Performance Modelling

David Henty
EPCC
The University of Edinburgh

Overview

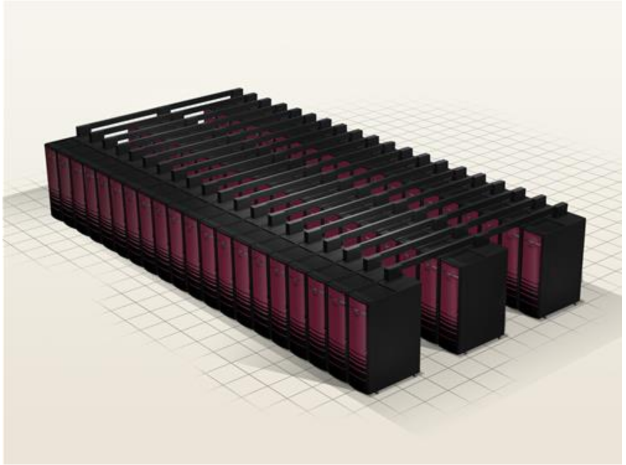
- Why bother?
- What is a performance model?
- Case study: MPI coursework example
- Estimating parameters
- Practical issues

Reference

- Talk to be read in conjunction with
  - “The Case of the Missing Supercomputer Performance: Achieving Optimal Performance on the 8,192 Processors of ASCI Q”
  - Fabrizio Petrini, Darren J. Kerbyson, Scott Pakin
  - “[W]hen you have eliminated the impossible, whatever remains, however improbable, must be the truth.”
  - Sherlock Holmes, *Sign of Four*, Sir Arthur Conan Doyle

Why?

What computer will run my program most cost effectively?



Performance Models

- Predict (parallel) performance without running the code

- So you can
  - buy appropriate hardware
  - run on appropriate resources
  - evaluate new algorithms before implementing them
  - identify unexpected performance problems

- Most importantly, gives **quantitative understanding** of performance
  - enables a scientific approach

Approaches (i)

Technique	Description	Purpose
measurement	running full applications under various configurations	determine how well application performs
microbenchmarking	measuring performance of primitive components of application	provide insight into application performance
simulation	running application or benchmark on software simulation	examine "what if" scenarios e.g. configuration changes
analytical modelling	devising parameterised, mathematical model that represents the performance of an application in terms of the performance of processors, nodes, and networks	rapidly predict the expected performance of an application on existing or hypothetical machines

Approaches (ii)

- Measurement
  - run experiments on real machines
  - fit to Amdahl's law, Gustafson's law, …

- Software Simulation
  - mostly used at the level of a single processor
  - can be done in parallel, e.g. MPI library with adjustable latency
  - "Performance Modeling using Variable Latency MPI", Lee-Shawn Chin, MSc in HPC 2005/06

Approaches (ii)

- Microbenchmarks
  - measure fundamental system properties
    - floating-point performance
    - memory bandwidth
    - message-passing costs (latency, bandwidth, collectives, …)
    - shared-memory overheads (parallel region, barrier, reduction, …)
    - etc.
  - measured by, e.g., Linpack, STREAMS, Intel MPI Benchmark (IMB), EPCC OpenMP Microbenchmarks, …
- Analytical modelling
  - use **measured** fundamental system properties to **predict** entire application performance

Case Study: MPI Coursework

- Simulation parameters (simplified)
  - total number of pixels: L x L
  - number of processors: P
  - decomposition: P x 1 (1D) or $\sqrt{P}$ x $\sqrt{P}$ (2D)

- System properties (simplified)
  - floating-point operations per second: f
  - message-passing latency: T_l
  - message-passing bandwidth: B

Calculation time (per iteration)

- Update

$$new_{i,j} = 0.25 * \left(old_{i-1,j} + old_{i+1,j} + old_{i,j-1} + old_{i,j+1} - edge_{i,j}\right)$$

  - 5 floating-point operations per pixel

- Delta calculation

$$delta = delta + \left(new_{i,j} - old_{i,j}\right) * \left(new_{i,j} - old_{i,j}\right)$$

