EE/CSCI 451: Parallel and Distributed Computation

Lecture #11

9/24/2020

Viktor Prasanna

prasanna@usc.edu

ceng.usc.edu/~prasanna

University of Southern California

Announcement

- Midterm 1 on 9/25
- PHW3 due 9/24
- HW4 due 9/24
- HW3 grades are out

HW3 Statistics	
Average	90.4
Median	94
Standard Deviation	13.9

Announcement: Midterm 1 Logistics

- **Open Book, Open Notes**
- Time: 9/25 3:30-5:30 PM (Los Angeles time)
- Exam will be released on **Piazza**
- Upload your answer (pdf file) on **Blackboard**
- Completing your exam:
  - Option 1 - Download the exam as a pdf file onto your tablet and annotate it with your answers.
  - Option 2 – Download and print the exam and write your answers on the (printed) paper. Scan the paper and save into PDF format.
- Turn on your camera on Zoom: We will be proctoring!

Midterm 1 Details

- Materials covered till the end of last week

- 5 problems in total

  - Memory System Performance Modeling

  - Shared Memory Programming

  - Shared Memory Programming

  - Message Passing Protocols & Programming

  - Interconnection Networks

Course Info.

- Academic Integrity
  - Cheating will not be tolerated
  - Grade of F will be assigned
  - Cheaters will be reported to USC Student Judicial Affairs and Community Standards (SJACS)

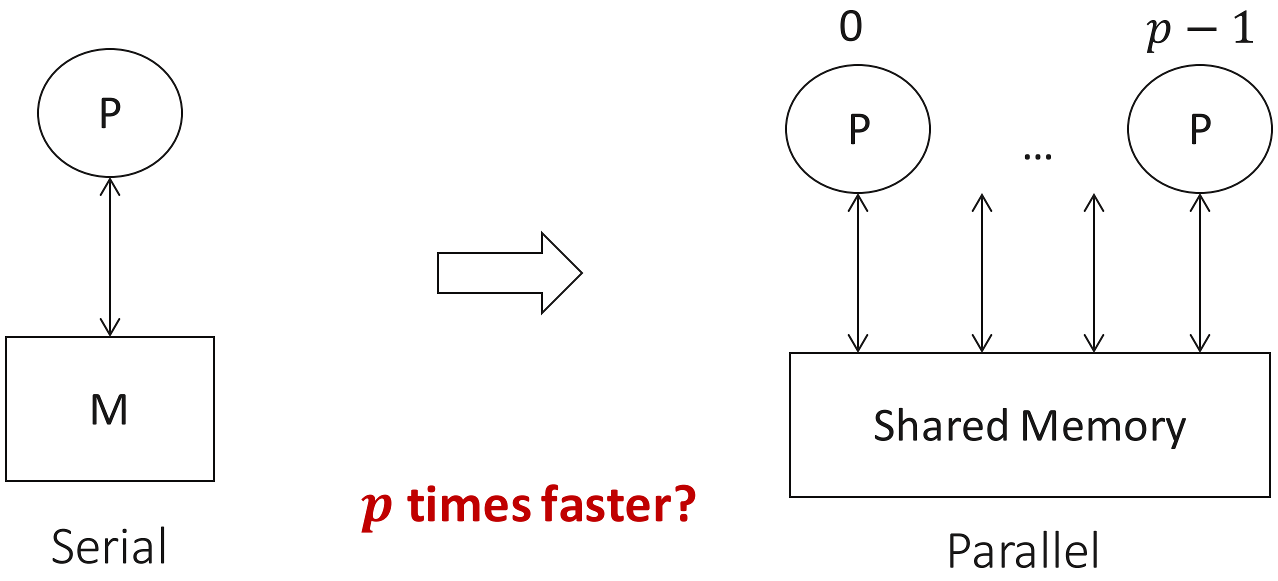
Outline

- From last class
  - Program and Data Mapping
    - Graph Embedding
      - Dilation
      - Expansion
      - Congestion
  - Network Model
  - Simulations
  - Hypercube on 1-D mesh (dilation and congestion)

- Today
  - Analytical Modeling of Parallel Systems (Chapter 5.2, 5.4.1)
    - Scalability
    - Achievable Speedup
      - Amdahl's Law
      - Gustafson's Law
    - Efficiency
    - Work Optimal parallel solution
    - Performance analysis
      - Big O notation

Scalability (1)

Does performance (execution time) improve as we use more resources (processors)



***p* times faster?**

Scalability (2)

$$\text{Speedup} = \frac{\text{Serial time (on a uniprocessor system)}}{\text{Parallel time using } p \text{ processors}}$$

If speedup = $O(p)$, then it is a **scalable** solution

Overheads in Parallel Computation

- Communication

- Coordination

- Load balance (processors may idle)

Amdahl’s Law (1)

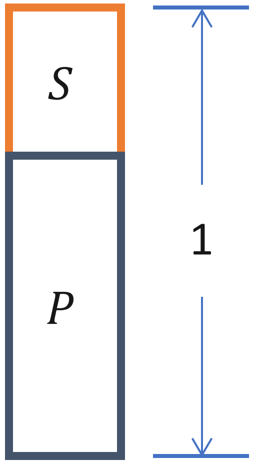
- Amdahl’s Law — Limit on speedup achievable when a program is run on a parallel machine
- Given an input program
  - Total execution time: $S + P$

