Parallel Programming

Parallel performance analysis

Objectives and contents

- Understand barriers to higher performance
- General speedup formula
- Amdahl's Law
- Gustafson-Barsis' Law
- Karp-Flatt metric
- Isoefficiency metric

Speedup

- **Definition**

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

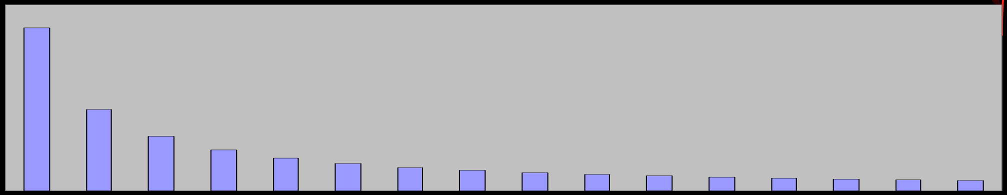
- **Inherently sequential computations: $\sigma(n)$**
- **Potentially parallel computations: $\varphi(n)$**
- **Communication operations: $\kappa(n, p)$**

- **Speedup - $\Psi(n, p)$**

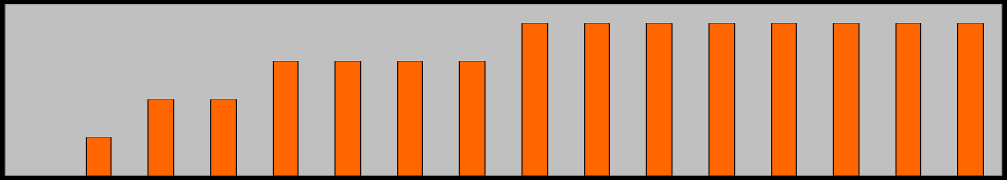
$$\psi(n,p) \le \frac{\sigma(n)+\varphi(n)}{\sigma(n)+\varphi(n)/p+\kappa(n,p)}$$

The communications effect

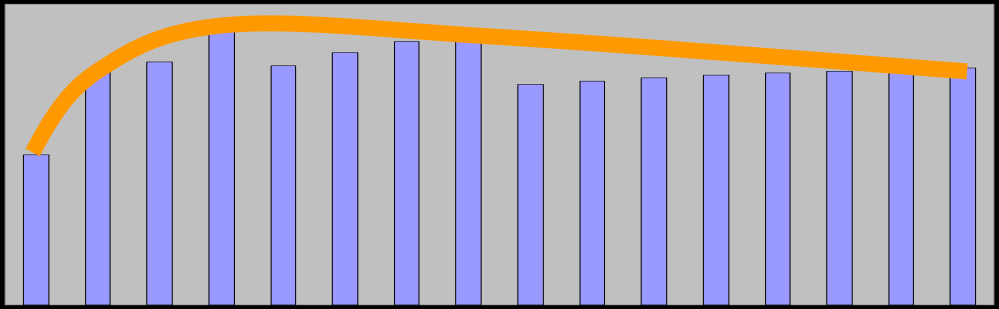
- $\varphi(n)/p$ as a function of p



- $\kappa(n,p)$ as a function of p



- Speedup as a function of p
  - elbowing out



Efficiency

- **Definition ε(n, p)**

$$\text{Efficiency} = \frac{\text{Sequential execution time}}{\text{Processors} \times \text{Parallel execution time}}$$

$$\text{Efficiency} = \frac{\text{Speedup}}{\text{Processors}}$$

- **0 <= ε(n, p) <= 1**

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

Amdahl's Law

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

Let f be fraction of sequential computations relative to all computations. Then $f = \sigma(n)/(\sigma(n) + \varphi(n))$

Amdahl's law states that in those conditions the maximum achievable speedup is:

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

Example

95% of a program's execution time occurs inside a loop that can be executed in parallel. What is the maximum speedup we should expect from a parallel version of the program executing on 8 CPUs?

$$\psi \le \frac{1}{0.05 + (1 - 0.05)/8} \cong 5.9$$

Example

20% of a program's execution time is spent within inherently sequential code.
What is the limit to the speedup achievable by a parallel version of the program?

