Parallel and Distributed Programming

Performance Metrics

Performance

- Two main goals to be achieved with the design of parallel applications are:
  - **Performance**: the capacity to reduce the time to solve the problem when the computing resources increase;
  - **Scalability**: the capacity to increase performance when the complexity, or size of the problem, increases.
- The main factors limiting the performance and the scalability of an application are:
  - **Architectural Limitations**
  - **Algorithmic Limitations**

Factors Limiting Performance

- Architectural Limitations
  - **Latency and Bandwidth**
  - **Data Coherency**
  - **Memory Capacity**

- Algorithmic Limitations
  - **Missing Parallelism** (sequential code)
  - **Communication Frequency**
  - **Synchronization Frequency**
  - **Poor Scheduling** (task granularity/load balancing)

Performance Metrics

- There are 2 distinct classes of performance metrics:
  - **Performance Metrics for Processors**: assess the performance of a processor using normally by measuring the speed or the number of operations that it does in a certain period of time.
  - **Performance Metrics of Parallel Applications**: assess the performance of a parallel application normally by comparing the execution time with multiple processors and the execution time with just one processor.
- We are mostly interested in metrics that allow the performance evaluation of parallel applications.

Performance Metrics for Processors

- Some of the best known metrics to measure performance of a processor architecture:
  - ***MIPS***: *Millions of Instructions Per Second*.
  - ***FLOPS***: *FLoating point Operations Per Second*.
  - ***SPECint***: SPEC (*Standard Performance Evaluation Corporation*) *benchmarks that evaluate processor performance on integer arithmetic* (1992).
  - ***SPECfp***: SPEC benchmarks that evaluate processor performance on floating point operations (2000).
  - ***Whetstone***: synthetic benchmarks to assess processor performance on floating point operations (1972).
  - ***Dhrystone***: synthetic benchmarks to asses processor performance on integer arithmetic (1984).

Performance Metrics for Parallel Applications

- There are a number of metrics, the best known are:
  - ***Speedup***
  - **Efficiency**
  - **Redundancy**
  - **Utilization**
  - **Quality**
- There also some laws/metrics that try to explain and assert the potential performance of a parallel application. The best known are:
  - **Amdahl Law**
  - **Gustafson-Barsis Law**
  - **Karp-Flatt Law**
  - **Isoeficiency Law**

Speedup

- *Speedup* is a measure of performance. It measures the ration between the sequential execution time and the parallel execution time.

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

T(1) is the execution time with one processor
T(p) is the execution time with *p* processors

	1 CPU	2 CPUs	4 CPUs	8 CPUs	16 CPUs
T(p)	1000	520	280	160	100
S(p)	1	1,92	3,57	6,25	10,00

Efficiency

- Efficiency is a measure of the usage of the computational resources. It measures the ration between performance and the resources used to achieve that performance.

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

S(p) is the speedup for *p* processors

	1 CPU	2 CPUs	4 CPUs	8 CPUs	16 CPUs
S(p)	1	1,92	3,57	6,25	10,00
E(p)	1	0,96	0,89	0,78	0,63

Redundancy

- Redundancy measures the increase in the required computation when using more processors. It measures the ration between the number of operations performed by the parallel execution and by the sequential execution.

$$R(p) = \frac{O(p)}{O(1)}$$

O(1) is the total number of operations performed with 1 processor

O(p) is the total number of operations performed with *p* processors

	1 CPU	2 CPUs	4 CPUs	8 CPUs	16 CPUs
O(p)	10000	10250	11000	12250	15000
R(p)	1	1,03	1,10	1,23	1,50

Utilization

- Utilization is a measure of the good use of the computational capacity. It measures the ratio between the computational capacity utilized during execution and the capacity that was available.

$$U(p) = R(p) \times E(p)$$

	1 CPU	2 CPUs	4 CPUs	8 CPUs	16 CPUs
R(p)	1	1,03	1,10	1,23	1,50
E(p)	1	0,96	0,89	0,78	0,63
U(p)	1	0,99	0,98	0,96	0,95

Quality

- Quality is a measure of the relevancy of using parallel computing.

$$Q(p) = \frac{S(p) \times E(p)}{R(p)}$$

	1 CPU	2 CPUs	4 CPUs	8 CPUs	16 CPUs
S(p)	1	1,92	3,57	6,25	10,00
E(p)	1	0,96	0,89	0,78	0,63
R(p)	1	1,03	1,10	1,23	1,50
Q(p)	1	1,79	2,89	3,96	4,20