Serial Portion

Parallelizable Portion

  - Time on a uni-processor machine: $1 = S + P$
  - Time on a parallel machine: $S + P/f$

    f = speedup factor



Amdahl's Law (2)

Example:

1 unit of time = 1 arithmetic operation

$$S = \frac{n}{2n}$$

$$P = \frac{n}{2n}$$

$$S + P = 1$$

```
$A(0) \leftarrow 0$

Do $i = 1$ to $n$
    $A(i) \leftarrow A(i) + A(i-1)$
```

n arithmetic operations

Serial operations

```
Do $i = 1$ to $n$
    $A(i) \leftarrow A(i) * A(i)$
```

n arithmetic operations

Parallelizable operations

Amdahl's Law (3)

Overall speedup

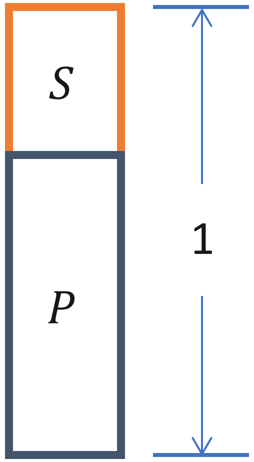
$$= \frac{\text{Serial Time}}{\text{Parallel Time}}$$

$$= \frac{S + P}{S + P/f}$$

f = speedup factor

$$= \frac{1}{S + P/f}$$

$$\leq \frac{1}{S}$$

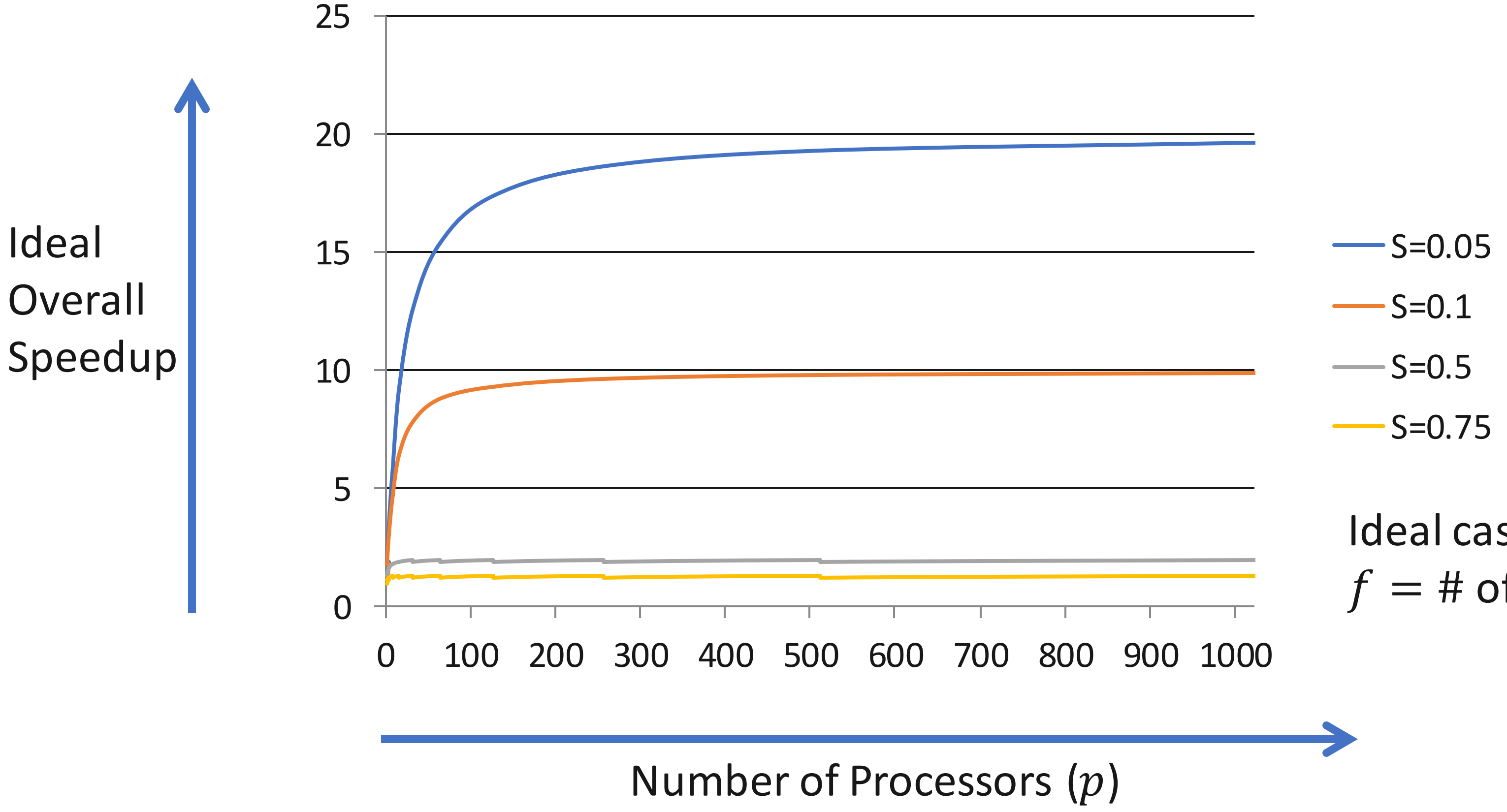


If serial portion is 50 %, then the **overall speedup ≤ 2**

Note: Speedup factor (f) ≤ # of processors

Amdahl’s Law (4)

$$\text{Overall Speedup} = \frac{1}{S + P/f} = \frac{1}{S + (1 - S)/f} \rightarrow \frac{1}{S}$$

