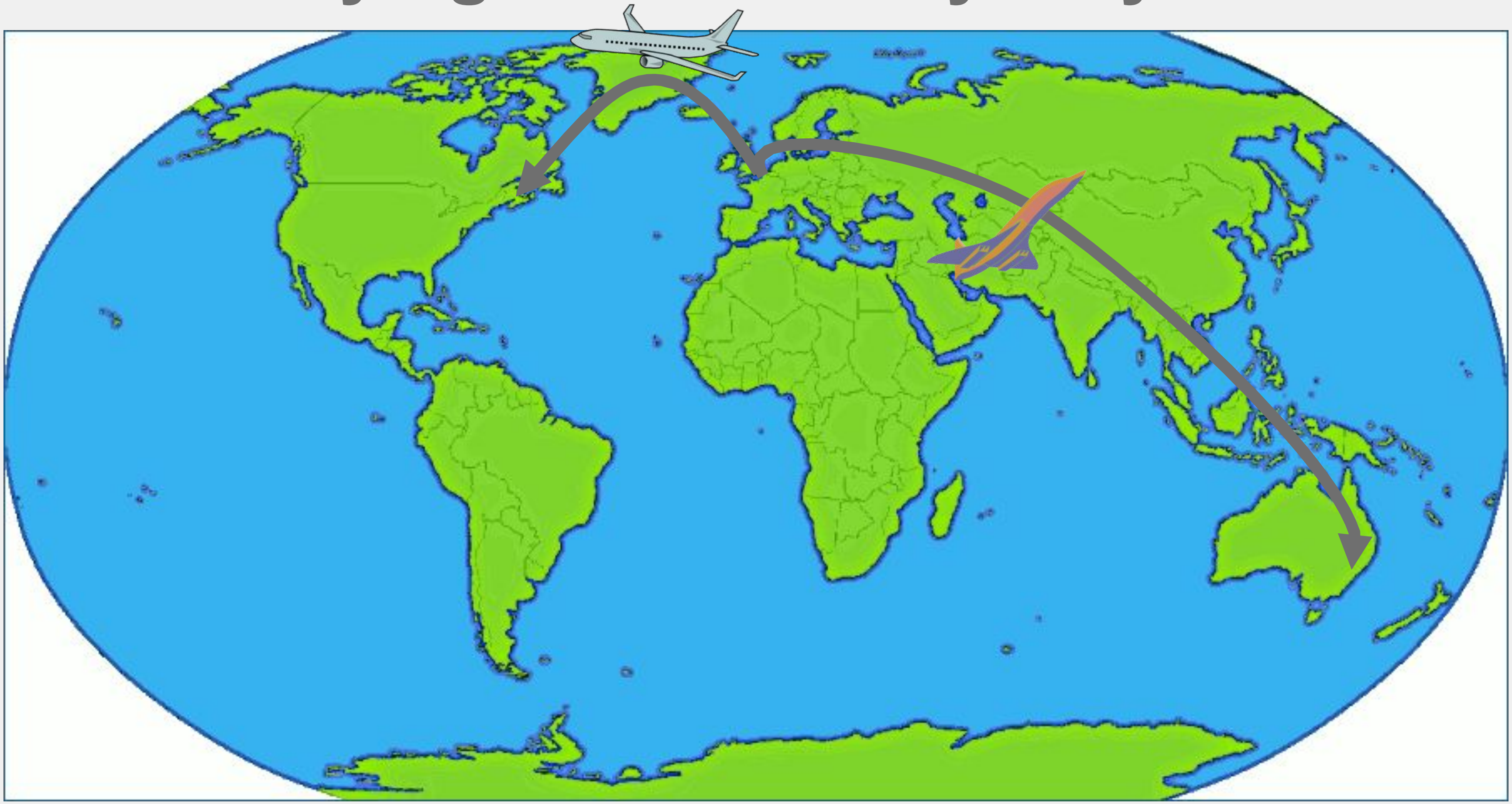
Exercise-2

- Memory operations currently take 30% of execution time.
- A new widget called a “cache” speeds up 80% of memory operations by a factor of 4
- A second new widget called a “L2 cache” speeds up 1/2 the remaining 20% by a factor or 2.
- What is the total speed up?

