Parallel Performance Analysis

Parallel and Distributed Computing

Department of Computer Science and Engineering (DEI)
Instituto Superior Técnico

November 8, 2012

Outline

- Performance Analysis
- Speedup and Efficiency
- Amdahl's Law
- Gustafson-Barsis' Law
- Karp-Flatt Metric
- Isoefficiency Metric

Performance Analysis

Objectives:

- predict performance of parallel programs
- understand barriers to higher performance

Speedup

Speedup

Measure of how much faster is the execution of a parallel program versus a sequential one.

$$Speedup = \frac{\text{Sequential execution time}}{\text{Parallel execution time}}$$

Speedup: $\psi(n, p)$

n: problem size

p: number of tasks

Execution Time Components

$\sigma(n)$: inherently sequential computations

Execution Time Components

$\sigma(n)$: inherently sequential computations

$\varphi(n)$: completely parallelizable computations

Execution Time Components

$\sigma(n)$: inherently sequential computations

$\varphi(n)$: completely parallelizable computations

$\kappa(n, p)$: communication / synchronization / redundant operations

Execution Time Components

$\sigma(n)$: inherently sequential computations

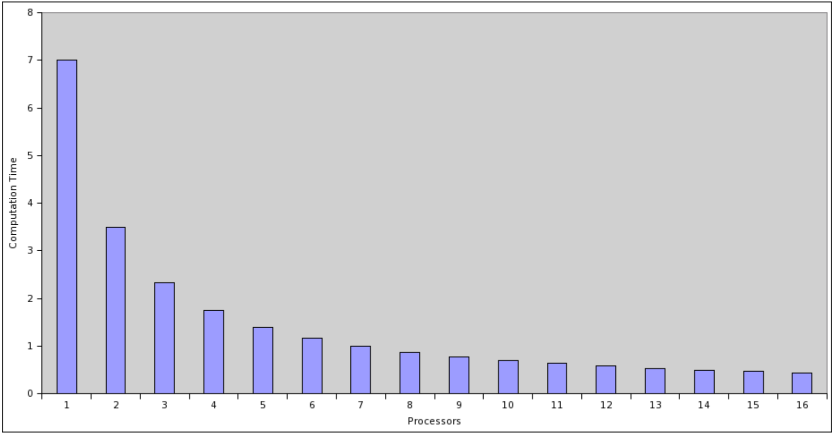
$\varphi(n)$: completely parallelizable computations

$\kappa(n,p)$: communication / synchronization / redundant operations

$$\psi(n,p) \leq \frac{\sigma(n) + \varphi(n)}{\sigma(n) + \varphi(n)/p + \kappa(n,p)}$$