Amdahl Law

- The computations performed by a parallel application are of 3 types:
  - *C(seq)*: computations that can only be realized sequencially.
  - *C(par)*: computations that can be realized in parallel.
  - *C(com)*: computations related to communication/synchronization/initialization.
- Using these 3 classes, the *speedup* of an application can be defined as:

$$S(p) = \frac{T(1)}{T(p)} = \frac{C(seq) + C(par)}{C(seq) + \frac{C(par)}{p} + C(com)}$$

Amdahl Law

- Since $C(com) \geq 0$ then:

$$S(p) \leq \frac{C(seq) + C(par)}{C(seq) + \frac{C(par)}{p}}$$

- If f is the fraction of the computation that can only be realized sequentially, then:

$$f = \frac{C(seq)}{C(seq) + C(par)} \quad \text{and} \quad S(p) \leq \frac{\frac{C(seq)}{f}}{C(seq) + \frac{C(seq) \times \left(\frac{1}{f} - 1\right)}{p}}$$

Amdahl Law

- Simplifying:

$$S(p) \leq \frac{\frac{C(seq)}{f}}{C(seq) + \frac{C(seq) \times \left(\frac{1}{f} - 1\right)}{p}}$$

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

Amdahl Law

- Let $0 \le f \le 1$ be the computation fraction that can only be realized sequentially. The Amdahl law tells us that the maximum *speedup* that a parallel application can attain with p processors is:

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

- The Amdahl law can also be used to determine the limit of maximum *speedup* that a determined application can achieve regardless of the number of processors uused.

Parallel and Distributed Programming | Performance Metrics

Amdahl Law

- Suppose one wants to determine if it is advantageos to develop a parallel version of a certain sequential application. Through experimentation, it was verified that 90% of the execution time is spent in procedures that may be parallelizable. What is the maximum *speedup* that can be achieved with a parallel version of the problem executing on 8 processors?

$$S(p) \le \frac{1}{0{,}1 + \frac{1-0{,}1}{8}} \approx 4{,}71$$

- And the limit of the maximum *speedup* that can be attained?

$$\lim_{p \to \infty} \frac{1}{0{,}1 + \frac{1-0{,}1}{p}} = 10$$

Limitations of the Amdahl Law

- The Amdahl law ignores the cost with communication/synchronization operations associated to the introduction of parallelism in an application. For this reason, the Amdahl law can result in predictions not very realistic for certain problems.
- Consider a parallel application, with complexity O(n^2), whose execution pattern is the following, where *n* is the size of the problem:
  - Execution time of the sequential part (*input* and *output* of data): $18.000 + n$
  - Execution time of the parallel parte: $\frac{n^2}{100}$
  - Total communication/synchronization points per processor: $\lceil \log n \rceil$
  - Execution time due to communication/synchronization (n=10.000):

$$10.000 \times \lceil \log p \rceil + \frac{n}{10}$$

Limitations of the Amdahl Law

- What is the maximum *speedup* attainable?
  - Uzing Amdahl law:

$$f = \frac{18.000 + n}{18.000 + n + \frac{n^2}{100}} \quad \text{and} \quad S(p) \leq \frac{18.000 + n + \frac{n^2}{100}}{18.000 + n + \frac{n^2}{p \times 100}}$$

  - Uzinf the *speedup* measure:

$$S(p) = \frac{18.000 + n + \frac{n^2}{100}}{18.000 + n + \frac{n^2}{p \times 100} + \lceil \log n \rceil \times \left(10.000 \times \lceil \log p \rceil + \frac{n}{10} \right)}$$

Limitations of the Amdahl Law

		1 CPU	2 CPUs	4 CPUs	8 CPUs	16 CPUs
Amdahl law	n = 10.000	1	1,95	3,70	6,72	11,36
	n = 20.000	1	1,98	3,89	7,51	14,02
	n = 30.000	1	1,99	3,94	7,71	14,82
Speedup	n = 10.000	1	1,61	2,11	2,22	2,57
	n = 20.000	1	1,87	3,21	4,71	6,64
	n = 30.000	1	1,93	3,55	5,89	9,29

Gustafson-Barsis Law

- Consider again the *speedup* measure defined previously:

$$S(p) \leq \frac{C(seq) + C(par)}{C(seq) + \frac{C(par)}{p}}$$

- If f is the fraction of the parallel computation spent executing sequential computations, then (1-f) is the fraction of the time spent in the parallel part:

$$f = \frac{C(seq)}{C(seq) + \frac{C(par)}{p}} \quad \text{and} \quad (1-f) = \frac{\frac{C(par)}{p}}{C(seq) + \frac{C(par)}{p}}$$

