Performance analysis

Jeremy Iverson

College of Saint Benedict & Saint John's University

plan for the day

- `reduction`
- theoretical analysis framework
- performing analysis
- experimental analysis
- conducting experiments

theoretical analysis framework

seeks to answer the following questions:

- how do we reason about parallel algorithms?
- how can we compare two algorithms and determine which is better?
- how do we measure improvement?

performance metrics

- execution time (T_s and T_p)
- speedup (S)
- efficiency (E)
- cost (C)

execution time

serial (T_s)

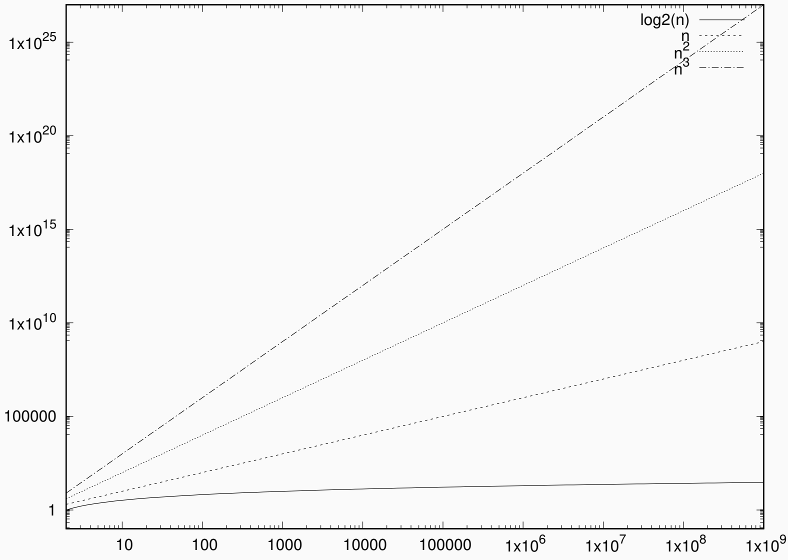
- time elapsed between beginning and end of execution

parallel (T_p)

- time elapsed between beginning of execution and the moment the last processing element finishes execution

- Axpy
- Reduction
- Dot-product
- Matrix-vector multiplication
- Matrix-matrix multiplication

execution time



speedup

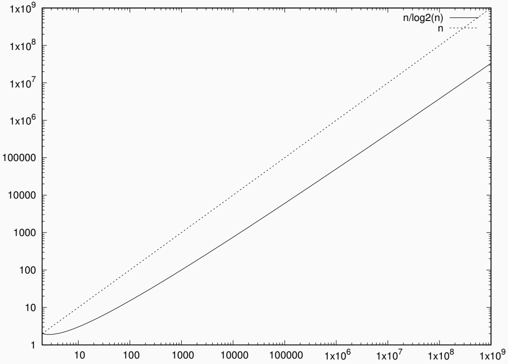
speedup ($S = T_s / T_p$)

- the ratio of time taken to solve a problem on a single processing element to the time required to solve the same problem on a parallel computer with p processing elements

speedup

speedup ($S = T_s/T_p$)

- the ratio of time taken to solve a problem on a single processing element to the time required to solve the same problem on a parallel computer with p processing elements



Amdahl's law

Speedup is limited by the fraction of a parallel program that is serial.

If r is the fraction of the code which is serial, then

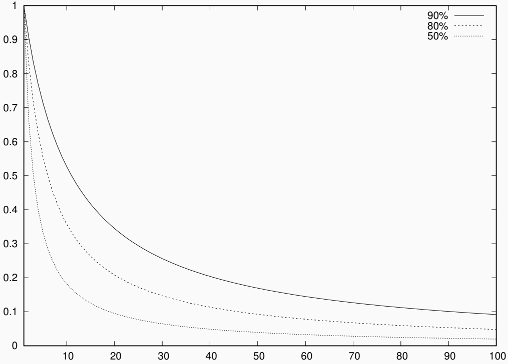
$$S = \frac{T_s}{T_p} = \frac{T_s}{(1-r)T_s/p + rT_s}$$

Amdahl's law

Speedup is limited by the fraction of a parallel program that is serial.

If r is the fraction of the code which is serial, then

$$S = \frac{T_s}{T_p} = \frac{T_s}{(1-r)T_s/p + rT_s}$$



Amdahl's law

Speedup is limited by the fraction of a parallel program that is serial.