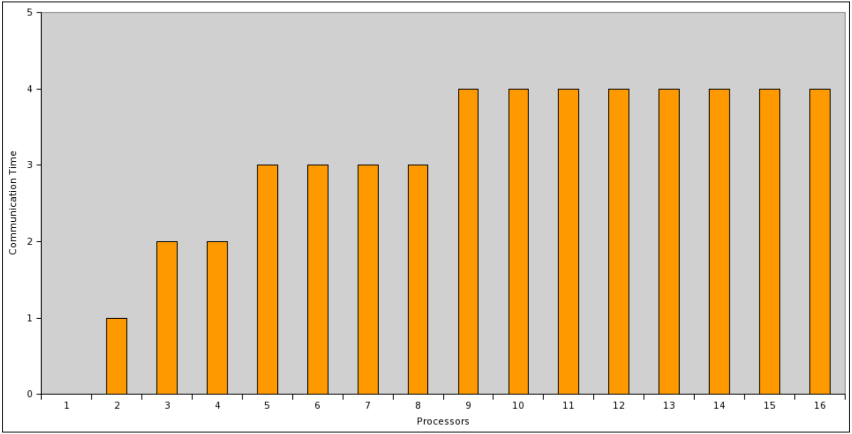
Computation Time

$\varphi(n)/p$



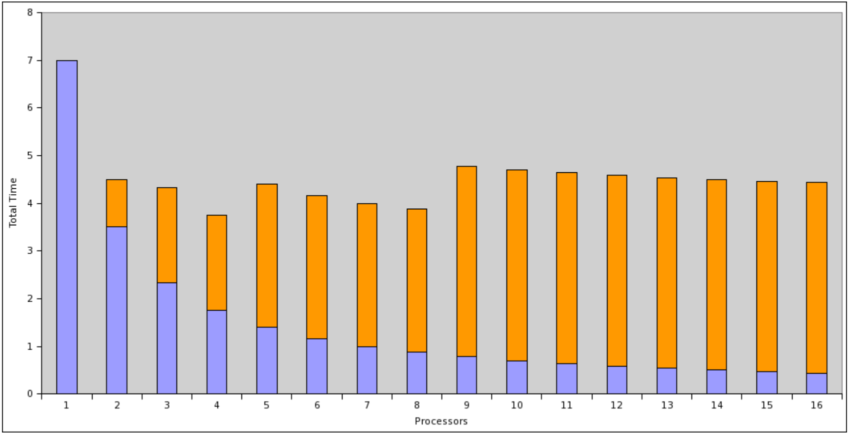
Communication Time

$\kappa(n, p)$

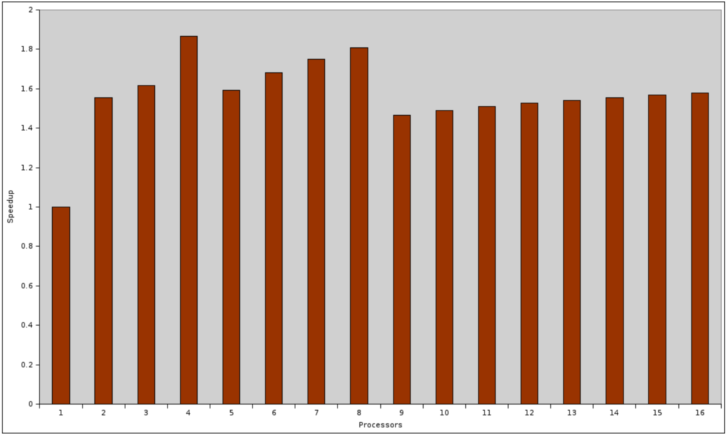


Total Time

$\varphi(n)/p + \kappa(n,p)$



Speedup


Speedup
Processors
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16

Efficiency

Efficiency

Measure of utilization of available processors.

$$\textit{Efficiency} = \frac{\text{Sequential time}}{\text{Processors used} \times \text{Parallel time}} = \frac{\text{Speedup}}{\text{Processors used}}$$

Efficiency: $\varepsilon(n, p)$

$$\varepsilon(n, p) \leq \frac{\sigma(n) + \varphi(n)}{p\sigma(n) + \varphi(n) + p\kappa(n, p)}$$

$$0 \leq \varepsilon(n, p) \leq 1$$

Amdahl's Law

Speedup:

$$\psi(n,p) \leq \frac{\sigma(n)+\varphi(n)}{\sigma(n)+\varphi(n)/p+\kappa(n,p)} \leq \frac{\sigma(n)+\varphi(n)}{\sigma(n)+\varphi(n)/p}$$

Amdahl's Law

Speedup:

$$\psi(n,p) \le \frac{\sigma(n)+\varphi(n)}{\sigma(n)+\varphi(n)/p+\kappa(n,p)} \le \frac{\sigma(n)+\varphi(n)}{\sigma(n)+\varphi(n)/p}$$

Let f be the fraction of sequential computation in the original sequential program:

$$f(n) = \frac{\sigma(n)}{\sigma(n)+\varphi(n)}$$

Amdahl's Law

$$\psi(n,p) \le \frac{1}{f(n)+\frac{1-f(n)}{p}}$$

Problem

A computer animation program generates a feature movie frame-by-frame. Each frame can be generated independently and is output to its own file. If it takes 99 seconds to render a frame and 1 second to output it, how much speedup can be achieved by rendering the movie on 100 processors?

Problem

A computer animation program generates a feature movie frame-by-frame. Each frame can be generated independently and is output to its own file. If it takes 99 seconds to render a frame and 1 second to output it, how much speedup can be achieved by rendering the movie on 100 processors?