Gustafson-Barsis Law

- Then:

$$C(seq) = f \times \left(C(seq) + \frac{C(par)}{p} \right)$$

$$C(par) = p \times (1 - f) \times \left(C(seq) + \frac{C(par)}{p} \right)$$

- Simplifying:

$$S(p) \leq \frac{\left(f + p \times (1 - f)\right) \times \left(C(seq) + \frac{C(par)}{p} \right)}{C(seq) + \frac{C(par)}{p}}$$

$$\Rightarrow \quad S(p) \leq f + p \times (1 - f) \quad \Rightarrow \quad S(p) \leq p + f \times (1 - p)$$

Gustafson-Barsis Law

- Let $0 \leq f \leq 1$ be the fraction of parallel computation spent executing sequential computations. The Gustafson-Barsis law tells us that the maximum *speedup* that a parallel application with p processors can attain is:

$$S(p) \leq p + f \times (1 - p)$$

- While the Amdahl law starts from the time of the sequential execution to estimate the maximum *speedup* that can be attained with multiple processors, the Gustafson-Barsis law does the opposite, that is, it starts from the parallel execution time to estimate the maximum *speedup* in comparison with the sequential execution.

Gustafson-Barsis Law

- Consider that a certain application executes in 220 seconds in 64 processors. What is the maximum *speedup* of an application knowing, by experimentation, that 5% of the execution time is spent on sequential computations.

$$S(p) \leq 64 + (0{,}05) \times (1 - 64) = 64 - 3{,}15 = 60{,}85$$

- Suppose that a certain company wants to buy a supercomputer with 16.384 processors to achieve a *speedup* of 15.000 in an important fundamental problem. What is the maximum fraction of the parallel execution that can be spent in sequential computations to attain the expected *speedup*?

$$15.000 \leq 16.384 + f \times (1 - 16.384)$$

$$f \times 16.383 \leq 1.384$$

$$f \leq 0{,}084$$

Gustafson-Barsis Law Limitations

- When using the execution time of the parallel execution as a starting point, instead of the sequential execution, the Gustafson-Barsis law assumes that the execution with one processor is, in the worst cases, p times slower than the execution with p processors.
- This may not be true if the available memory for the execution with one processor is insufficient when compared to the the computation with p processors. For this reason, the estimated *speedup* by the Gustafson-Barsis law is normally designated as ***scaled speedup***.

Karp-Flatt Metric

- Let us consider again the definition of sequential execution time and parallel execution time:

$$T(1) = C(seq) + C(par)$$

$$T(p) = C(seq) + \frac{C(par)}{p} + C(com)$$

- Let e be the **experimentally determined sequential fraction** of a parallel computation:

$$e = \frac{C(seq)}{T(1)}$$

Karp-Flatt Metric

- Then:

$$C(seq) = e \times T(1)$$

$$C(par) = (1-e) \times T(1)$$

- If one considers that *C(com)* is negligible then:

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

- On the other hand:

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

$$\Rightarrow \quad T(1) = S(p) \times T(p)$$

Karp-Flatt Metric

- Simplifying:

$$T(p) = e \times S(p) \times T(p) + \frac{(1-e) \times S(p) \times T(p)}{p}$$

$$\Rightarrow \quad 1 = e \times S(p) + \frac{(1-e) \times S(p)}{p}$$

$$\Rightarrow \quad \frac{1}{S(p)} = e + \frac{(1-e)}{p} \qquad \Rightarrow \qquad \frac{1}{S(p)} = e + \frac{1}{p} - \frac{e}{p}$$

$$\Rightarrow \quad \frac{1}{S(p)} = e \times \left(1 - \frac{1}{p}\right) + \frac{1}{p} \qquad \Rightarrow \qquad e = \frac{\frac{1}{S(p)} - \frac{1}{p}}{1 - \frac{1}{p}}$$

Karp-Flatt Metric

- Let *S(p)* be the *speedup* of a parallel application with $p > 1$ processors. The Karp-Flatt metric tells us that the experimentally determined sequential fraction is:

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

- The less the value *e* the better the parallelization
- The Karp-Flatt metric is interesting because by negleting the costs with communication/synchronization/initialization operations associated with parallelism, allows us, a posteriori, to determine the relevance of the *C(com)* component in the eventual decrease of the application's efficiency.

Karp-Flatt Metric

- By definition, the experimentally determined sequential fraction is a constant value that does not depend on the number of processors.

$$e = \frac{C(seq)}{T(1)}$$

- On the other hand, the Karp-Flatt metric is a function of the number of processors.

