Performance Analysis Metrics

Ricardo Rocha, Fernando Silva e Eduardo R. B. Marques

Departamento de Ciência de Computadores
Faculdade de Ciências
Universidade do Porto

Computação Paralela 2018/19

Performance and scalability

Key aspects:

- **Performance**: reduction in computation time as computing resources increase
- **Scalability**: the ability to maintain or increase performance as the computing resources **and/or** the problem size increases.

What may undermine performance and/or scalability?

- **Architectural limitations**: latency and bandwidth, data coherency, memory capacity.
- **Algorithmic limitations**: lack of parallelism (sequential parts of computation), communication and synchronization overheads, poor scheduling / load balance.

Performance metrics

Metrics for processors/core

- Apply to single processors, cores, or entire parallel computer.
- Measure the number of operations the system may accomplish per time-unit.
- Benchmarks are used without concern for measuring speedup or scalability.

Metrics for parallel applications – our main interest:

- Assess the performance of a parallel application, in terms of speedup or scalability.
- Account for variation in execution time (and its subcomponents) of an application as the number of processors and/or the problem size increase.

Metrics and benchmarks for processors/core

- Typical metrics:
  - **MIPS**: Million Instructions Per Second
  - **MFLOPS**: Millions of FLOating point Operations Per Second
  - Derived metrics are sometimes employed in order to normalize the impact of aspects such as processor clock frequency.
- **Single processor, general-purpose benchmarks**
  - **SPEC CPU** = SPECint + SPECfp – widely used, apply only to single processing units (single-core CPUs or 1 core in a multi-core processor, hyperthreading is disabled).
  - Historical, influential benchmarks in academia: **Whetstone** and **Dhrystone**, also mostly directed to single-processor/core performance.
- **Specific to parallel computers**
  - **LINPACK**
  - **HPCG**

Performance Metrics for Parallel Applications

"Direct" metrics, derived from comparing sequential vs. parallel execution time:

- **Speedup**
- **Efficiency**

"Laws" and metrics that help us quantify performance bounds for a parallel application:

- **Amdhal's law**
- **Gustafson-Barsis' law**
- **Karp-Flatt metric**
- The **isoeffiency relation** and the (memory) **scalability metric**

Speedup and Efficiency

- Let $T(p, n)$ be the execution time of a program with p processors for a problem of size n.
- **Sequential execution time** $= T(1, n)$.s
- **Speedup**, a direct measure of performance:

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

- **Efficiency**, provides a normalized metric for performance, illustrating scalability more clearly:

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

- Example (assuming some fixed n):

p	1	2	4	8	16
T	1000	520	280	160	100
S	1	1.92	3.57	6.25	10.0
E	1	0.96	0.89	0.78	0.63

Speedup and Efficiency

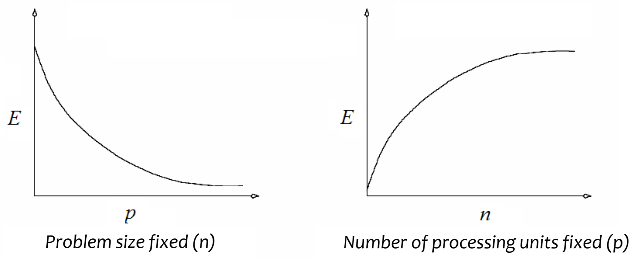
Reasoning on speedup / efficiency:

- Ideal scenario:
  - $S(p, n) \approx p \Leftrightarrow E(n, p) \approx 1$ — **linear speedup**.
  - Perfect parallelism: the execution of the program in parallel has no overheads.
- Most common scenario, as p increases:
  - $S(p, n) < p \Leftrightarrow E(n, p) < 1$ — **sub-linear speedup**.
  - $E(p_1, n) > E(p_2, n)$ for $p_1 < p_2$: efficiency decreases as the number of processors increase.
  - Parallel execution overheads typically increase with p.

