CS586: Distributed Computing

Tutorial 1

Professor: Panagiota Fatourou
TA: Eleftherios Kosmas
CSD - October 2011

Amdahl's Law

- It is used to predict the **theoretical maximum speedup** of a sequential program, when it is parallelized and executed in parallel

- **Basic observation**: "a portion of a sequential program may not be parallelizable"

Amdahl's Law

- Let *A* be a **sequential** program
  - assume that *P* is the portion of *A* that **can** be parallelized
  - then, *1-P* is the portion of *A* that **can not be** parallelized
- Assume that the execution time of *A* is 1 unit of time
- Then, the execution time of the *A*'s parallel version in a multicore machine with N processors is:

Time to execute the sequential portion of *A*

$$1 - P + \frac{P}{N}$$

Time to execute the parallel portion of *A*

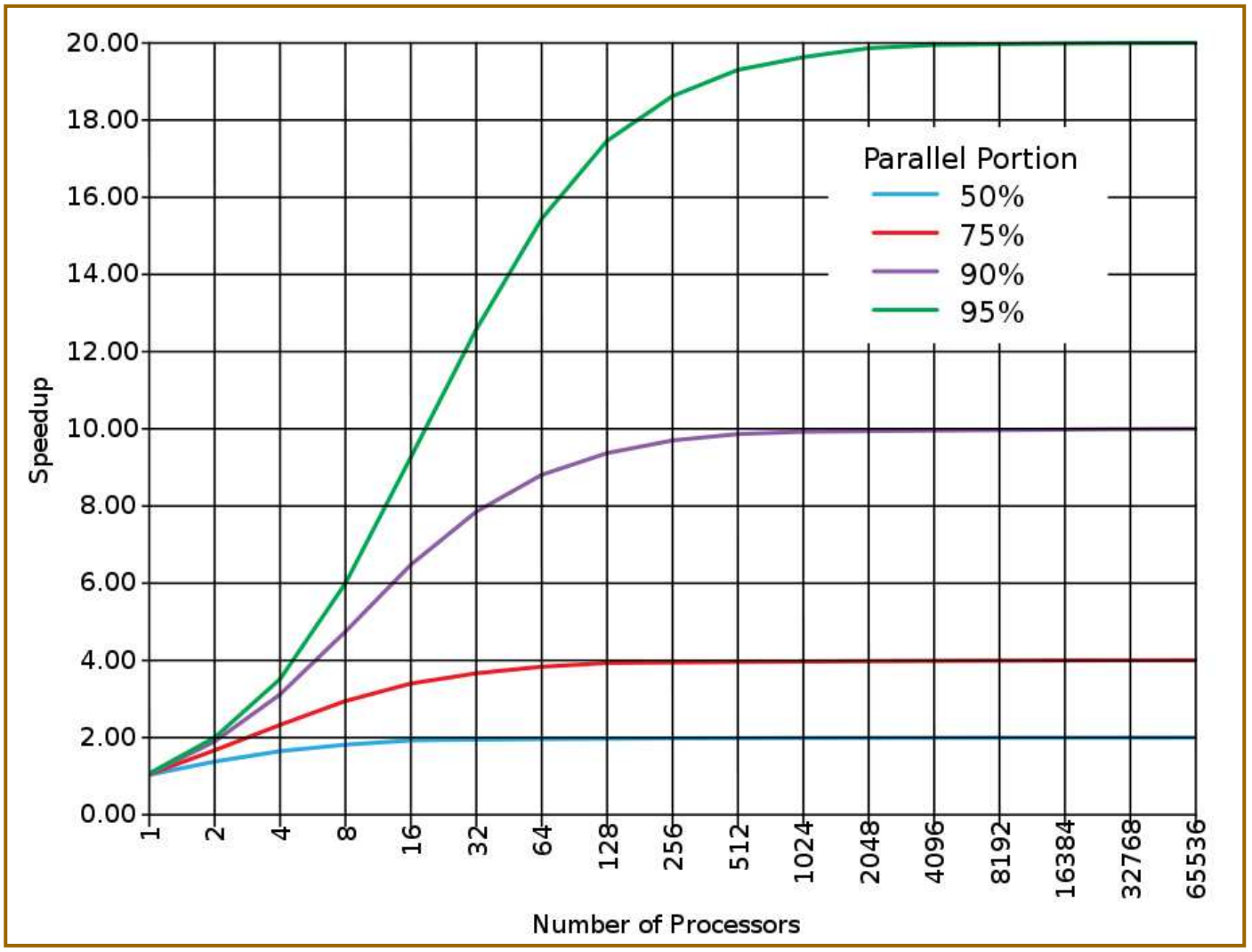
- Maximum speedup (S):

$$S = \frac{1}{1 - P + \frac{P}{N}}$$

Amdahl's Law

- Maximum speedup (S): $S = \frac{1}{1 - P + \frac{P}{N}}$
- When N tends to infinity then : $S = \frac{1}{1 - P}$
  - thus, when only 10% of a program can not be parallelized, then the theoretical maximum speedup that can be achieved is only 10, no matter the number of processors

Amdahl's Law



* Figure from: http://en.wikipedia.org/wiki/Amdahl%27s_law

Amdahl's Law

- **Conclusion:** "A parallel application can not run faster than it's sequential portion"
  - Therefore, based on Amdahl's law, only embarrassingly parallel programs (with high values of *P*) are suitable for parallel computing

- However, Amdahl's law assumes that the size of a problem remains constant, while the number of processors is increased
  - if this size is not fixed then … → **Gustafson's Law**