If r is the fraction of the code which is serial, then

$$S = \frac{T_s}{T_p} = \frac{T_s}{(1-r)T_s/p + rT_s}$$

In general, we cannot get a speed up better than

$$\frac{1}{r}$$

efficiency

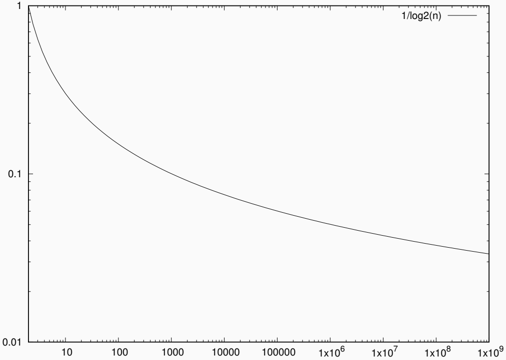
efficiency ($E = S/p$)

- the ratio of speedup to the number of processing elements — the fraction of time for which a processing element is usefully employed

efficiency

efficiency ($E = S/p$)

- the ratio of speedup to the number of processing elements — the fraction of time for which a processing element is usefully employed



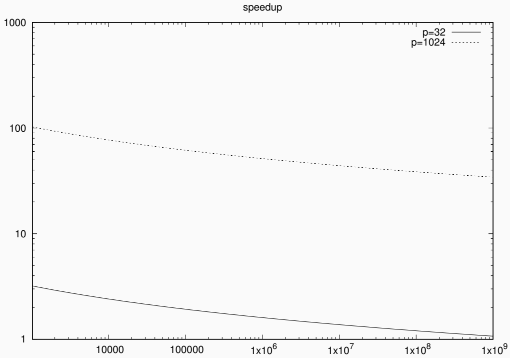
cost

cost ($C = pT_p$)

- the sum of the time spent by all processing elements solving the problem
- *cost-optimal* if $C = T_s$

cost ($C = pT_p$)

- the sum of the time spent by all processing elements solving the problem
- *cost-optimal* if $C = T_s$



exercise — axpy

- $T_p = ?$
- $S = ?$
- $E = ?$
- $C = ?$

exercise — axpy

$p = n$ — cost-optimal

- $T_p = \Theta(n/p)$
- $S = \Theta(\frac{n}{n/p} = p)$
- $E = \Theta(1)$
- $C = \Theta(n)$

exercise — reduction

$p = n$ — *not cost-optimal*

- $T_p = \Theta(\log n)$
- $S = \Theta(\frac{n}{\log n})$
- $E = \Theta(\frac{1}{\log n})$
- $C = \Theta(n \log n)$

$p > n$ — too many processing elements, use less

$p < n$ — ?

exercise — reduction

$p = n$ — *not cost-optimal*

- $T_p = \Theta(\log n)$
- $S = \Theta(\frac{n}{\log n})$
- $E = \Theta(\frac{1}{\log n})$
- $C = \Theta(n \log n)$

$p > n$ — too many processing elements, use less

$p < n$ — *not cost-optimal?*

- $T_p = \Theta(\frac{n}{p} + \log p)$
- $S = \Theta(\frac{n}{\frac{n}{p} + \log p})$
- $E = \Theta(\frac{n}{n + \log p})$
- $C = \Theta(n + p \log p)$

exercise — reduction

$p = n$ — *not cost-optimal*

- $T_p = \Theta(\log n)$
- $S = \Theta(\frac{n}{\log n})$
- $E = \Theta(\frac{1}{\log n})$
- $C = \Theta(n \log n)$

$p > n$ — too many processing elements, use less

$p < n$ — *cost-optimal iff* $n = \Omega(p \log p)$

- $T_p = \Theta(\frac{n}{p} + \log p)$
- $S = \Theta(\frac{n}{\frac{n}{p} + \log p})$
- $E = \Theta(\frac{n}{n + \log p})$
- $C = \Theta(n + p \log p)$

algorithm analysis

case analysis
best, worst, average

asymptotic analysis
Ω, O, Θ

empirical analysis

- use wall clock time
- time only the code that directly corresponds to your algorithm
- repeat experiments multiple times to account for discrepencies in timings
    - take minimum execution time among several trials

scalability

How the execution time varies with the number of processors.

- *strong* scaling — how the solution time varies with the number of processors for a fixed *total* problem size.
- *weak* scaling — how the solution time varies with the number of processors for a fixed problem size *per processor*.

scalability

How the execution time varies with the number of processors.

- *strong* scaling — how the solution time varies with the number of processors for a fixed *total* problem size.
- *weak* scaling — how the solution time varies with the number of processors for a fixed problem size *per processor*.

Algorithms that require the problem size to grow at a lower rate are more scalable.