Super-linear speedup

- Less often, we may have $\boldsymbol{S(p) > p \Leftrightarrow E(p) > 1}$ — **super-linear speedup** – and $\boldsymbol{E(p_1, n) < E(p_2, n)}$ for $\boldsymbol{p_1 < p2}$.
- Possible reasons for super-linear speed-up may include:
  - Better memory performance, due to higher cache hit ratios and/or lower memory usage;
  - Low initialization/communication/synchronization costs;
  - Improved work division / load balance;

Speedup and efficiency



Typically:

- For fixed ***n*** (shown left), efficiency decreases as ***p*** grows. Parallel execution overheads due to aspects such as communication or synchronization tend to grow with ***p***.
- For fixed ***p*** (shown right), efficiency increases with ***n*** – a trait known as **the Amdhal effect**. The significance of parallel execution overheads in total execution time tends to decrease as ***n*** increases.

Modelling performance

$T(p, n)$, the execution time of a program using p processors for a problem size of n, can be modelled as:

$$T(p, n) = \text{seq}(n) + \frac{\text{par}(n)}{p} + \text{ovh}(p, n)$$

where:

- $\text{seq}(n)$: time for computation that can only be performed sequentially (e.g., reading input, writing output results);
- $\text{par}(n)$: time for computation that can be performed in parallel [1]
- $\text{ovh}(p, n)$: overhead time of running the program in parallel (e.g., synchronization, communication, redundant operations)

Given that $\text{ovh}(1, n) = 0$ the **sequential execution time** is given by:

$$T(1, n) = \text{seq}(n) + \text{par}(n)$$

[1] the fact that it does not depend on p may be a simplification, why?

Modelling performance(2)

Under the previously considered model, we get the following formula for speedup:

$$S(p,n) = \frac{T(1,n)}{T(p,n)} = \frac{\mathbf{seq}(n) + \mathbf{par}(n)}{\mathbf{seq}(n) + \mathbf{par}(n)/p + \mathbf{ovh}(p,n)}$$

Note: for simpler notation, we will omit the p and n arguments for $S, \mathbf{seq}, \mathbf{par}, \mathbf{ovh}$ when clear in context.

Amdhal's law

- Amhdal asked: **If $f \in [0,1]$ is the fraction of computation (in the sequential program) that can only be executed sequentially, what is the maximum possible speedup?**
- Considering our model, we have:

$$f = \frac{\mathbf{seq}}{\mathbf{seq} + \mathbf{par}}$$

- Amdahl's reasoning discards **ovh** ≥ 0 for a speedup upperbound:

$$S = \frac{seq + par}{seq + par/p + comm} \leq \frac{seq}{seq + par/p}$$

- We may then obtain:

$$S \leq \frac{\text{seq} + \text{par}}{\text{seq} + \frac{\text{par}}{p}} = \frac{\text{seq} + \text{par}}{\frac{p-1}{p}\,\text{seq} + \frac{\text{seq} + \text{par}}{p}} = \frac{\text{seq}\,/f}{\frac{p-1}{p}\,\text{seq} + \frac{\text{seq}\,/f}{p}}$$

$$= \frac{\text{seq}/f}{\frac{p-1}{p}\text{seq} + \frac{\text{seq}(n)/f}{p}} = \frac{1/f}{\frac{p-1}{p} + \frac{1}{f\,p}} = \frac{1}{\frac{f\,(p-1)}{p} + \frac{1}{p}} = \frac{\mathbf{1}}{f + (1-f)/p}$$

Amdhal's law

Let $f \in [0, 1]$ be the fraction of operations in a program that can only be executed sequentially.

The maximum speedup that can be achieved by a program with p processors is:

$$S \leq \frac{1}{f + (1 - f)/p}$$

Observe also that

$$\lim_{p \to +\infty} \frac{1}{f + (1 - f)/p} = \frac{1}{f}$$

and that in any case $\mathbf{S} \leq \frac{1}{f}$.

