Parallel Speedup and Scaling

Troels Henriksen

Gotta go fast!

Parallelism is about speed.

- How do we quantify whether parallelisation made a program faster, and by how much?
- How can we estimate the potential benefit of parallelising a program?
- Are there limits to the potential gains of parallelism?

Latency

Latency (often called *runtime*)

How long it takes for a program to run on some workload.

- Easy to measure, but:
  - May need multiple measurements to eliminate noise.
  - *Especially* if the runtime is low.

Latency

Latency (often called *runtime*)

How long it takes for a program to run on some workload.

- Easy to measure, but:
  - May need multiple measurements to eliminate noise.
  - *Especially* if the runtime is low.
- Measure the right thing:
  - *Wall time* is the time it takes in the real world.
  - *CPU time* is the total amount of time spent executing code *counting all processors*.
  - When we parallelise a program, CPU time remains constant (or increases slightly), while wall time goes down. Consider:
    - 16 processors that that run for 60 seconds simultaneously.
    - 1 processor that runs for 960 seconds.
    - Both take 960 seconds of CPU time, but the former only takes 60 seconds in wall time.

With measurements for two programs, we can compute the *speedup in latency*.

Speedup in latency

If T_1, T_2 are the runtimes of two programs P_1, P_2, then the *speedup in latency* of P_2 over P_1 is

$$\frac{T_1}{T_2}$$

- Suppose a sequential program runs in $25s$ and we write a parallel version that runs in $10s$.
- The speedup is then:

$$\frac{25s}{10s} = 2.5$$

- Speedup greater than one means P_2 is *faster* than P_1, else it is *slower*.
- P_1 and P_2 must *solve the same problem* for their latencies to be comparable.

This is typically what I will expect you to report for your own programs.

Throughput

- Sometimes latency is inappropriate—what is the running time of a potentially-infinite web server?
- Can measure latency for individual requests, but is that really useful?

Throughput

- Sometimes latency is inappropriate—what is the running time of a potentially-infinite web server?
- Can measure latency for individual requests, but is that really useful?

Throughput

The *workload* W processed in some time-span T:

$$Q = \frac{W}{T}$$

- E.g. web requests served per second, or number of bytes processed per clock cycle.
- Using throughput we can compare *programs that have different workloads*.

Speedup in throughput

If Q_1, Q_2 are the throughputs of two programs P_1, P_2, then the *speedup in throughput* of P_2 over P_1 is

$$\frac{Q_2}{Q_1}$$

Suppose

- P_1 sums 1MiB in $69\mu\mathrm{s}$
- P_2 sums 1GiB in $28,489\mu\mathrm{s}$
- Cannot compare these latencies meaningfully, but...

$$Q_1 = \frac{2^{10}B}{69\mu s} = 15196\mathrm{B}/\mu\mathrm{s} = 14.2\mathrm{GiB}/\mathrm{s} \qquad Q_2 = \frac{2^{30}}{28589} = 37558\mathrm{B}/\mu\mathrm{s} = 35.0\mathrm{GiB}/\mathrm{s}$$

Speedup in throughput of P_2 over P_1 is then

$$\frac{35.0\mathrm{GiB}/\mathrm{s}}{14.2\mathrm{GiB}/\mathrm{s}} = 2.46$$

Scalability

Scalability

- How the system improves in its capacity (runtime or throughput) as we add more resources, such as more processors
- How the system throughput changes as we add more workload.

Scalability

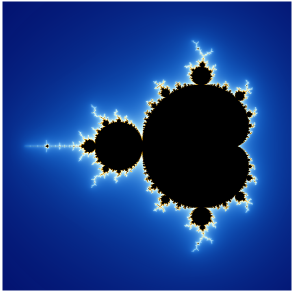
Scalability

- How the system improves in its capacity (runtime or throughput) as we add more resources, such as more processors
- How the system throughput changes as we add more workload.

With respect to parallelism, there are two sorts of scalability that we are interested in.

Example: rendering Mandelbrot fractals.

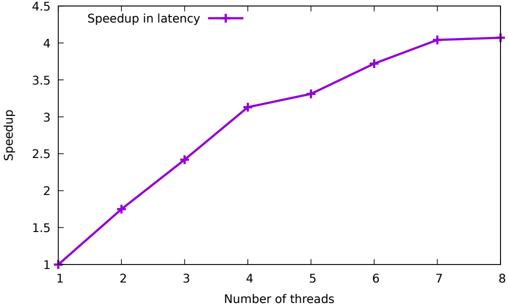
- Each pixel can be computed independently.
- The image size corresponds to the workload.



Strong scaling

How the runtime varies with the number of processors for a fixed problem size.

Speedup graph for rendering a 10^4 pixel Mandelbrot fractal as we use more threads:

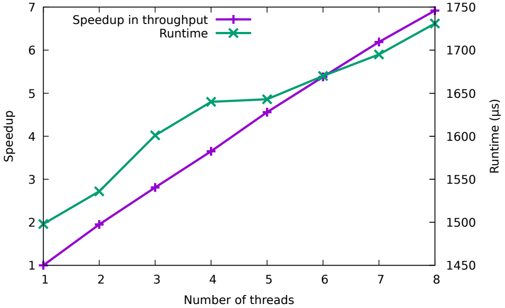


This shows sub-linear strong scalability—8 threads *barely* gets us $4\times$ speedup.