  - 3 flops per pixel

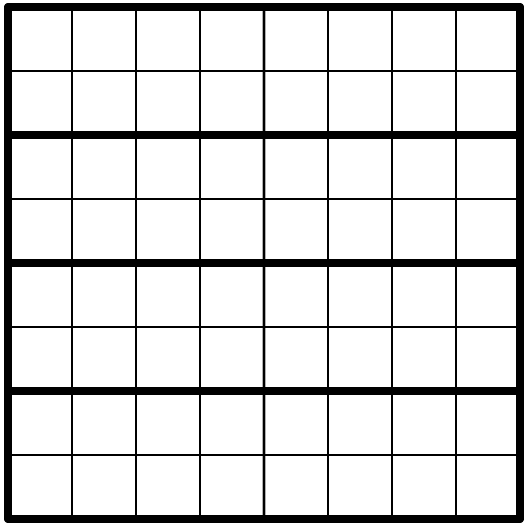
- Time taken: $Time = flops\_per\_pixel * \frac{N_{pixel}}{f}$

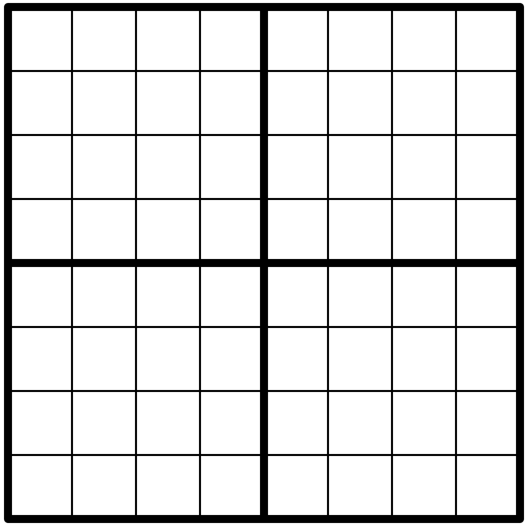
$$N_{pixel} = \frac{L^2}{P}$$

Communication time: halo swaps

- Example:

  L=8, P=4

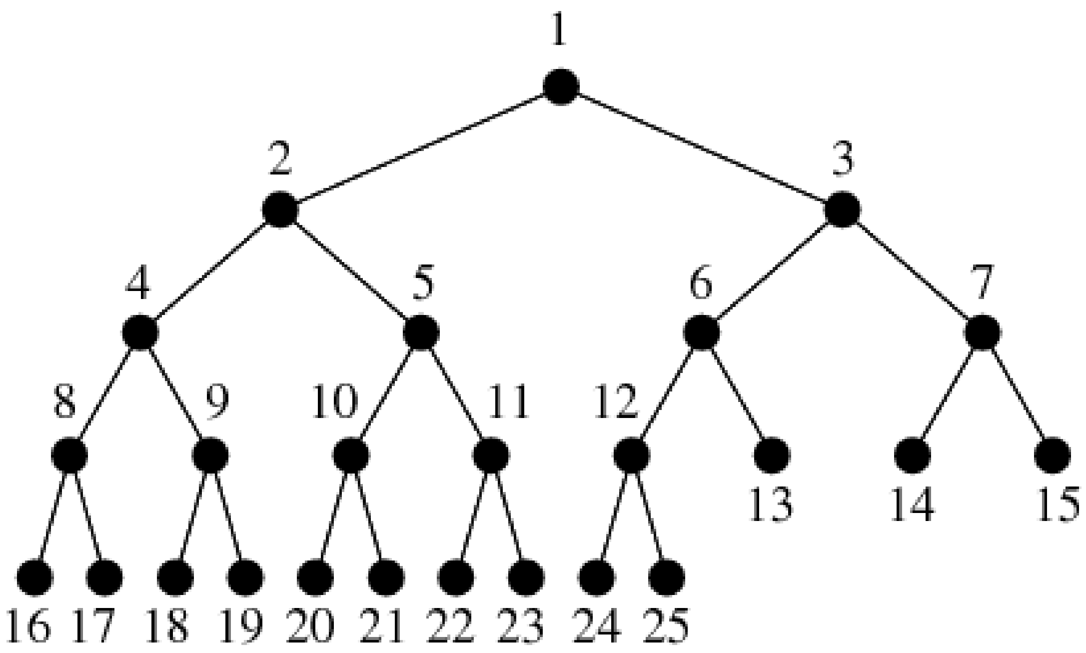




Decomposition	No. messages (large P)	Message Length (doubles)
1D	2	L
2D	4	$L / \sqrt{P}$

- Time per message = T_l + length_in_bytes / B

Communications time: reductions



- Assume a binary tree
  - no. of messages = 2 $\log_2(P)$
  - time taken = 2 $T_l \log_2(P)$