Applying Amdhal's law – example

Program `Foo` spends 90 % of the running time in computation that can be parallelized. Using Amdhal's law, estimate the maximum speedup:

1. when using $\mathbf{8}$ and $\mathbf{16}$ processors;
2. when using an arbitrary number of processors;

Resolution:

1. We have $f = 0.1$ thus $S \leq \frac{1}{0.1 + 0.9/p}$. This means that $S \leq 4.8$ for $p = 8$ and $S \leq 6.7$ for $p = 16$.
2. $S \leq \frac{1}{0.1} = 10$.

Limitations of Amdhal's law

- Amdhal's law does not account for $\mathbf{ovh}(p, n)$, Thus, it may provide a too optimistic upper bound for the speedup!
- Suppose that we have a parallel program where $\mathbf{seq} = n + 1000, \mathbf{par} = n^2/10, \mathbf{ovh} = 10\,(p-1)\log n$.
- This gives us $f = \frac{n+1000}{n+1000+n^2/10}$.
- The following table compares $S = (\mathbf{seq} + \mathbf{par})/(\mathbf{seq} + \mathbf{par}\,/p + \mathbf{ovh})$ with Amdhal's bound (in blue).

	$n = 100, f = 0.52$		$n = 200, f = 0.23$		$n = 400, f = 0.08$		$n = 800, f = 0.02$	
$p = 2$	1.28	1.31	1.60	1.63	1.84	1.85	1.94	1.95
$p = 4$	1.41	1.56	2.20	2.36	3.12	3.22	3.66	3.70
$p = 8$	1.36	1.71	2.51	3.06	4.56	5.12	6.41	6.71
$p = 16$	1.13	1.81	2.32	3.59	5.27	7.25	9.67	11.34
$p = 32$	0.82	1.86	1.75	3.92	4.63	9.16	11.21	17.32
$p = 64$	0.52	1.88	1.13	4.12	3.21	10.55	9.38	23.50
$p \to \infty$		1.92		4.34		12.50		50

From Amdhal's law to Gustafson-Barsis Law

- Amdhal's law demonstrates that speedup increases as the number of processors increases too, but usually assuming a fixed problem size (n) and making a prediction based on the *sequential version* of a program.
- Gustafson and Barsis (in "Reevaluating Amdahl's Law", 1988) shift the focus by trying to estimate maximum speedup, based on the *parallel version* of a program.
- As a basis of their argument, they consider s to be the fraction of *parallel* computation that is devoted to inherently sequential computations, i.e.,

$$s = \frac{seq}{seq + par/p}$$

Deriving Gustafson-Barsis' Law

- As introduced previously, let

$$\boldsymbol{s = \frac{\text{seq}}{\text{seq} + \text{par}\,/p}} \quad \text{and note that} \quad \boldsymbol{1 - s = \frac{\text{par}\,/p}{\text{seq} + \text{par}\,/p}}$$

- Thus $\boldsymbol{\text{seq} = (\text{seq} + \text{par}\,/p)\; s}$ and $\boldsymbol{\text{par} = (\text{seq} + \text{par}\,/p)(1 - s)p}$
- As in the derivation of Amdahl's law, ignore **ovh** to obtain

$$\boldsymbol{S \leq \frac{\text{seq} + \text{par}}{\text{seq} + \text{par}\,/p}}$$

- We then have

$$\begin{aligned} S \leq \frac{\text{seq} + \text{par}}{\text{seq} + \text{par}\,/p} &= \frac{(\text{seq} + \text{par}\,/p)(s + (1 - s)p)}{\text{seq} + \text{par}\,/p} \\ &= s + (1 - s)\; p \\ &= \boldsymbol{p + (1 - p)\; s} \end{aligned}$$

Gustafson-Barsis' Law

Given a parallel program solving a problem of size n using p processors, let s be the fraction of total execution time spent in serial code.

The maximum achievable speedup is:

$$S \leq p + (1 - p)\, s$$

Gustafson-Barsis' speedup upperbound is called the scaled speedup.

Gustafson-Barsis' Law – example application

A profile of program `Foo` running on $\mathbf{16}$ processors revealed that $s = 5\%$ of the time is spent on inherently sequential computation.