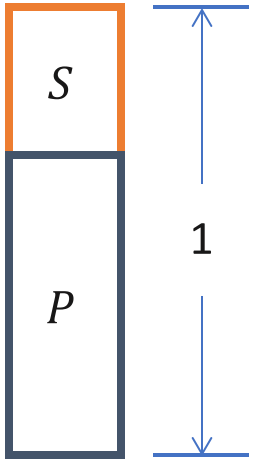


Ideal case

f = # of processors

Amdahl’s Law (5)

(Overall) Speedup is upper bounded by $\frac{1}{\text{Serial portion}}$ $(\frac{1}{S})$

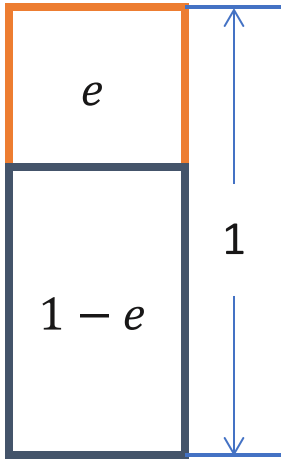


Amdahl's Law (6)

An alternate interpretation

- **Amdahl's Law for Energy:** Limit on energy improvement when only part of the energy consumption can be improved

- Given a program
  - Total energy consumed by a code: 1
  - The portion of energy that can not be improved: e (ex. Static Power)
  - The portion of energy that can be improved: $1 - e$ (ex. Dynamic Power)
    - Energy dissipation improvement factor = f

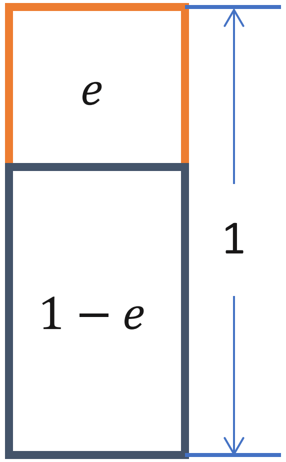


Amdahl's Law (7)

Overall Energy Improvement

$$= \frac{1}{e + (1-e)/f}$$

$$\leq \frac{1}{e}$$ (When f is large)



If e is 50 %, the overall energy improvement ≤ 2

Scaled Speedup (Gustafson's Law) (1)

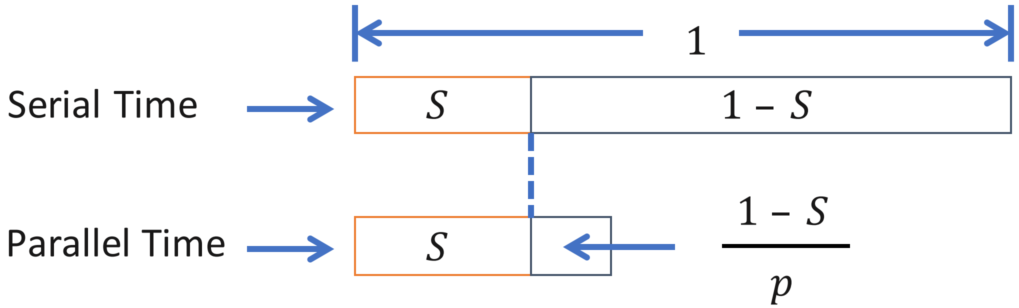
- Amdahl's Law — Serial portion of code limits performance
  (As we use more processors)

- As we use more processors
  → we use more data
  → e.g., more fine grained model ⇨ more opportunities for parallelism

- E.g. Processing $N \times N$ image
  Using $p \times p$ processor array
  As we increase p we usually increase image size

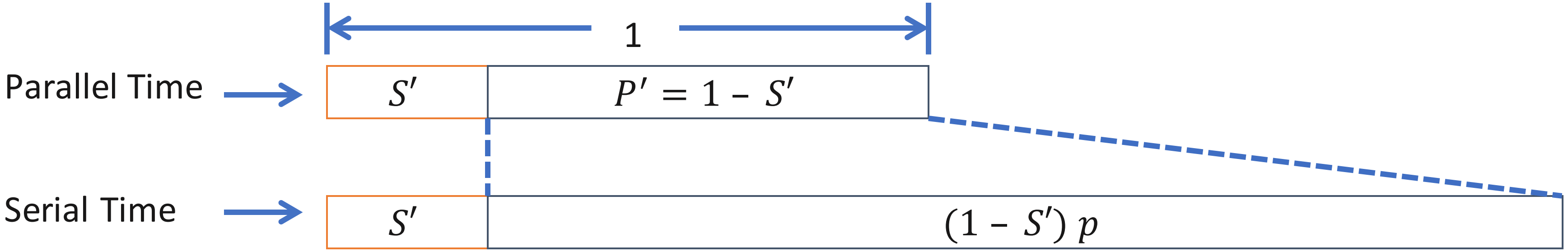
Scaled Speedup (Gustafson's Law) (2)