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

Karp-Flatt Metric

- Considering that the efficiency of an application is a decreasing function on the number of processors, Karp-Flatt metric allows us to determine the importance of *C(com)* in that decrease.
  - If the values of *e* are constant when the number of processors increases, that means that the *C(com)* component is constant. Therefore, the efficiency decrease is due to the scarse parallelism available in the application.
  - If the values of *e* increase with the increase in the number of processors, it means that the decrease is due to the *C(com)* component, that is, due to the excessive costs associated with the parallel computation (communication costs, synchronization and/or computation initialization).

Karp-Flatt Metric

- For example, the Karp-Flatt metric allows us to detect sources of inefficiency not considered by the model, which assumes that *p* processors execute the parallel part *p* times faster then when executing with just one processor.
  - If we have 5 processors to solve a problem decomposed in 20 atomic tasks, then all processors can execute 4 tasks. If all tasks take the same time to execute, then the parallel execution time should be a fraction of 5.
  - On the other hand, if we have 6 processors to solve the same problem, 4 processors can execute 3 tasks but the other 2 must necessarily execute 4. This makes the execution time again a fraction of 5 and not of 6.

Karp-Flatt Metric

- Consider the following *speedups* obtained by a certain parallel application:

	2 CPUs	3 CPUs	4 CPUs	5 CPUs	6 CPUs	7 CPUs	8 CPUs
S(p)	1,82	2,50	3,08	3,57	4,00	4,38	4,71
e	0,099	0,100	0,100	0,100	0,100	0,100	0,100

- What is the main reason for the application to just achieve a *speedup* of 4,71 with 8 processors?
  - Given that *e* doesn't increase with the number of processors, it means that the main reason for the small *speedup* is the little parallelism avaiable in the problem.

Karp-Flatt Metric

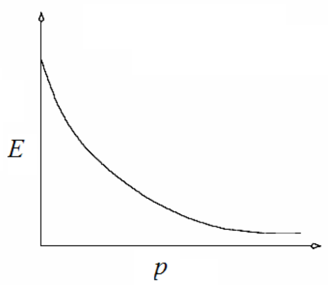
- Consider the following *speedups* obtained by a certain parallel application:

	2 CPUs	3 CPUs	4 CPUs	5 CPUs	6 CPUs	7 CPUs	8 CPUs
S(p)	1,87	2,61	3,23	3,73	4,14	4,46	4,71
e	0,070	0,075	0,079	0,085	0,090	0,095	0,100

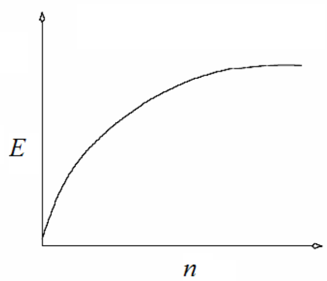
- What is the main reason for the application to just achieve a *speedup* of 4,71 with 8 processors?
  - Given that *e* increases slightly with the number of processors, it means that the main reason for the small *speedup* are the costs associated to the parallel computation.

Efficiency and Scalability

- From previous results, we can conclude that the efficiency of an application is:
  - A decreasing function of the number of processors.
  - Typically, an increasing function on the size of the probem.



tamanho do problema fixo (n)



número de processadores fixo (p)

Efficiency and Scalability

- An application is said **scalable** when its efficiency is maintained when we increase proportionally the number of processors and the size of the problem.
- The scalability of an application reflects its capacity in making use of available resources effectively.

		1 CPU	2 CPUs	4 CPUs	8 CPUs	16 CPUs
Efficiency	n = 10.000	1	0,81	0,53	0,28	0,16
	n = 20.000	1	0,94	0,80	0,59	0,42
	n = 30.000	1	0,96	0,89	0,74	0,58

Isoefficiency Metric

- The efficiency of an application is tipically an increasing function of the size of the problem since the complexity of communication is, normally, smaller then the computation complexity, that is, **to maintain the same level of efficiency when we increase the number of processors one needs to increase the size of the problem**. The isoefficiency metric formalizes this idea.
- Lets consider again the definition of *speedup*:

$$S(p) = \frac{C(seq) + C(par)}{C(seq) + \frac{C(par)}{p} + C(com)} = \frac{p \times (C(seq) + C(par))}{p \times C(seq) + C(par) + p \times C(com)}$$

$$= \frac{p \times (C(seq) + C(par))}{C(seq) + C(par) + (p-1) \times C(seq) + p \times C(com)}$$

Isoefficiency Metric

- Let $T_0(p)$ be the execution time spent by p processors on the parallel algorithm performing computations not done in sequential algorithm:

$$T_0(p) = (p-1) \times C(seq) + p \times C(com)$$

- Simplifying:

$$S(p) = \frac{p \times \left(C(seq) + C(par)\right)}{C(seq) + C(par) + T_0(p)}$$

$$E(p) = \frac{C(seq) + C(par)}{C(seq) + C(par) + T_0(p)} = \frac{1}{1 + \frac{T_0(p)}{C(seq) + C(par)}} = \frac{1}{1 + \frac{T_0(p)}{T(1)}}$$