$$\lim_{p \to \infty} \frac{1}{0.2 + (1 - 0.2)/p} = \frac{1}{0.2} = 5$$

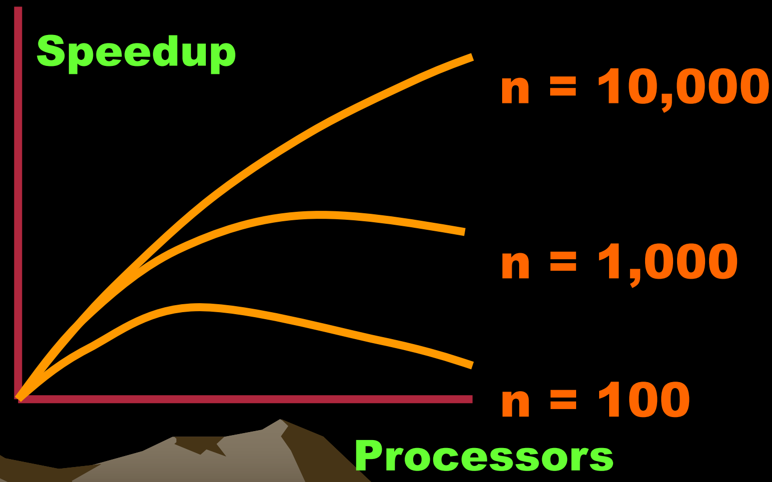
Question

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?

Conclusions about Amdhal's law

- Ignores $\kappa(n,p)$
- Overestimates speedup achievable
- But 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 (Amdahl effect)



Another perspective

- **We often use more processors to solve larger problem instances**
- **Let's treat time as a constant and allow problem size to increase with the number of processors**

Consider a parallel program solving a problem of size n using p processors. Let s be fraction spent in sequencial computations. Hence $s = \sigma(n)/(\sigma(n)+\varphi(n)/p)$.

Gustafson-Barsis's Law states that in those conditions the maximum speedup achievable by the program is

aka scaled speedup

$$\psi \leq p + (1 - p)s$$

Example

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

$$\psi = 10 + (1-10)(0.03) = 10 - 0.27 = 9.73$$

…except 9 do not execute serial code

Execution on 1 CPU takes 10 times as long…

Example

- **What is the maximum fraction of a program's parallel execution time that can be spent in serial code if it is to achieve a scaled speedup of 7 on 8 processors?**

$$7 = 8 + (1-8)s \Rightarrow s \approx 0.14$$

The Karp-Flatt Metric

- **Definition**

Inherently serial component of parallel computation + processor communication and synchronization overhead

Single processor execution time

aka Experimentally determined serial fraction

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

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

Experimentally determined serial fraction

- Takes into account parallel overhead
- Detects other sources of overhead or inefficiency ignored in speedup model
  - Process startup time
  - Process synchronization time
  - Imbalanced workload
  - Architectural overhead

Example

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?

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

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?

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.

Isoefficiency Metric

- Parallel system: parallel program executing on a parallel computer
- Scalability of a parallel system: measure of its ability to increase performance as number of processors increases
- A scalable system maintains efficiency as processors are added
- Isoefficiency: way to measure scalability

Isoefficiency Relation

Determine overhead time

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

Substitute overhead time into speedup equation

$$\psi(n,p) \le \frac{p(\sigma(n)+\varphi(n))}{\sigma(n)+\varphi(n)+T_0(n,p)}$$

Substitute T(*n*,1) = σ(*n*) + φ(*n*). Assume efficiency is constant.

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

Isoefficiency Relation

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

In order to maintain the same efficiency as p increases, n must be increased in order to satisfy the above inequality