Amdahl's Law: Fixed amount of computations



p : Number of Processors

Gustafson's Law: Increase p and amount of computations



If parallelism scales linearly with p, number of processors

$$\text{Scaled Speedup} = \frac{\text{Serial time}}{\text{Parallel time}} = \frac{S' + (1 - S')p}{S' + P'} = \frac{S' + (1 - S')p}{1}$$

Scaled Speedup (Gustafson's Law) (3)

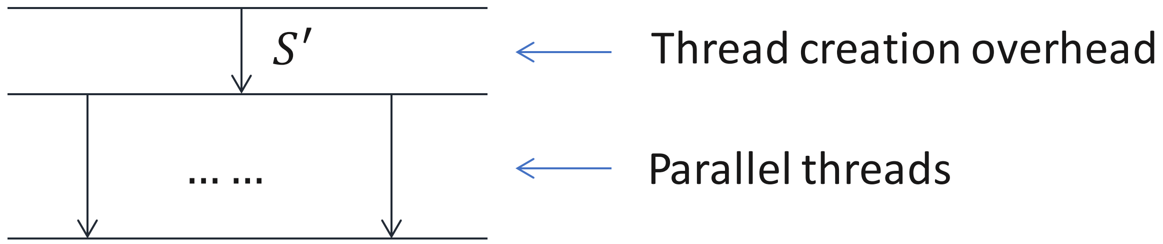
Gustafson's Law

$$\text{Scaled Speedup} = S' + (1 - S')p$$
$$\approx (1 - S')p$$

$S' = 0$ → Scaled Speedup $=$ Ideal speedup (p)

$S' = 0.5$ → Scaled Speedup $= 0.5\,p$, 50% of Ideal speedup

Example: Cloud



Scaled Speedup $\propto (1 - S')$ × # of threads

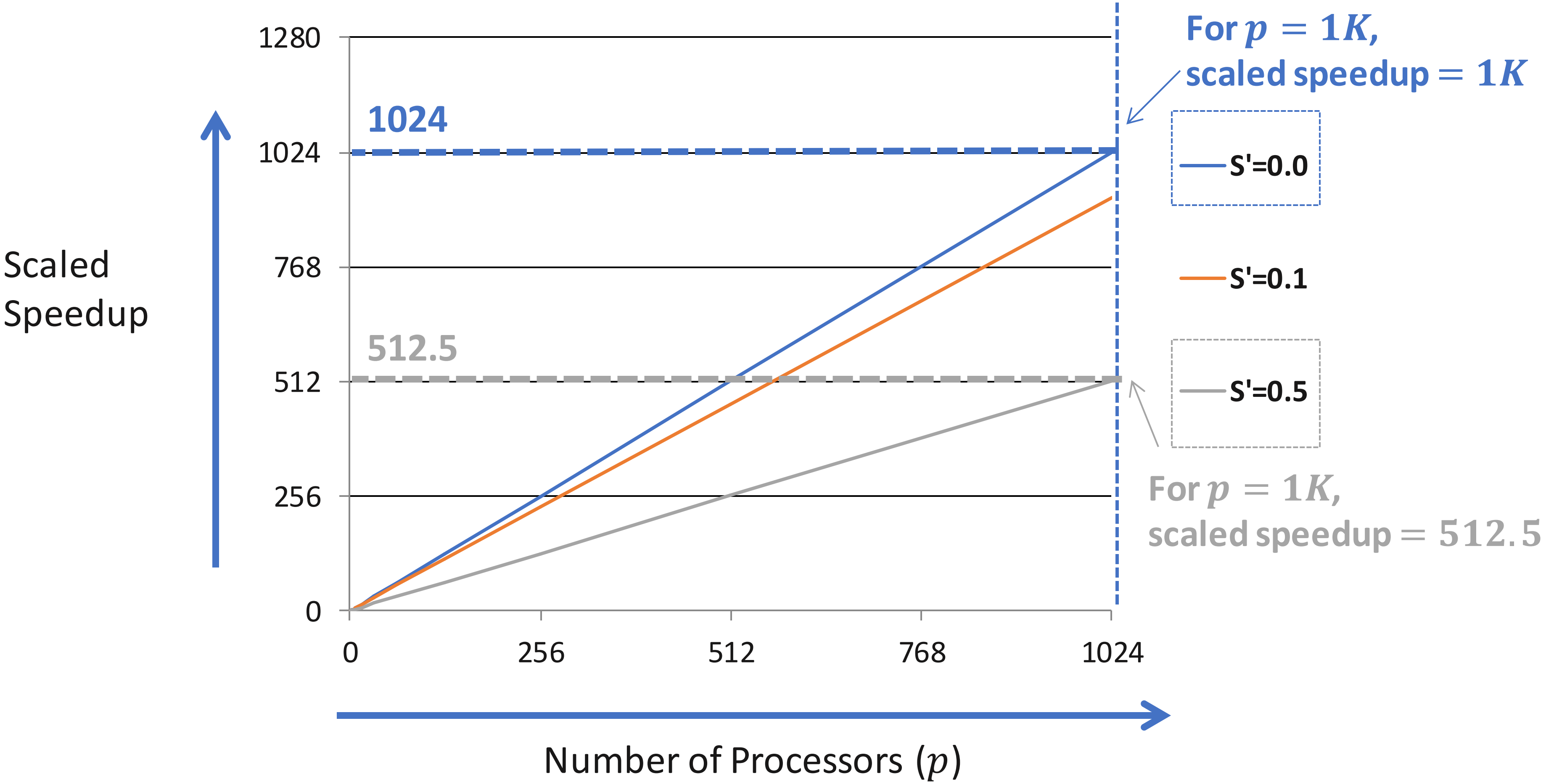
Scaled Speedup (Gustafson’s Law) (4)

Gustafson’s Law

- If we increase
  - the number of processors (p)
  - the amount of parallelizable portion of computations

- Scaled speedup is limited by the fraction of program that can be parallelized (higher the fraction, higher the speedup).