Time per iteration

- 1D Decomposition

1 reduction every *D* iterations

$$time = \frac{5L^2}{fP} + 2\left(T_l + \frac{8L}{B}\right) + \frac{1}{D}\left(\frac{3L^2}{fP} + 2T_l \log_2(P)\right)$$

5 operations per update

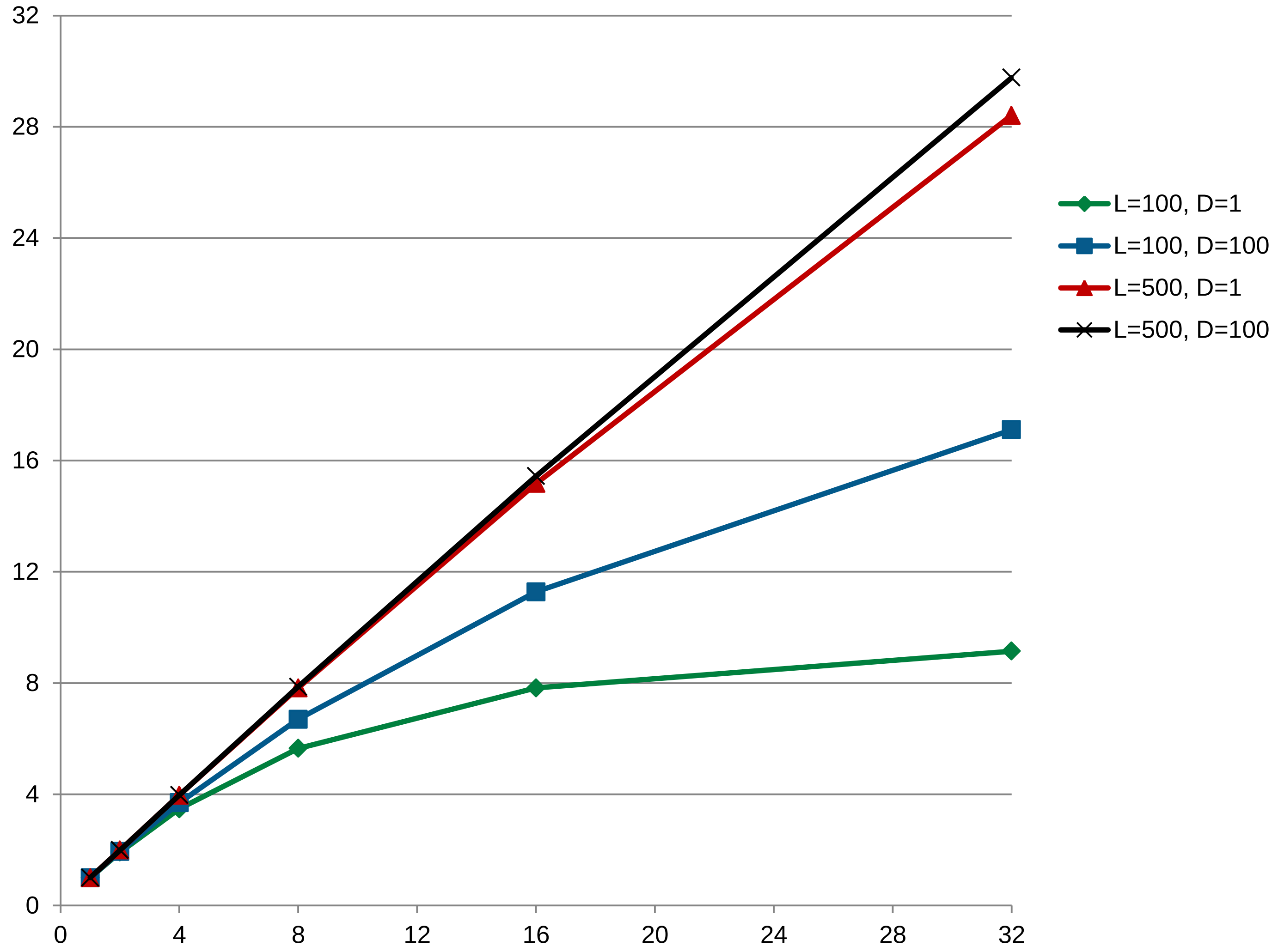
8 bytes (1 double) per point

- 2D Decomposition

$$time = \frac{5L^2}{fP} + 4\left(T_l + \frac{8L}{B\sqrt{P}}\right) + \frac{1}{D}\left(\frac{3L^2}{fP} + 2T_l \log_2(P)\right)$$