$f(n) = 0,01 \quad p = 100$

$$\psi(n,p) \leq \frac{1}{0,01 + \frac{0{,}99}{100}} = 50,3$$

Amdahl's Law

Limitations of Amdahl's Law:

Amdahl's Law

Limitations of Amdahl's Law:

- only considers 2 execution modes

Amdahl's Law

Limitations of Amdahl's Law:

- only considers 2 execution modes
- does not take into account parallel overhead, $\kappa(n, p)$

$\Rightarrow$ Overestimates achievable speedup

Amdahl Effect

- typically, $\kappa(n, p)$ has lower complexity than $\varphi(n)/p$
- as n increases, $\varphi(n)/p$ dominates $\kappa(n, p)$
- as n increases, speedup increases

Alternative Optimization Measure

Amdahl's Law:

- treats problem size as a constant
- shows how execution time decreases as number of processors increases

Alternative Optimization Measure

Amdahl's Law:

- treats problem size as a constant
- shows how execution time decreases as number of processors increases

A different perspective:

- faster computers solve larger problem instances
- consider time as a constant and allow problem size to increase with number of processors

Gustafson-Barsis' Law

Speedup:

$$\psi(n,p) \leq \frac{\sigma(n)+\varphi(n)}{\sigma(n)+\varphi(n)/p+\kappa(n,p)} \leq \frac{\sigma(n)+\varphi(n)}{\sigma(n)+\varphi(n)/p}$$

Gustafson-Barsis' Law

Speedup:

$$\psi(n,p) \le \frac{\sigma(n)+\varphi(n)}{\sigma(n)+\varphi(n)/p+\kappa(n,p)} \le \frac{\sigma(n)+\varphi(n)}{\sigma(n)+\varphi(n)/p}$$

Let s be the fraction of sequential computation in the parallel program:

$$s = \frac{\sigma(n)}{\sigma(n)+\frac{\varphi(n)}{p}}$$

Gustafson-Barsis' Law

$$\psi(n,p) \le p + (1-p)s$$

Gustafson-Barsis' Law

- starts from parallel execution time
- estimates sequential execution time to solve the same problem
- problem size is an increasing function of p
- predicts scaled speedup

Example

An application running on 10 processors spends 3% of its time in serial code. What is the scaled speedup of the application?

Example

An application running on 10 processors spends 3% of its time in serial code. What is the scaled speedup of the application?

Scaled Speedup:

$$p + (1 - p)s = 9,7$$

The Karp-Flatt Metric

- Amdahl's Law and Gustafson-Barsis Law ignore $\kappa(n, p)$
- overestimate speedup or scaled speedup
- Karp and Flatt proposed another metric
  $\Rightarrow$ experimentally determined serial fraction

The Karp-Flatt Metric

- Amdahl's Law and Gustafson-Barsis Law ignore $\kappa(n, p)$
- overestimate speedup or scaled speedup
- Karp and Flatt proposed another metric
  $\Rightarrow$ experimentally determined serial fraction

Experimentally determined serial fraction

Represents the fraction of the original program that cannot be parallelized with respect to the sequential execution time.

$$e = \frac{\sigma(n) + \kappa(n, p)}{\sigma(n) + \varphi(n)}$$

The Karp-Flatt Metric

$$e = \frac{\sigma(n) + \kappa(n,p)}{\sigma(n) + \varphi(n)}$$

Execution time of a parallel program in p processors:

$$T(n,p) = \sigma(n) + \varphi(n)/p + \kappa(n,p)$$

$$T(n,1) = \sigma(n) + \varphi(n) \qquad\qquad e = \frac{\sigma(n) + \kappa(n,p)}{T(n,1)}$$

$$e = \frac{1/\psi(n,p) - 1/p}{1 - 1/p}$$

The Karp-Flatt Metric

- takes into account parallel overhead
- allow for the analysis of the source of parallel inefficiency
  - limited opportunity for computational parallelism
  - parallel overhead (communication, synchronization, load balancing, etc)

Example 1

p	2	3	4	5	6	7	8
ψ	1,8	2,5	3,1	3,6	4,0	4,4	4,7

What is the primary reason for speedup of only 4,7 on 8 CPUs?