Scaled Speedup (Gustafson’s Law) (5)



For $p = 1K$,
scaled speedup $= 1K$

For $p = 1K$,
scaled speedup $= 512.5$

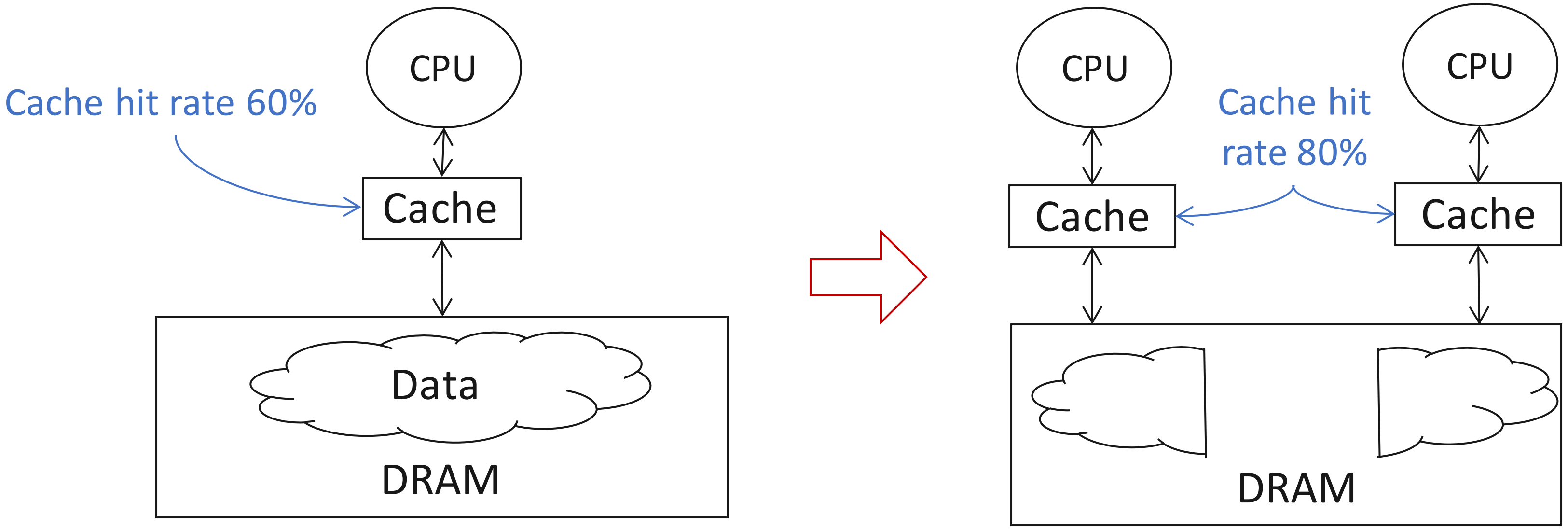
Strong or Weak Scaling

- Strong Scaling
  - Governed by Amdahl’s Law
  - The number of processors is **increased** while the problem size **remains constant**
  - Results in a **reduced** workload per processor

- Weak Scaling
  - Governed by Gustafson’s Law
  - **Both** the number of processors and the problem size are **increased**
  - Results in a **constant** workload per processor

Superlinear Speedup (1)

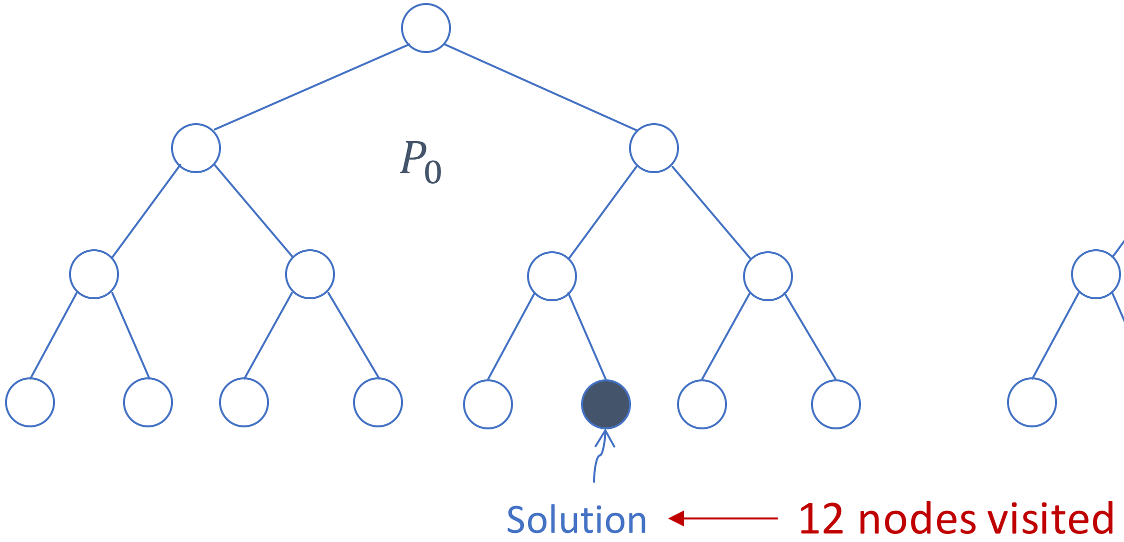
- Achieved speedup > number of processors
  - Hardware features (ex. cache effect)