1. What is the scaled speedup for the $\mathbf{16}$ processors?
2. What is the scaled speedup prediction for $\mathbf{32}$ processors?

Resolution:

1. For $s = 0.05, p = 8$ the scaled speedup is $S \leq p + (1 - p)s = 16 - 15 \times 0.05 = 15.25$.
2. For $s = 0.05, p = 16$ we have $S \leq 32 - 31 \times 0.05 = 30.45$.

Gustafson-Barsis' Law – example application (2)

We wish that program `Foo` running on 1024 processors achieves a speedup of **800** for a certain problem.

1. What is the maximum fraction s of parallel execution that can be devoted to inherently sequential computation?
2. What about in the case of a desired speedup of **900**?

Resolution:

1. $800 \leq 1024 - 1023\, s \Leftrightarrow s \leq 224/1023 \approx 0.21$.
2. $900 \leq 1024 - 1023\, s \Leftrightarrow s \leq 124/1023 \approx 0.12$.

Karp-Flatt metric

- Amdhal's law and Gustafson-Barsis' law ignore $\mathbf{ovh}(p, n)$, the overhead of parallel computation, overestimating possible speedup.
- Karp and Flatt propose another metric that takes $\mathbf{ovh}(p, n)$ into account, called the **experimentally determined serial fraction** e of the parallel computation, defined as:

$$e = \frac{\mathrm{seq}(n) + \mathrm{ovh}(p, n)}{\mathrm{seq}(n) + \mathrm{par}(n)} = \frac{\mathrm{seq}(n) + \mathrm{ovh}(p, n)}{T(1, n)}$$

- Thus e can be seen as the fraction of serial computation, **including parallel overheads**.
- e can be rewritten in as (derivation omitted):

$$e = \frac{1/S - 1/p}{1 - 1/p}$$

- The metric is useful to provide other insights into performance beyond speedup.

Karp-Flatt metric

Given a parallel program with speedup S on $p > 1$ processors, the experimentally determined serial fraction e is defined as:

$$e = \frac{1/S - 1/p}{1 - 1/p}$$

Applications of the Karp-Flatt metric

For fixed n, the efficiency E of a parallel program typically decreases as the number of processors p increase.

The Karp-Flatt metric is useful to identify the reasons for that decrease in efficiency **a posteriori**, i.e., from the results of program execution since it depends on S (can be measured) and p (a known value):

- If e **does not increase with** p, the decrease in E should relate to **lack of parallelism in the program**.
- If e **increases with** p, the decrease in E is explained by **algorithmic/architectural overheads in the parallelisation** (**ovh**).

Applications of the Karp-Flatt metric – examples

- Example 1:

p	2	4	8	16	32
S	1.994	3.943	7.553	12.932	16.438
E	0.997	0.986	0.944	0.808	0.514
e	0.003	0.005	0.008	0.016	0.031

The decrease in E is explained by the increase in e. The program suffers from greater overhead in parallel execution as p increases.

- Example 2:

p	2	4	8	16	32
S	1.978	3.873	7.430	13.729	23.768
E	0.989	0.968	0.929	0.858	0.743
e	0.011	0.011	0.011	0.011	0.011

E decreases but e remains stable, as p increases. The program suffers from lack of parallelism as p increases.

Isoefficiency relation and the scalability metric

- E typically increases n and decreases with the number of processors p.
- This begs the question: **to maintain the same level of efficiency, when p is increased, how should n be also increased?**
- Follow-up question: is the increase in n sustainable in memory terms? How does the program scale in terms of memory requirements?
- To help answer these questions Grama et al. introduced the **isoefficiency relation** and the **scalability metric** ("Isoefficiency: Measuring the scalability of parallel algorithms and architectures", IEEE Parallel & Distributed Technology: Systems & Technology, 1(3):21, 1993).