Example 1

p	2	3	4	5	6	7	8
ψ	1,8	2,5	3,1	3,6	4,0	4,4	4,7

What is the primary reason for speedup of only 4,7 on 8 CPUs?

p	2	3	4	5	6	7	8
e	0,1	0,1	0,1	0,1	0,1	0,1	0,1

Since e is constant, large serial fraction is the primary reason.

Example 2

p	2	3	4	5	6	7	8
ψ	1,9	2,6	3,2	3,7	4,1	4,5	4,7

What is the primary reason for speedup of only 4,7 on 8 CPUs?

Example 2

p	2	3	4	5	6	7	8
ψ	1,9	2,6	3,2	3,7	4,1	4,5	4,7

What is the primary reason for speedup of only 4,7 on 8 CPUs?

p	2	3	4	5	6	7	8
e	0,070	0,075	0,080	0,085	0,090	0,095	0,100

Since e is steadily increasing, overhead is the primary reason.

Problem

p	4	8	12
ψ	3,9	6,5	?

Is this program likely to achieve a speedup of 10 on 12 processors?

Problem

p	4	8	12
ψ	3,9	6,5	?

Is this program likely to achieve a speedup of 10 on 12 processors?

p	4	8	12
e	0,009	0,033	0,018

Problem

p	4	8	12
ψ	3,9	6,5	?

Is this program likely to achieve a speedup of 10 on 12 processors?

p	4	8	12
e	0,009	0,033	0,018

e typically increases with p. Speedup probably closer to 8 on 12 processors.

Isoefficiency Metric

Scalability

Scalability of a parallel system measures the ability to increase performance as number of processors increases.
(parallel system: parallel program executing on a parallel computer)

A scalable system maintains efficiency as processors are added.

Isoefficiency Metric

Scalability

Scalability of a parallel system measures the ability to increase performance as number of processors increases.
(parallel system: parallel program executing on a parallel computer)

A scalable system maintains efficiency as processors are added.

Isoefficiency is a way to measure scalability.

Isoefficiency Metric

Execution time of parallel program in p processors:

$$T(n,p) = \sigma(n) + \varphi(n)/p + \kappa(n,p)$$

Let $T_0(n,p)$ be the time spent doing work not done by the sequential algorithm:

$$T_0(n,p) = (p-1)\sigma(n) + p\kappa(n,p)$$

Isoefficiency Metric

Execution time of parallel program in p processors:

$$T(n,p) = \sigma(n) + \varphi(n)/p + \kappa(n,p)$$

Let $T_0(n,p)$ be the time spent doing work not done by the sequential algorithm:

$$T_0(n,p) = (p-1)\sigma(n) + p\kappa(n,p)$$

$$\psi(n,p) \quad \leq \quad \frac{\sigma(n) + \varphi(n)}{\sigma(n) + \varphi(n)/p + \kappa(n,p)}$$

Isoefficiency Metric

Execution time of parallel program in p processors:

$$T(n,p) = \sigma(n) + \varphi(n)/p + \kappa(n,p)$$

Let $T_0(n,p)$ be the time spent doing work not done by the sequential algorithm:

$$T_0(n,p) = (p-1)\sigma(n) + p\kappa(n,p)$$

$$\begin{aligned}\psi(n,p) &\leq \frac{\sigma(n)+\varphi(n)}{\sigma(n)+\varphi(n)/p+\kappa(n,p)} \\ &= \frac{p(\sigma(n)+\varphi(n))}{\sigma(n)+\varphi(n)+(p-1)\sigma(n)+p\kappa(n,p)}\end{aligned}$$

Isoefficiency Metric

Execution time of parallel program in p processors:

$$T(n,p) = \sigma(n) + \varphi(n)/p + \kappa(n,p)$$

Let $T_0(n,p)$ be the time spent doing work not done by the sequential algorithm:

$$T_0(n,p) = (p-1)\sigma(n) + p\kappa(n,p)$$

$$\begin{aligned}
\psi(n,p) &\leq \frac{\sigma(n)+\varphi(n)}{\sigma(n)+\varphi(n)/p+\kappa(n,p)} \\
&= \frac{p(\sigma(n)+\varphi(n))}{\sigma(n)+\varphi(n)+(p-1)\sigma(n)+p\kappa(n,p)} \\
&= \frac{p(\sigma(n)+\varphi(n))}{\sigma(n)+\varphi(n)+T_0(n,p)}
\end{aligned}$$