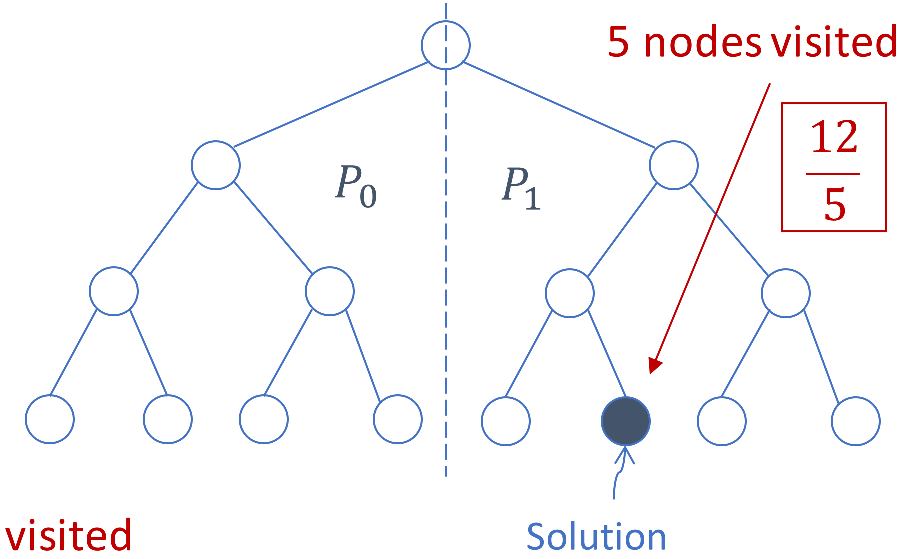
Superlinear Speedup (2)

- Achieved speedup > number of processors
  - Work performed by serial algorithm is greater than its parallel formulation

Tree search (in order traversal) using one processor



Tree search using two processors



Performance (1)

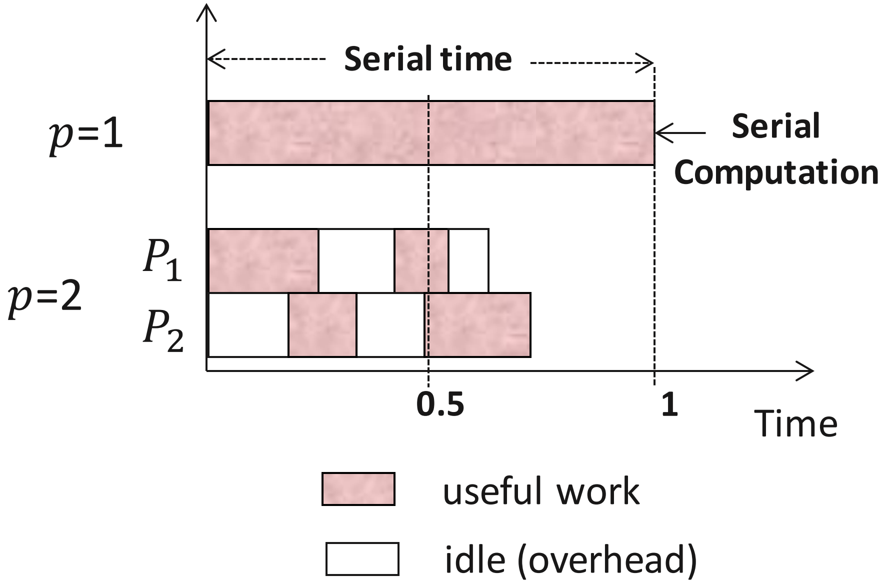
Efficiency

- Question: If we use p processors, is speedup = p ?

- Efficiency $\triangleq$ Fraction of time a processor is usefully employed during the computation

- E = Speedup / # of processors used
  - E is the average efficiency over all the processors
  - Efficiency of each processor can be different from the average value

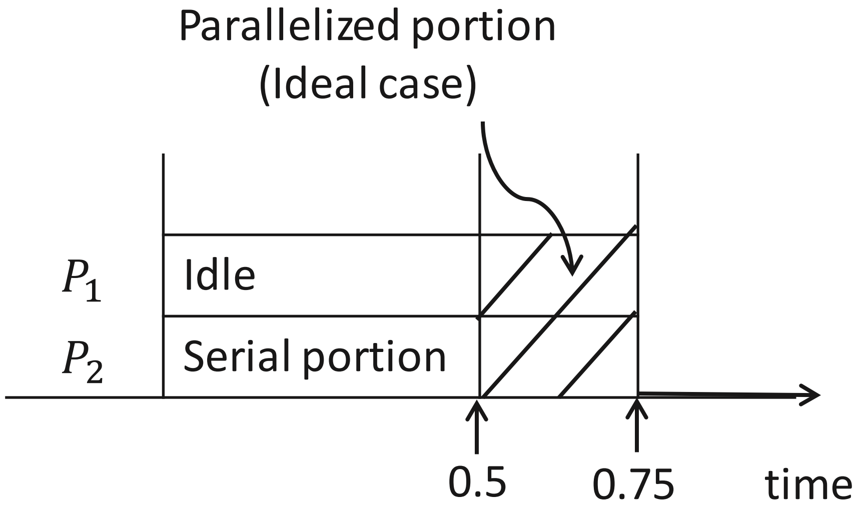
Typical execution of a program on a parallel machine