Isoefficiency relation — derivation

- Let $T_0(p, n)$ be the **total** amount of time spent by all processors in the parallel program performing work not done by the serial program, i.e.:

$$T_0(p, n) = (p - 1)\,\mathrm{seq}(n) + p\,\mathrm{ovh}(p, n)$$

- It can be shown that:

$$E(p, n) \geq \frac{1}{1 + \frac{T_0(p,n)}{T(1,n)}} \Leftrightarrow T(1, n) \geq \frac{E(p, n)}{1 - E(p, n)} T_0(p, n)$$

- The isoefficiency relation is then written as:

$$T(1, n) \geq C\ T_0(p, n) \text{ where } C = \frac{E(p, n)}{1 - E(p, n)}$$

Isoefficiency relation

Let

$$T_0(p,n) = (p-1)\,\mathrm{seq}(n) + p\ \mathrm{ovh}(p,n)$$

and

$$C = \frac{E(p,n)}{1 - E(n,p)}$$

To maintain the same level of efficiency as p increases, n must be increased such that:

$$T(1,n) \geq C\ T_0(p,n)$$

Isoefficiency relation – example

- Suppose that we have a parallel program where where $\boldsymbol{T_0(p, n) = n\ p}$ and $\boldsymbol{T(1, n) = 0.1\ n^2}$.
- Suppose the desired level of efficiency is $\boldsymbol{E = 0.9}$. Then:

$$
\begin{aligned}
T(1, n) \geq \frac{0.9}{0.1}\ T_0(p, n) &= 9\, T_0(p, n) \\
&\Longleftrightarrow 0.1\ n^2 \geq 9 n\ p \\
&\Longleftrightarrow n \geq 90\ p
\end{aligned}
$$

- Say that $\boldsymbol{p = 10}$. Then we should have $\boldsymbol{n \geq 900}$.
- Say $\boldsymbol{n = 2700}$. Then we should have $\boldsymbol{p \leq 30}$.

The scalability metric

- The isoefficiency relation is an expression of the form $n \geq f(p)$ It establishes conditions to maintain efficiency in **relation to execution time, but not memory requirements!**
- To quantify the scalability in memory terms, let $M(n)$ be the amount of memory required to solve a problem of size n.
- $M(n)$ **cannot grow arbitrarily**, i.e., beyond the amount of memory available per processor.
- We then must have $M(n) \geq M(f(p))$. To maintain the same level of efficiency the amount of required memory per processor is

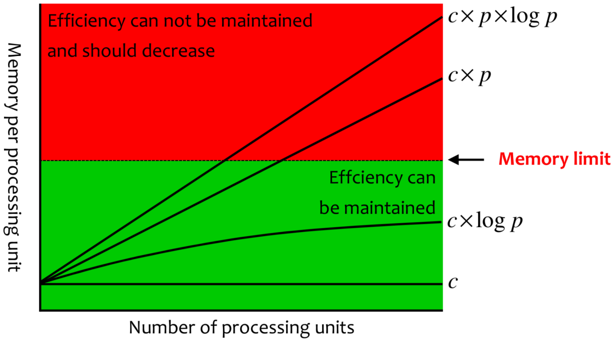
$$\frac{M(n)}{p} \geq \frac{M(f(p))}{p}$$

- The term

$$\frac{M(f(p))}{p}$$

is called the **scalability metric**.

The scalability metirc



- The lower the complexity of the scalability function, the more scalable is the parallel program. **Efficiency cannot be maintained and should decrease as $\frac{M(f(p))}{p}$ approximates or exceeds the available memory per processor.**

Scalability metric – example

- In our previous example the isoefficiency relation is expressed by $n \geq 90\, p$.
- If $M(n) = n^2$ then

$$\frac{M(f(p))}{p} = \frac{8100\, p^2}{p} = 8100p \text{ is } \Theta(p)$$

  an indication of low scalability.
- On the other hand if $M(n) = n \log n$

$$\frac{M(f(p))}{p} = \frac{90\, p \log(90\, p)}{p} = 90 \log(90\, p) \text{is } \Theta(\log p)$$

  has better scalability.