Limits to scaling – problem size

Gustafson's Law

Flying London to Sydney

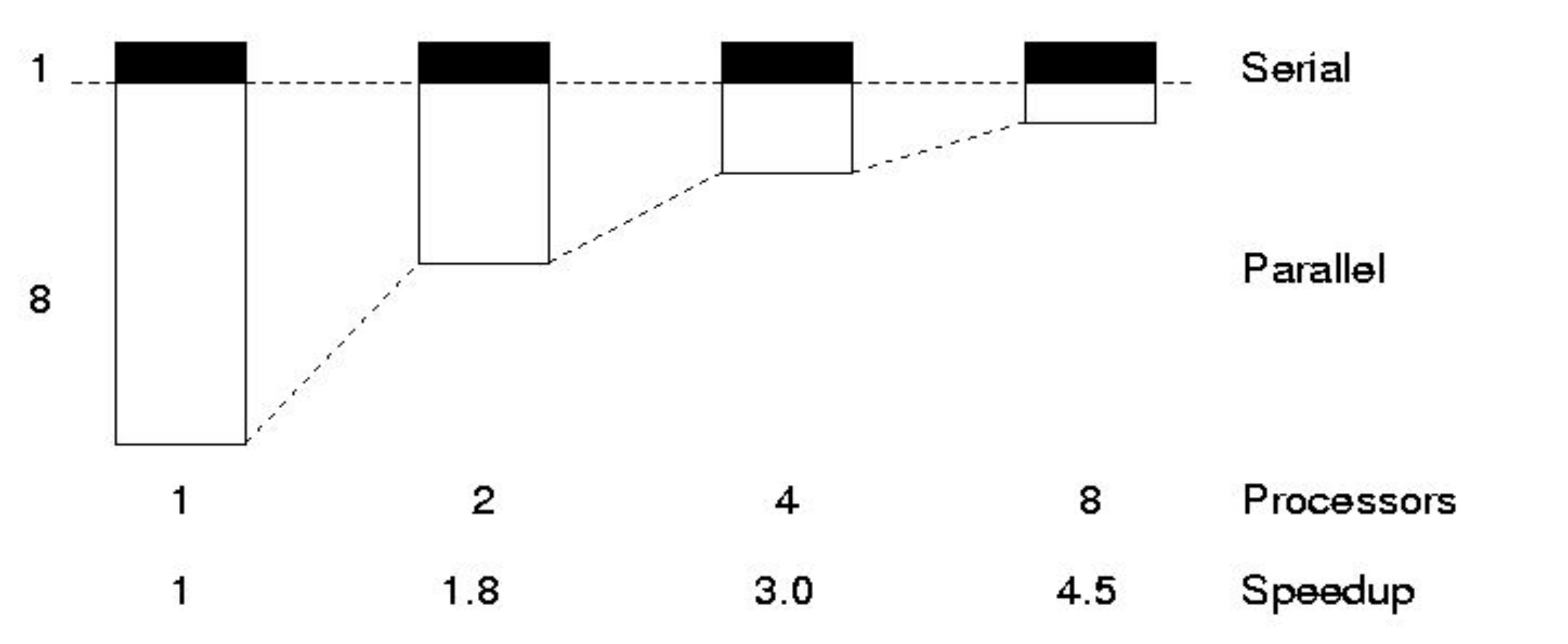


Buckingham Palace to Sydney Opera

- 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”

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

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

Gustafson's Law

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

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

α	Number of processors	Strong scaling (Amdahl's law)	Weak scaling (Gustafson's law)
0.1	16	6.4	14.5
0.1	1024	9.9	921.7

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

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