Speedup, Efficiency, Scalability

Ned Nedialkov

McMaster University

31 January 2023

Outline

- Speedup
- Efficiency
- Amdahl's law
- Gustafson's law
- Sources of overhead

Speedup

- Given a problem, let $T(1)$ be the time for the fastest (practical) sequential algorithm to solve it (on one processor).
- Let $T(p)$ be the time to solve it in parallel on p processors.
- Speedup is defined as

$$S(p) = \frac{T(1)}{T(p)}.$$

- In practice, often $T(1)$ is the time for the parallel program to run on a single processor.
- Theoretically, $S(p) \leq p$.
- In practice, $S(p) > p$ is possible: superlinear speedup. E.g.
  - the data of a serial program may not fit into cache, but could fit in a parallel version
  - in a parallel search algorithm a processor may find a solution quickly and all processors finish

Efficiency

- Defined as

$$E(p) = \frac{S(p)}{p} = \frac{1}{p}\frac{T(1)}{T(p)}$$

- $0 < E(p) \leq 1$.
- It is a measure of the fraction of time for which a processing element is used.

Example

On my 8-core Mac I get the following times, speedups and efficiencies for multiplying 100 times a $10,000 \times 10,000$ matrix times vector:

p	1	2	4	8	16
time (sec)	9.4	4.9	2.6	1.3	2.3
$S(p)$	1.00	1.92	3.62	7.23	4.09
$E(p)$	1.00	0.96	0.90	0.90	0.26

Amdahl's law

Assume

-
  - a program needs 10 hours on a single processor
  - one hour is sequential, the rest 9 hours can be parallelized
  - cannot run in less than an hour
  - speedup is at most 10
- Speedup is limited by the sequential part.
- Note: the problem size is fixed.

Amdahl's law cont.

- Let r be the fraction of statements that can be executed in parallel.
- Then $(1 - r)$ is the fraction of statements that is inherently serial.
- $T(1)$ is serial time, $T(p)$ is parallel time.
- Time for serial part: $(1 - r)T(1)$.
- Total time with p processors:

$$T(p) = r\frac{T(1)}{p} + (1 - r)T(1)$$

Amdahl's law cont.

- Then

$$S(p) = \frac{T(1)}{T(p)} = \frac{T(1)}{r\frac{T(1)}{p} + (1-r)T(1)} = \frac{1}{\frac{r}{p} + (1-r)}.$$

- $S(p)$ increases as $p \to \infty$
- $\lim_{p\to\infty} S(p) = \frac{1}{1-r}$
- $S(p) \leq \frac{1}{1-r}$
- Speedup cannot exceed $\frac{1}{1-r}$

Amdahl's law cont.

- $S \leq \frac{1}{1-r}$
- E.g. if
  - $r = 0.5$, $S(p) \leq 2$
  - $r = 0.75$, $S(p) \leq 4$
  - $r = 0.99$, $S(p) \leq 100$

Gustafson's law

- Assume that we execute a program on p processors.
  Denote the time of sequential execution by a and the time of the parallel part by b. Then

$$T(p) = a + b.$$

- Assume that the work to be done in parallel varies linearly with p.
- Then, if we execute the program serially,

$$T(1) = a + p \cdot b.$$

  (b work on each processor times p, $p \cdot b$)

Gustafson's law cont.

- Denote the fraction of the serial time by

$$F_s = \frac{a}{a+b} = \frac{\text{time of serial part}}{T(p)}.$$

  The fraction of the parallel part is $F_p = \frac{b}{a+b} = 1 - F_s$.

- Then

$$\begin{aligned} S(p) &= \frac{T(1)}{T(p)} = \frac{a + p \cdot b}{a+b} \\ &= \frac{a}{a+b} + p\frac{b}{a+b} \\ &= F_s + p(1 - F_s) \\ &= p - F_s(p-1) \end{aligned}$$

- If F_s small, $S(p)$ approaches p as p increases

- Amdahl's law: we keep the size fixed, but increase the number of processors.
- Gustafson's law: we increase both problem size and number of processors.

Sources of overhead

- Interprocess communication
- Idling
  - load imbalance: a problem is not subdivided equally
  - synchronization
  - serial component in a program: a process may execute a sequential part, while the remaining processes are waiting
- Excess computation
  - the fastest serial algorithm may be difficult or impossible to parallelize
  - a poorer algorithm may be easier to parallelize
  - excess computation $=$
    (computation performed by the parallel algorithm) $-$
    (computation performed by the best serial algorithm)
    E.g. serial (quicksort, mergesort) are $O(n \log n)$ but parallel bitonic sort is $O(n \log^2 n)$.

Scalability

- In general, an algorithm is scalable if we can handle increasing problem sizes.
- Strongly scalable
  - problem size is fixed
  - we increase number of processes/threads
  - the efficiency is about the same
- Weakly scalable
  - we increase problem size and number of processes/threads at the same rate
  - the efficiency is about the same