SFWRENG 4F03
Parallel Computing

Winter 2020

04 Performance and Scalability

Dr Asghar Bokhari

Faculty of Engineering, McMaster University

February 3, 2020

Performance

- Speedup Factor
  - Measure of performance improvement between a single processor and parallel computer is the *speedup factor*, $S(n)$ (n the number of processors).

$$\begin{aligned} S(n) &= \frac{\text{Execution time using single CPU}}{\text{Execution time using } n \text{ processors}} \\ &= \frac{t_s}{t_p} \end{aligned}$$

Performance

- Linear Speedup
  - The maximum speedup factor due to parallelization is n with n processors. This is called *linear speedup.* For example:

$$S(n) = \frac{t_s}{t_s/n} = n$$

  - *Superlinear speedup* is when $S(n) > n$.
  - Unusual, and usually due to a suboptimal sequential algorithm or special features of the architecture which favor parallel programs.
  - Normally, can't fully parallelize a program. Some part must be done serially.
  - There could be periods when only one processor doing work, and others idle.

Performance

- Overhead
  - *Overhead* consists of the time delays that are added due to the parallelization.
  - Factors that cause overhead and reduce speedup are:
    - Periods when all processors are not doing useful work.
    - Additional computations not in serial version.
    - Communication time for sending messages.

Performance

- Speedup Limitations **Amdahl's Law**
  - Normally some fraction of a program must be performed serially
  - Let the serial fraction be $= f$.
  - The fraction that can be parallized $= (1 - f)$
  - Assume no overhead results when program divided into concurrent parts,
  - Then computation time with n processors is:

$$t_p = ft_s + (1 - f)t_s/n$$

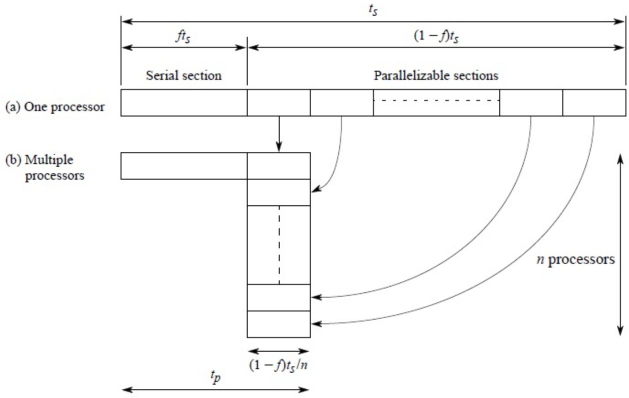
  - Amdahl's Law:

$$S(n) = \frac{t_s}{ft_s + (1 - f)t_s/n} = \frac{n}{1 + (n - 1)f}$$

  - Limit of speedup with infinite number of processors:

$$S(n)_{n \to \infty} = \frac{1}{f}$$

Amdahl's Law



Parallelizing sequential problem — Amdahl's law. [1]

[1] B. Wilkinson, and M. Allen, *Parallel Programming. Techniques and Applications Using Networked Workstations and Parallel Computers*, Prentice-Hall, 1999.

Amdahl's Law

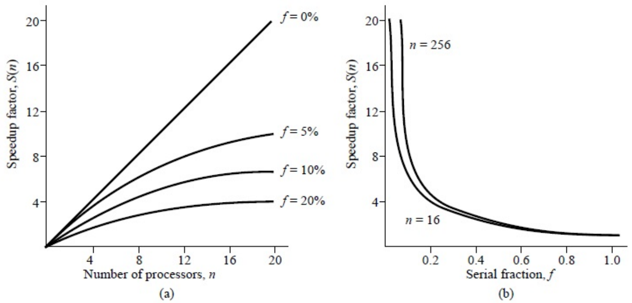


Figure 1.30 (a) Speedup against number of processors. (b) Speedup against serial fraction, f.

[2]

[2] B. Wilkinson, and M. Allen, *Parallel Programming. Techniques and Applications Using Networked Workstations and Parallel Computers*, Prentice-Hall, 1999.

Performance

- Efficiency
  - The system *efficiency*, E, is defined as:

$$\begin{aligned} E &= \frac{\text{Execution time with single CPU}}{\text{Execution time with multiprocessor} \times \text{number of CPUs}} \\ &= \frac{t_s}{t_p \times n} \end{aligned}$$

  - Efficiency as a percentage is given by:

$$E = \frac{S(n)}{n} \times 100\ \%$$

  - The *processor-time* product, or *cost* of a computation is:

$$Cost = t_p \times n = \frac{t_s n}{S(n)} = \frac{t_s}{E}$$

Performance

- Gustafson's Law
  - Gustafson argued that Amdahl's law is not as limiting as it seems.
  - Observed that a large multiprocessor means you can handle bigger problems.
  - In practice, problem size is related to number of processors.
  - Assume parallel processing time fixed, not problem size.
  - Gustafson claimed that serial section doesn't increase as problem size increases.
  - Let s be the serial computation time. Let p be the time required to execute the parallel portion of the program on a single processor.
  - Assume $s + p$ is fixed. For convenience, we choose $s + p = 1$, so they represent fractions.