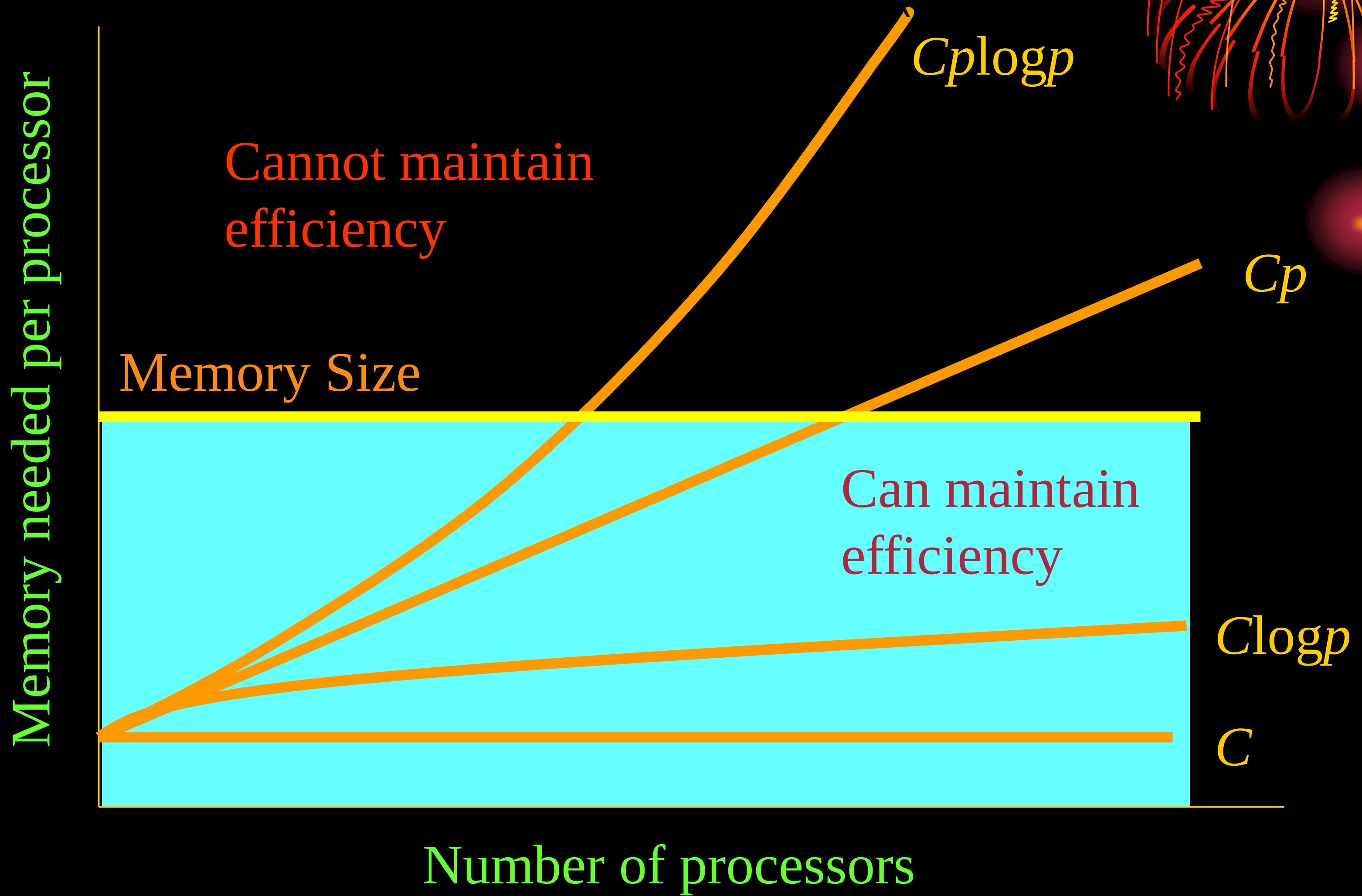
Scalability function

- Suppose that to verify the isoefficiency relation we need to satisfy $n \geq f(p)$
- Let $M(n)$ denote memory required for problem of size n
- $M(f(p))/p$ shows how memory usage per processor must increase to maintain same efficiency
- We call $M(f(p))/p$ 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 <u>per processor</u> must grow to maintain efficiency
- Scalability function a constant means parallel system is perfectly scalable

Interpreting Scalability Function



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**
  - $T_0(n,p) = \Theta(p \log p)$

Reduction (continued)

- **Isoefficiency relation:**
  - $n \geq C\,p \log p$
- **We ask: To maintain same level of efficiency, how must *n* increase when *p* increases?**
- **Memory usage:**
  - $M(n) = n$

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

- **The system has good scalability**

Example 2: Floyd's Algorithm

- **Sequential time complexity:**
  - $\Theta(n^3)$
- **Parallel computation time:**
  - $\Theta(n^3/p)$
- **Parallel communication time:**
  - $\Theta(n^2 \log p)$
- **Parallel overhead:**
  - $T_0(n,p) = \Theta(pn^2 \log p)$

Floyd's Algorithm (continued)

- **Isoefficiency relation**
  - $n^3 \geq C(p\, n^3 \log p) \Rightarrow n \geq C\, p \log p$
- **Memory usage:**
  - $M(n) = n^2$

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

- **The parallel system has poor scalability**

Example 3: Finite Differences

- **Sequential time complexity per iteration:**
  - $\Theta(n^2)$

- **Parallel communication complexity per iteration:**
  - $\Theta(n/\sqrt{p})$

- **Parallel overhead:**
  - $\Theta(n\sqrt{p})$

Finite Differences (continued)

- **Isoefficiency relation**
  - $n^2 \geq C\, n \sqrt{p} \Rightarrow n \geq C \sqrt{p}$
- **Memory usage:**
  - $M(n) = n^2$

$$M(C\sqrt{p})/p = C^2 p/p = C^2$$

- **This algorithm is perfectly scalable**