Weak scaling

How the runtime varies with the number of processors for a fixed problem size *relative to the number of processors*.

Performance graph for rendering a Mandelbrot fractal with 10^4 pixels *per thread*:



Pretty decent weak scalability! **By far more common than strong scalability.**

How do we estimate the potential benefits of parallelising a program, without just implementing it and measuring how well it goes?

Amdahl's Law

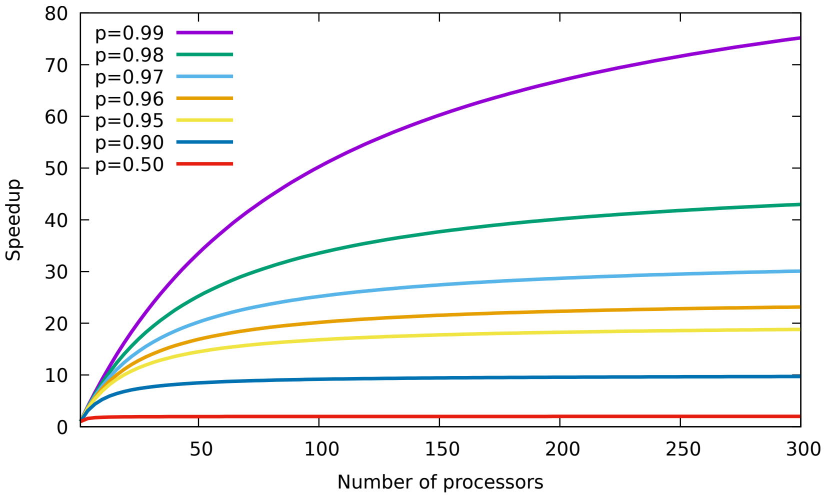
If p is the proportion of execution time that benefits from parallelisation, then $S(N)$ is maximum theoretical speedup achievable by execution on N processors, and is given by

$$S(N) = \frac{1}{(1-p) + \frac{p}{N}}$$

Why is $p \neq 1$?

- Reading input data.
- Writing output data.
- Fundamentally sequential algorithms.

Plotting Amdahl's law



Amdahl's Law predicts strong scalability

- Amdahl's Law predicts *strong scalability*.
- For a *fixed problem size*, there will always come a point of diminishing returns.
- Makes parallelisation seem pointless...

But!

Amdahl's Law predicts strong scalability

- Amdahl's Law predicts *strong scalability*.
- For a *fixed problem size*, there will always come a point of diminishing returns.
- Makes parallelisation seem pointless...

But!

Often it is *time* that is fixed, not the problem!

- You have a simulation that runs in 1 hour.
- Then you get a computer that is ten times faster.
- Usually you don't decide to run the simulation in six minutes, *instead you make a simulation that is ten times as precise* and still runs in 1 hour.
- Consider weather forecasting...
- **Bigger machines** let us solve **bigger problems** in the **same time**!

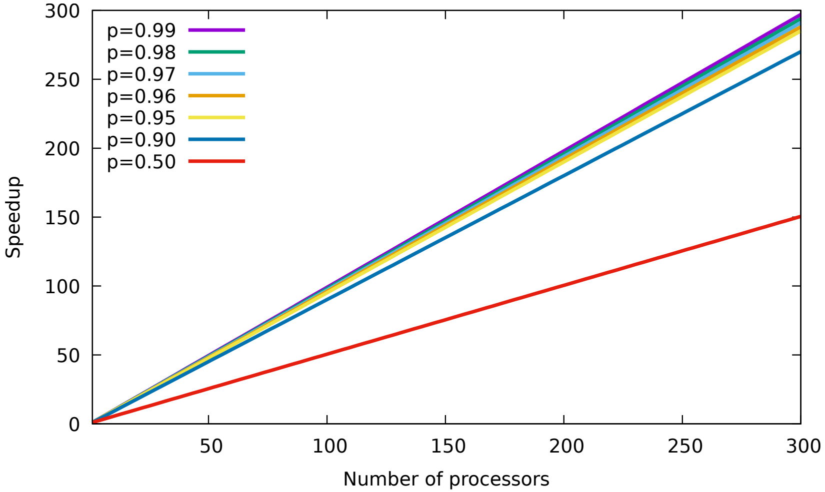
Gustafson's Law

If s is the proportion of execution time that must be sequential, then $S(N)$ is maximum theoretical speedup achievable by execution on N processors, and is given by

$$S(N) = N + (1 - N) \times s$$

- Assumes that *parallel workload* increases just as fast as number of processors.
- Predicts *weak scalability*.
- **In practice often more relevant than Amdahl's Law.**

Plotting Gustafson's law



The fine print

Both Amdahl's and Gustafson's Laws are idealised abstractions and ignore important real-world concerns:

- Locality.
- Communication.
- Synchronisation.

But they are still a good theoretical framework for estimating the value of parallelising some program.

Summary

- Parallelism is about speedup.
- Describe performance differences with speedup.
    - Latency for programs that compute the same thing.
    - Throughput if not.
- Strong scaling is solving the same problem faster.
- Weak scaling is solving a bigger problem in the same time.
- Use Amdahl's Law to predict strong scaling.
- Use Gustafson's Law to predict weak scaling.
- **Measure the real-world scaling of your code for various problem sizes parallelisation degrees.**