Performance

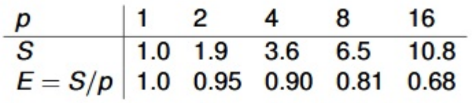
- Gustafson's Law
  - We can then represent Amdahl's law as:

$$S(n) = \frac{s+p}{s+p/n} = \frac{1}{s+(1-s)/n}$$

  - For Gustafson's *scaled speedup factor*, $S_s(n)$, he assumes the parallel execution is constant, and the serial time (t_s) changes.
  - Again, we have $s+p=1$. Execution time on single processor is now $s+pn$. This gives us:

$$S_s(n) = \frac{s+np}{s+p} = s+np = n+(1-n)s$$

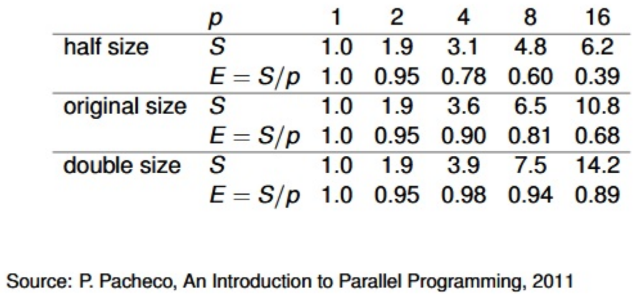
Amdahl's Law



p	1	2	4	8	16
S	1.0	1.9	3.6	6.5	10.8
$E = S/p$	1.0	0.95	0.90	0.81	0.68

Source: P. Pacheco, An Introduction to Parallel Programming, 2011

Gustafson's Law



	p	1	2	4	8	16
half size	S	1.0	1.9	3.1	4.8	6.2
	$E = S/p$	1.0	0.95	0.78	0.60	0.39
original size	S	1.0	1.9	3.6	6.5	10.8
	$E = S/p$	1.0	0.95	0.90	0.81	0.68
double size	S	1.0	1.9	3.9	7.5	14.2
	$E = S/p$	1.0	0.95	0.98	0.94	0.89

Source: P. Pacheco, An Introduction to Parallel Programming, 2011

GFLOPs Performance

- Performance measured by speed up factor
  - If the size of a problem is fixed, speedup calculated by using Amdahl's law
  - If the size is allowed to linearly increas, use Gustafson's law for speed up.
- If the peak performace of a processor is known in GFLOPS, then performnce of the parallel system is = CPU peak performance x Speed up factor.

Performance

- Scalability
  - Has different meanings.

    Architecture or Hardware Scalability: A hardware design that permits system to be enlarged which results in increased performance. Usually, adding more processors means network must be increased. Means greater delay and contention thus lower efficiency.

    Algorithmic Scalability: Means parallel algorithm can handle increase in data with low and bounded increase in computational steps.

Performance

- Scalability Continued ..
  - In general, an algorithm is scalable if we can handle increasing problem sizes
    $E = \frac{n}{p(n/p+1)} = \frac{n}{n+p}$
  - Increase problem size by x and processors by k
    $E = \frac{nx}{nx+pk}$
  - Increasing both to same extent (e.g. x = k) possible to keep E same.
  - Strongly scalable
    - Problem size is fixed
    - Increase number of processes
    - The efficiency is about the same
  - Weakly scalable
    - Increase problem size and number of processes at the same rate
    - The efficiency is about the same

Performance

- Computation-Communication Ratio
  - Communication overhead for message passing
  - Want to reduce amount of intercomputer communication
  - There's a point when the communication time dominates execution time

$$\text{Computation/communication ratio} = \frac{\text{Computation time}}{\text{Communication time}} = \frac{t_{comp}}{t_{comm}}$$

  - Want to maximize ratio, but still maintain sufficient parallelism.

Performance

- Example
  - Consider two computers connected by a network.
  - Each CPU has clock period T, and the communication latency for the network is L.
  - Each process (1 per CPU) must process 10 commands.
  - Each command takes 30 clock cycles to execute. Process 1 then sends a message to process 2, who then replies.
  - For our system, T = 12.5ns, and L = 500us.

    $t_{comp} = 10 \cdot 30 \cdot T = 10 \cdot 30 \cdot 12.5 \times 10^{-9}s = 3.75 \times 10^{-6}s$

    $t_{comm} = 2 \cdot L = 2 \cdot 500 \times 10^{-6}s = 0.01s$

    $ratio = \frac{t_{comp}}{t_{comm}} = \frac{3.75 \times 10^{-6}s}{0.01s} = 0.00375$

    NOTE: total execution time of task is: $t_{comp} + t_{comm}$

Performance

- Comments
  - Granularity related to number of processors.
  - For domain decomposition (partitioning the data), the size of the per CPU data can be increased to improve ratio.
  - For a fixed data size, could reduce number of CPUs used.
  - Want to design a program where it is easy to vary granularity.

Title

-
-
- Template!