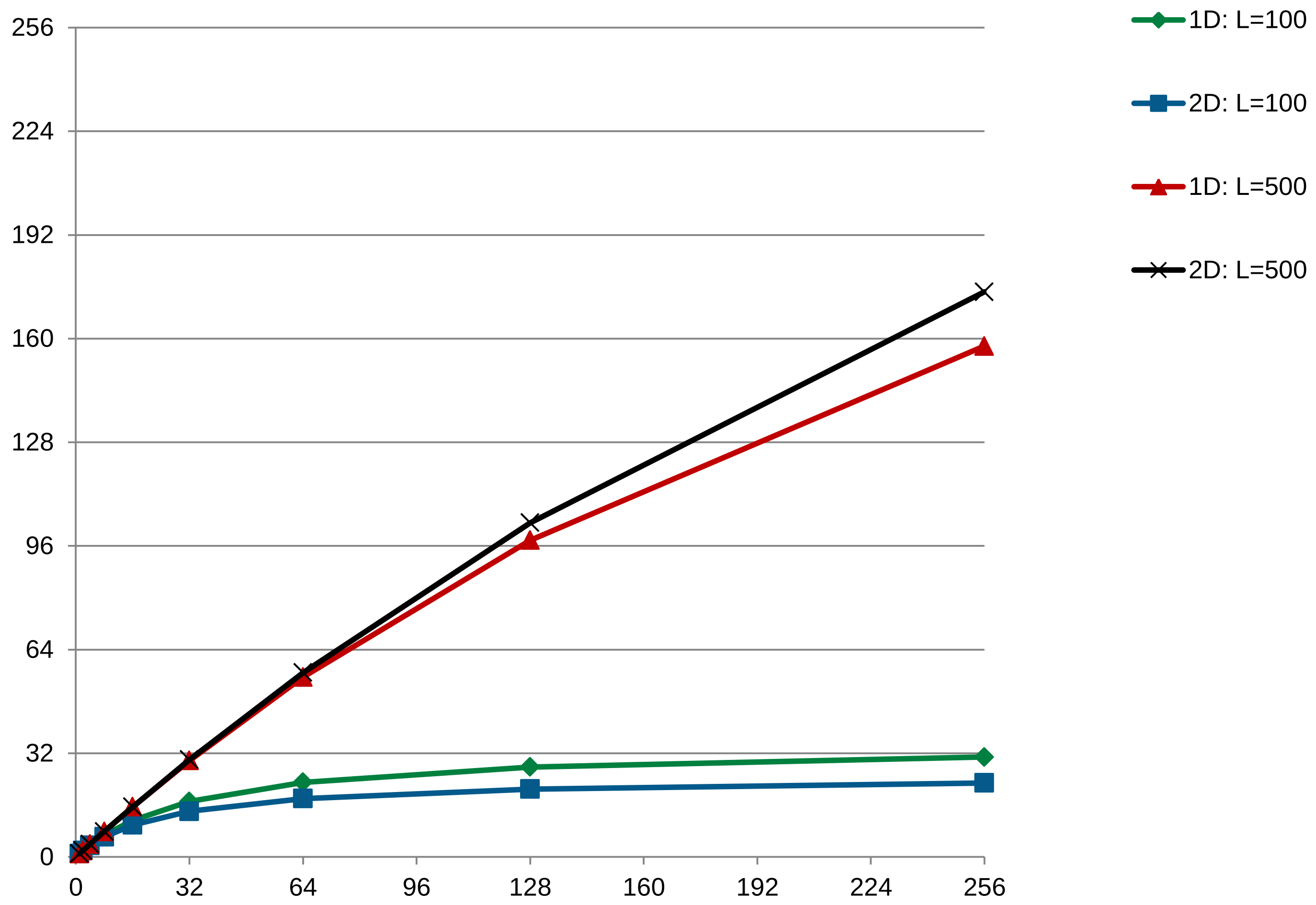
Sample model: 1D speedup

- f = 100 MFLOPs, T_l = 5 μs, B = 400 MB/s



Sample model: 1D vs 2D speedup

- f = 100 MFLOPs, T_l = 5 μs, B = 400 MB/s, D=100



Estimating parameters

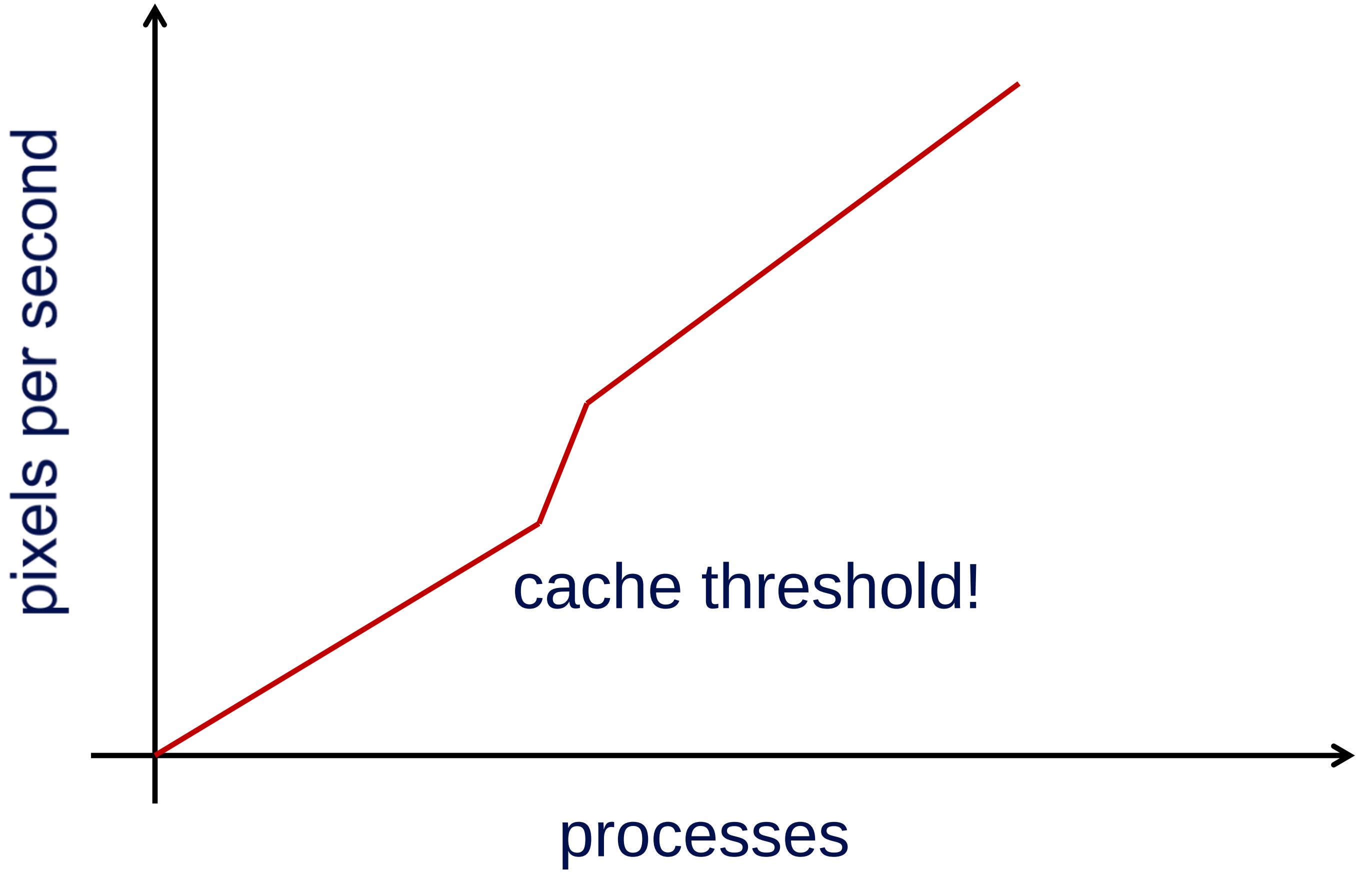
- How do we estimate *f*
  - LINPACK will overestimate it

- Expect that calculation is memory bandwidth dominated
  - measure time for serial image processing program
  - divide by number of pixels

- Similar issues with message-passing
  - ping-pong has all processes idle except for two
  - in halo-swapping, all processes are active …