Isoefficiency Metric

- Then:

$$E(p) = \frac{1}{1 + \frac{T_0(p)}{T(1)}}$$

$$\Rightarrow \quad \frac{T_0(p)}{T(1)} = \frac{1 - E(p)}{E(p)} \quad \Rightarrow \quad T(1) = \frac{E(p)}{1 - E(p)} \times T_0(p)$$

- If one wants to maintain the same level of efficiency when we increase the number of processors, then:

$$\frac{E(p)}{1 - E(p)} = c \qquad \text{e} \qquad T(1) \geq c \times T_0(p)$$

Isoefficiency Metric

- Let $E(p)$ be the efficiency of a parallel application with p processors. The isoefficiency metric tells us that to maintain the same level of efficiency when we increase the number of processors, then the size of the problem must be increased so that the following inequality is satisfied:

$$T(1) \geq c \times T_0(p)$$

$$with \quad c = \frac{E(p)}{1 - E(p)} \quad \text{and} \quad T_0(p) = (p-1) \times C(seq) + p \times C(com)$$

- The applicability of the isoefficiency metric may depend on the available memory, considering the maximum size of the problem that can be solved is limited by that quantity.

Isoefficiency Metric

- Suppose that the isoefficiency metric for a problem size n is given as a function on the number of processors p:

$$n \geq f(p)$$

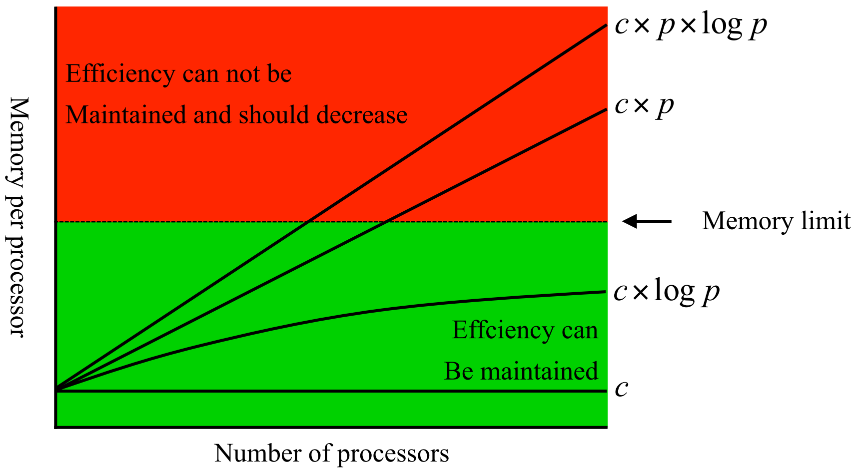
- If $M(n)$ designates the quantity of required memory to solve a problem of size n then:

$$M(n) \geq M\big(f(p)\big)$$

- That is, to maintain the same level of efficiency, the quantity of required memory per processor is:

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

Isoefficiency Metric



Isoefficiency Metric

- Consider that the sequential version of a certain application has complexity $O(n^3)$, and that the execution time spent by each of the p processors of the parallel version in communication/synchronization operations is $O(n^2 \log p)$. If the amount of memory necessary to represent a problem of size n is n^2, what is the scalability of the application in terms of memory?

$$n^3 \geq c \times p \times n^2 \times \log p$$

$$n \geq c \times p \times \log p$$

$$M(n) = n^2 \quad \Rightarrow \quad \frac{M(c \times p \times \log p)}{p} = \frac{c^2 \times p^2 \times \log^2 p}{p} = c^2 \times p \times \log^2 p$$

- Then, the scalability of the application is low.

Superlinear *Speedup*

- The *speedup* is said to be superlinear when the ratio between the sequential execution time and the parallel execution time with p processors is greater than p.

$$\frac{T(1)}{T(p)} \geq p$$

- Some factors that may make the speedup superlinear are:
  - Comunication/synchronization/initialization costs are almost inexistent.
  - Tolerancy to communication latency.
  - Increase the memory capacity (the problem may have to fit all in memory).
  - Subdivisions of the problema (smaller tasks may generate less *cache misses)*.
  - Computation randomness in optimization problems or with multiple solutions.

Superlinear *Speedup*

"If just one computer (processador) can solve a problem in N seconds, could N computers (processors) Solve the same problem in 1 second?"