Isoefficiency Metric

$$\varepsilon(n, p) = \frac{\psi(n, p)}{p}$$

Isoefficiency Metric

$$
\begin{aligned}
\varepsilon(n,p) &= \frac{\psi(n,p)}{p} \\
&\leq \frac{1}{1+\frac{T_0(n,p)}{\sigma(n)+\varphi(n)}}
\end{aligned}
$$

Isoefficiency Metric

$$\begin{aligned}
\varepsilon(n,p) &= \frac{\psi(n,p)}{p} \\
&\leq \frac{1}{1 + \frac{T_0(n,p)}{\sigma(n)+\varphi(n)}} \\
&= \frac{1}{1 + T_0(n,p)/T(n,1)}
\end{aligned}$$

Isoefficiency Metric

$$\begin{aligned}
\varepsilon(n,p) &= \frac{\psi(n,p)}{p} \\
&\leq \frac{1}{1 + \frac{T_0(n,p)}{\sigma(n)+\varphi(n)}} \\
&= \frac{1}{1 + T_0(n,p)/T(n,1)} \\
\Rightarrow T(n,1) &\geq \frac{\varepsilon(n,p)}{1-\varepsilon(n,p)} T_0(n,p)
\end{aligned}$$

Isoefficiency Metric

$$
\begin{aligned}
\varepsilon(n,p) &= \frac{\psi(n,p)}{p} \\
&\leq \frac{1}{1+\frac{T_0(n,p)}{\sigma(n)+\varphi(n)}} \\
&= \frac{1}{1+T_0(n,p)/T(n,1)} \\
\Rightarrow T(n,1) &\geq \frac{\varepsilon(n,p)}{1-\varepsilon(n,p)} T_0(n,p)
\end{aligned}
$$

In order to maintain efficiency: constant $\frac{\varepsilon(n,p)}{1-\varepsilon(n,p)} = C$

Isoefficiency Metric

$$\begin{aligned}
\varepsilon(n,p) &= \frac{\psi(n,p)}{p} \\
&\leq \frac{1}{1+\frac{T_0(n,p)}{\sigma(n)+\varphi(n)}} \\
&= \frac{1}{1+T_0(n,p)/T(n,1)} \\
\Rightarrow T(n,1) &\geq \frac{\varepsilon(n,p)}{1-\varepsilon(n,p)} T_0(n,p)
\end{aligned}$$

In order to maintain efficiency: constant $\frac{\varepsilon(n,p)}{1-\varepsilon(n,p)} = C$

Isoefficiency Relation

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

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

Scalability Function

Suppose isoefficiency relation is $n \geq f(p)$.

Scalability Function

Suppose isoefficiency relation is $n \geq f(p)$.

Let $M(n)$ denote memory required for problem of size n.

Scalability Function

Suppose isoefficiency relation is $n \geq f(p)$.

Let $M(n)$ denote memory required for problem of size n.

$M(f(p))/p$ indicates how memory usage per processor must increase to maintain same efficiency.

Scalability Function

Suppose isoefficiency relation is $n \geq f(p)$.

Let $M(n)$ denote memory required for problem of size n.