Performance (2)

Ex. $S = 0.5$
$P = 0.5$

2 processor system



$$\text{Speedup} = \frac{1}{0.75} = {}^{4}/_{3}$$

$$\text{(Average) Efficiency} = \frac{{}^{4}/_{3}}{2} = {}^{2}/_{3}$$

$$\text{Efficiency of } P_1 = \frac{0.25}{0.75} = {}^{1}/_{3}$$

$$\text{Efficiency of } P_2 = \frac{0.75}{0.75} = 1$$

$$\text{(Average) Efficiency} = \frac{{}^{1}/_{3}+1}{2} = 2/3$$

Performance (3)

- Cost = Total amount of work done by a parallel system

  = Parallel Execution Time x Number of Processors

  = $T_p \times p$

- Cost is also called Processor Time Product

- COST OPTIMAL (or WORK OPTIMAL) Parallel Algorithm
  - Total work done = Serial Complexity of the problem

Performance (4)

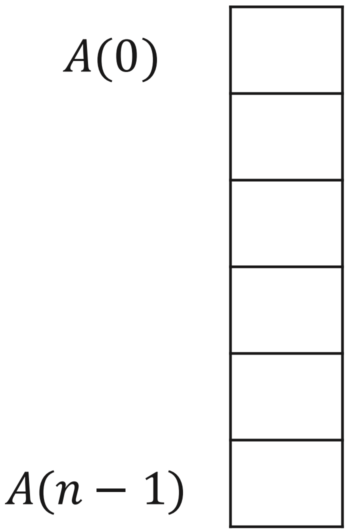
- Example: addition on PRAM
  - – n processor PRAM
  - – n input data
  - – Add n numbers

Performance (5)

- Algorithm

Program in processor $j, 0 \leq j \leq n-1$

```
1.  Do $i = 0$ to $\log_2 n - 1$
2.      If $j = k \cdot 2^{i+1}$, for some $k \in N$
            then $A(j) \leftarrow A(j) + A(j + 2^i)$
3.  end
```



Note:

A is shared among all the processors
Synchronous operation [For ex. all the processors execute instruction 2 during the same cycle, $\log_2 n$ time]
N = set of natural numbers = {0, 1, …}

Parallel time = $O(\log n)$

Performance (6)

Serial time = $O(n)$ (Serial complexity)

Parallel time = $O(\log n)$

Speedup $= O(n/\log n)$

# of processors $= n$

$$E = \frac{O(n/\log n)}{n} = O(1/\log n)$$

NOT WORK OPTIMAL

Performance Analysis (1)

Asymptotic Analysis

Big O Notation or Order Notation

Worst case execution time of an algorithm

Upper bound on the growth rate of the execution time

Example: $n \times n$ matrix multiplication

```
Do $i$
    Do $j$
        $C(i,j) \leftarrow 0$
        Do $k = 1$ to $n$
            $C(i,j) \leftarrow C(i,j) + A(i,k) * B(k,j)$
        End
    End
End
```

$T(n) =$ time complexity function $= n^2 + n^3 + n^3$

Performance Analysis (2)

- Actual execution time depends on the processor infrastructure, compiler, etc.

  Number of computation steps is upper bounded by cn^3
  For some constant c, c does not depend on n.

  We say $T(n) = O(n^3)$

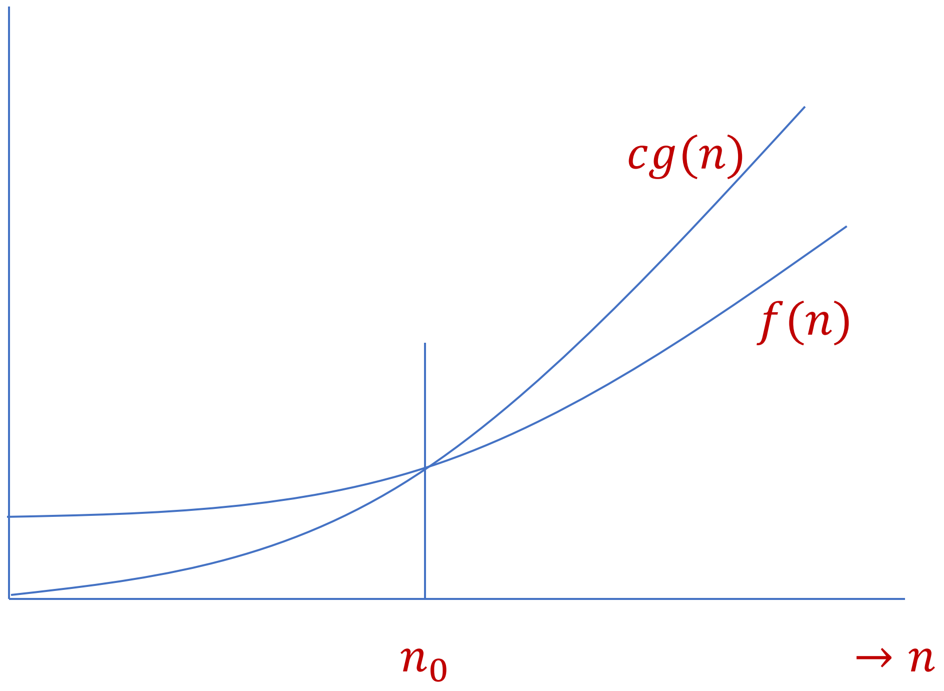
- Definition:

  $f(n)$ is $O(\mathrm{g}(n))$ if there is a constant c such that
  $f(n) \leq c \cdot \mathrm{g}(n)$ for sufficiently large n, i.e. there
  exists n_0 such that for all $n \geq n_0$, $f(n) \leq c \cdot \mathrm{g}(n)$

- Ex: $f(n) = 2n^3 + n^2 + n$
  $f(n) \ = \ O(n^3)$

Performance Analysis (3)

$$\boldsymbol{f(n) = \mathrm{O}(g(n))}$$



Performance Analysis (4)

- Lower bound on execution time
  - Definition: $f(n) = \Omega(\mathrm{g}(n))$

    if there exist constants c and n_0 such that

    for all $n \geq n_0$, $f(n) \geq c \cdot \mathrm{g}(n)$

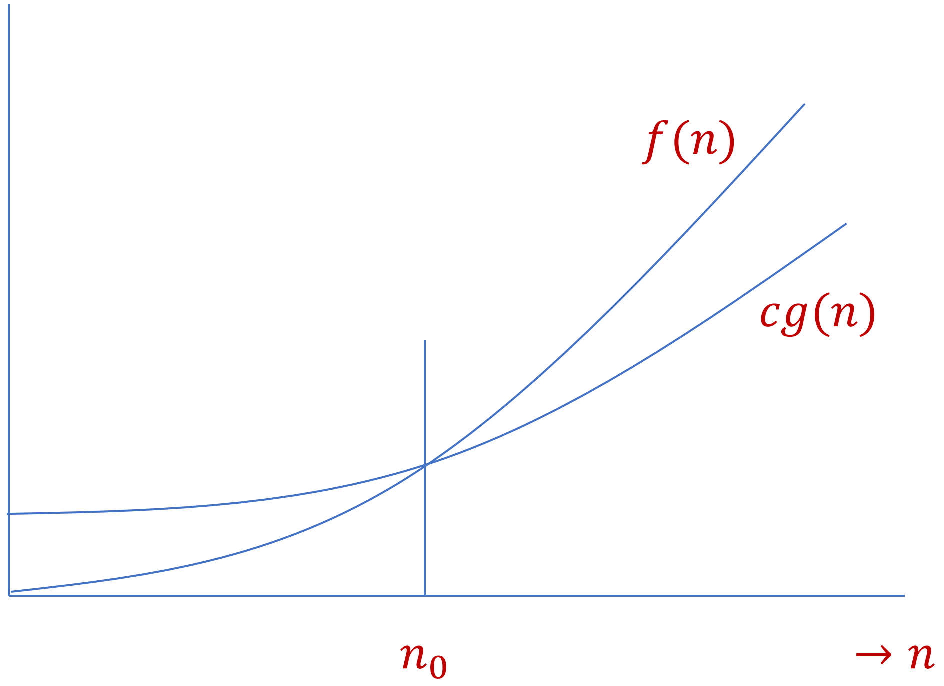
  ex: $f(n) = n^3 + n^2 + n$ then, $f(n) = \Omega(n^3)$

- Tight bound on execution time
  - Definition: $f(n) = \Theta(\mathrm{g}(n))$

    if $f(n) = O(\mathrm{g}(n))$ and $f(n) = \Omega(\mathrm{g}(n))$

Performance Analysis (5)

$$f(n) = \Omega(g(n))$$



Performance Analysis (6)

Ex. Execution time of two algorithms

$$A_1: \;\; T_1(n) = 100n$$

$$A_2: \;\; T_2(n) = n^2$$

A_1 is asymptotically superior to A_2

Ex.

$$A_1: \;\; T_1(n) = 5n^2$$

$$A_2: \;\; T_2(n) = 100n^2$$

A_1 and A_2 are asymptotically of the same complexity

$$T_1(n) = T_2(n) = \theta(n^2)$$

Performance Analysis (7)

$f_1(n) = n$

$f_3(n) = n^2$

$f_2(n) = n \log n$

$f_4(n) = n^{1+\varepsilon}, \quad 0 < \varepsilon < 1$

$f_1(n) = O\big(f_2(n)\big)$

$f_2(n) = O(f_3(n))$

$f_2(n) = O(f_4(n))$ ✓

$f_2(n) = \Omega(f_3(n))$ ×

$f_4(n) = O(f_3(n))$ ✓

$f_2(n) = \Omega(f_4(n))$ ×

Summary

- Scalability
- Achievable Speedup
  - Amdahl’s Law
  - Gustafson’s Law
- Efficiency
  - Processor Time Product
  - COST OPTIMAL (work optimal)
- Performance Analysis