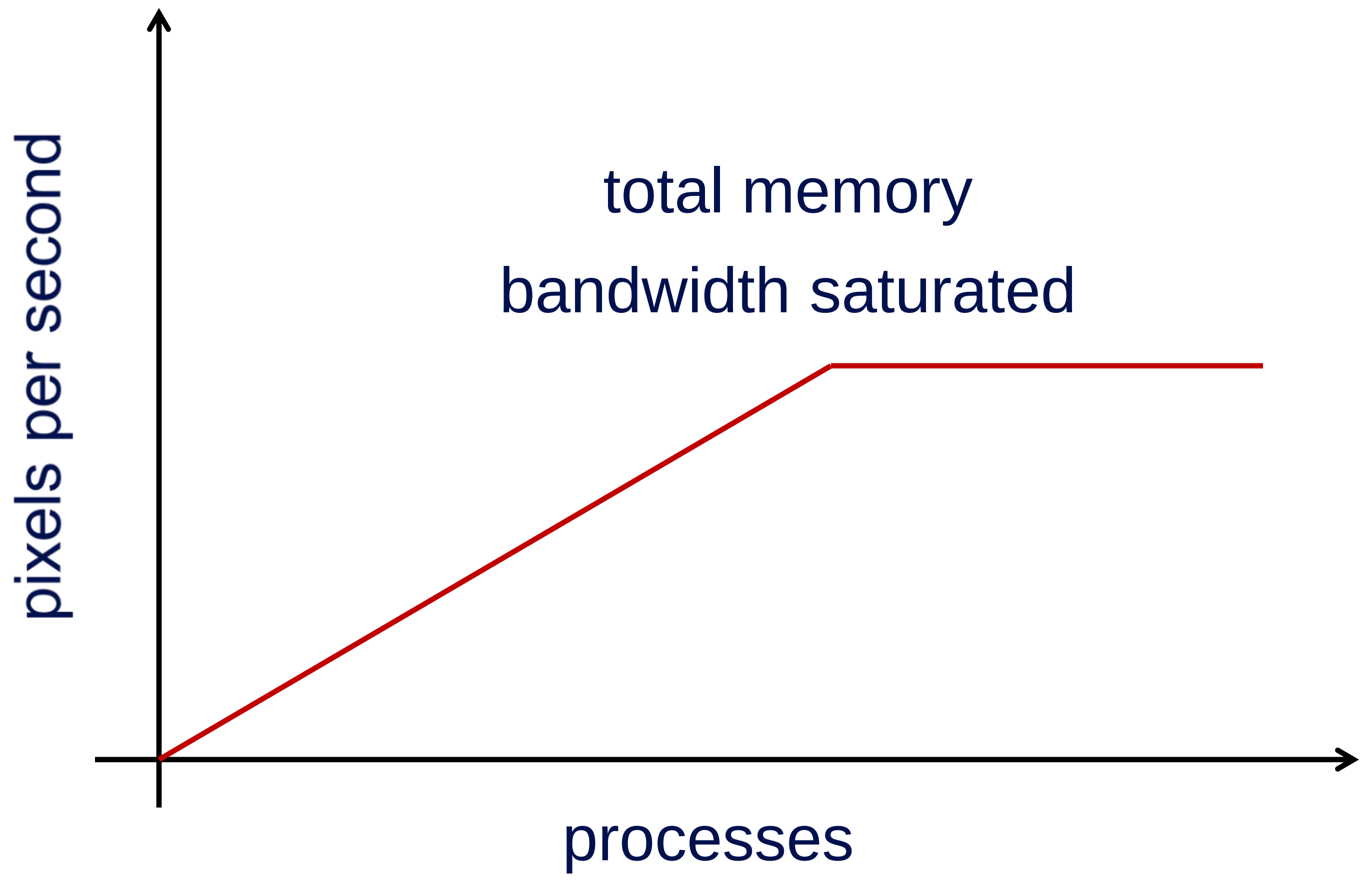
Practical issues

- Simple model ignores some important effects
  - e.g. **strong scaling** (fixed problem size)



Practical issues

- e.g. **weak scaling** (fixed size per processor) on SMP node



Practical performance models

- Don't make equations more and more complex …
- Find out the limiting factors for your code
  - memory bandwidth?
  - floating point performance?
  - MPI bandwidth?
  - MPI latency?
  - MPI collectives?
  - bisection bandwidth?
  - IO?
- This can be enough to decide what areas to investigate

Summary

- Very hard to get a good performance model
  - easy to do the maths and derive impressive (?) equations …
  - … but does it mean anything in practice?

- A good model can be very helpful
  - e.g. see “Missing Performance” paper
  - takes a lot of work to develop and maintain

- Even a simple model can be very useful
  - what is the limiting factor in my code?