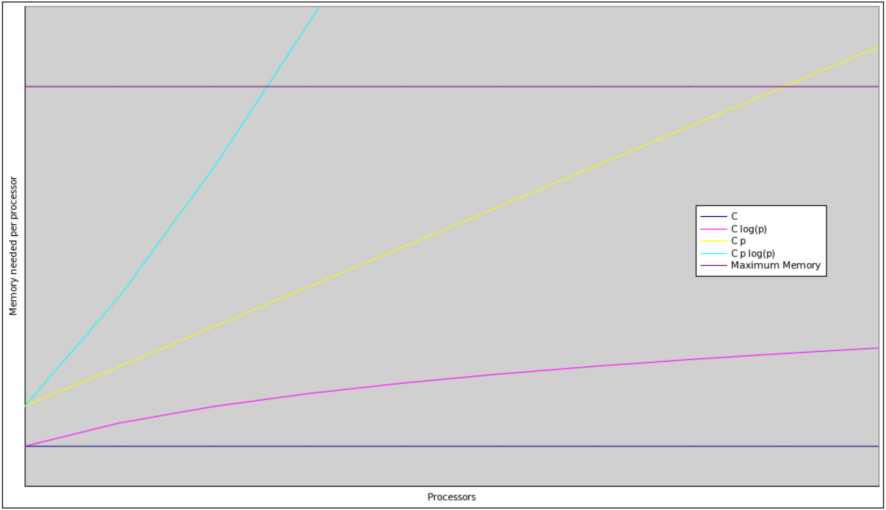
$M(f(p))/p$ indicates how memory usage per processor must increase to maintain same efficiency.

$M(f(p))/p$ is called the scalability function.

Meaning of Scalability Function

- to maintain efficiency when increasing p, we must increase n
- maximum problem size limited by available memory, which is linear in p
- scalability function shows how memory usage per processor must grow to maintain efficiency
- scalability function a constant means parallel system is perfectly scalable

Interpreting the Scalability Function



Example 1: Reduction

Sequential algorithm complexity:

Example 1: Reduction

Sequential algorithm complexity: $T(n, 1) = \Theta(n)$

Parallel algorithm:

Computational complexity:

Example 1: Reduction

Sequential algorithm complexity: $T(n, 1) = \Theta(n)$

Parallel algorithm:

Computational complexity: $\Theta(n/p)$

Communication complexity:

Example 1: Reduction

Sequential algorithm complexity: $T(n, 1) = \Theta(n)$

Parallel algorithm:

Computational complexity: $\Theta(n/p)$
Communication complexity: $\Theta(\log p)$

Parallel overhead: $T_0(n, p) = \Theta(p \log p)$

Example 1: Reduction

Isoefficiency relation: $n \geq Cp \log p$

To maintain same level of efficiency, how must n increase when p increases?

Example 1: Reduction

Isoefficiency relation: $n \geq Cp \log p$

To maintain same level of efficiency, how must n increase when p increases?

$M(n) = n$

Scalability function:

$$M(Cp \log p)/p = C \log p$$

The system has good scalability!

Example 2: Floyd-Warshall Algorithm

Sequential algorithm complexity:

Example 2: Floyd-Warshall Algorithm

Sequential algorithm complexity: $T(n, 1) = \Theta(n^3)$

Parallel algorithm:

Computational complexity:

Example 2: Floyd-Warshall Algorithm

Sequential algorithm complexity: $T(n, 1) = \Theta(n^3)$

Parallel algorithm:

Computational complexity: $\Theta(n^3/p)$

Communication complexity:

Example 2: Floyd-Warshall Algorithm

Sequential algorithm complexity: $T(n, 1) = \Theta(n^3)$

Parallel algorithm:

Computational complexity: $\Theta(n^3/p)$

Communication complexity: $\Theta(n^2 \log p)$

Parallel overhead: $T_0(n, p) = \Theta(pn^2 \log p)$

Example 2: Floyd-Warshall Algorithm

Isoefficiency relation: $n^3 \geq Cpn^2 \log p \Rightarrow n \geq Cp \log p$

To maintain same level of efficiency, how must n increase when p increases?

Example 2: Floyd-Warshall Algorithm

Isoefficiency relation: $n^3 \geq Cpn^2 \log p \Rightarrow n \geq Cp \log p$

To maintain same level of efficiency, how must n increase when p increases?

$M(n) = n^2$

Scalability function:

$$M(Cp \log p)/p = C^2 p \log^2 p$$

The system has poor scalability!

Review

- Performance Analysis
- Speedup and Efficiency
- Amdahl's Law
- Gustafson-Barsis' Law
- Karp-Flatt Metric
- Isoefficiency Metric

Next Class

- Matrix-vector multiplication