Gustafson's Law

- Let A^p be a **parallel** program
  - assume that P is the parallel portion of A^p
  - then, $1\text{-}P$ is the sequential portion of A^p
- Assume that the execution time of A^p is 1 unit of time
- Then, the execution time of A^p's sequential version in a single processor machine is:

Time to execute the sequential portion of A^p $\quad 1-P+P\cdot N \quad$ Time to execute the parallel portion of A^p

- Maximum speedup (S): $S=1-P+P\cdot N$

Gustafson's Law

- Maximum speedup (S):
  - $S = 1 - P + P \cdot N$, or
  - $S = a + (1 - a) \cdot N$, where $1\text{-}P = a$
- Assuming that the sequential portion (a) of A^p diminishes as N tends to infinity then : $S = N$

Gustafson's Law

Amdahl's Law suggests:

- Suppose a car is traveling between two cities 60 miles apart, and has already spent one hour traveling half the distance at 30 mph.
- No matter how fast you drive the last half, it is impossible to achieve 90 mph average before reaching the second city.
- Since it has already taken you 1 hour and you only have a distance of 60 miles total; going infinitely fast you would only achieve 60 mph.

Gustafson's Law suggests:

- Suppose a car has already been traveling for some time at less than 90mph.
- Given enough time and distance to travel, the car's average speed can always eventually reach 90mph, no matter how long or how slowly it has already traveled.
- For example, if the car spent one hour at 30 mph, it could achieve this by driving at 120 mph for two additional hours, or at 150 mph for an hour, and so on.

This example is presented in wikipedia.org

Practice

Exercise 1:

- Suppose a computer program has a method M that cannot be parallelized, and this method accounts for 40% of the program's execution time.
  - What is the limit for the overall speedup that can be achieved by running the program on an n-processor multiprocessor machine?

Exercise 2:

- You have a choice between buying one uni-processor that executes $5*10^{15}$ instructions per second, or a ten-processor multiprocessor where each processor executes 10^{15} instructions per second.
  - Using Amdahl's Law, explain how you would decide which to buy for a particular application.

Why synchronization is necessary

- Sequential implementation of a counter:
  - *Read(c)* : *return counter;*
  - *Increment(C)* : *counter++;* ⟹
    tmp = counter;
    tmp = tmp + 1;
    counter = tmp;
- Assume we have to implement a counter in a multiprocessor system
  - is the sequential implementation still correct?
- Consider the following execution (*counter* is initially 0):

Process p_A	Process p_B
tmp_A = counter;	
tmp_A = tmp_A + 1;	
	tmp_B = counter;
	tmp_B = tmp_B + 1;
counter = tmp_A;	
	counter = tmp_B;

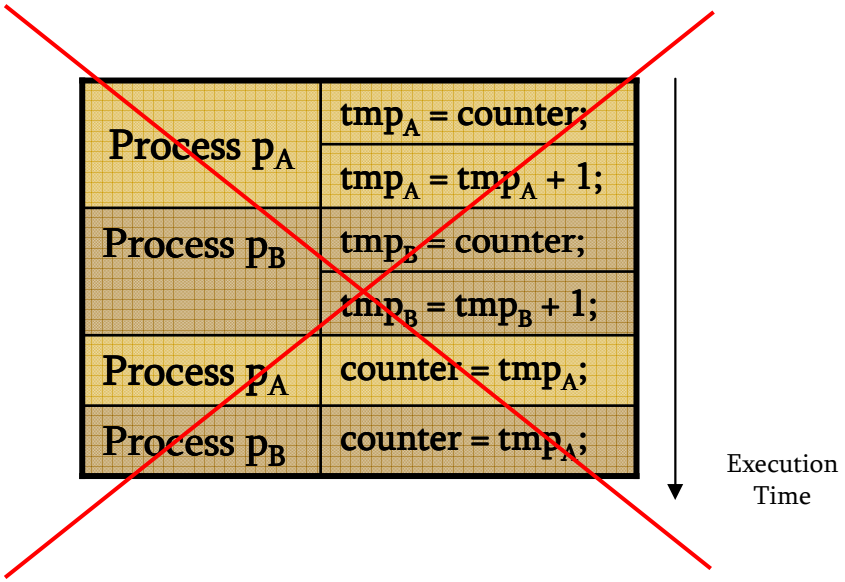
Execution Time

✓ Expected new value of counter: 2

✘ **Actual** new value of counter: **1 !**

☞ ***counter*** **may result with a value from {1,2}**

Don’t use this representation



Why synchronization is necessary

- Consider another execution: (*counter* is initially 0)

Process p_A	Process p_B	Process p_C
tmp_A = counter;		
tmp_A = tmp_A + 1;		
	tmp_B = counter;	
	tmp_B = tmp_B + 1;	
		tmp_C = counter;
		tmp_C = tmp_C + 1;
counter = tmp_A;		
	counter = tmp_B;	
		counter = tmp_C;

Execution Time

✓ Expected new value of counter: 3

✗ **Actual** new value of counter: **1 !**

☞ ***counter*** **may result with a value from {1,2,3}**

Why synchronization is necessary

- A wrong execution where counter results with value 2 (*counter* is initially 0):

Process p_A	Process p_B	Process p_C
$tmp_A = counter;$		
$tmp_A = tmp_A + 1;$		
	$tmp_B = counter;$	
	$tmp_B = tmp_B + 1;$	
$counter = tmp_A;$		
	$counter = tmp_B;$	
		$tmp_C = counter;$
		$tmp_C = tmp_C + 1;$
		$counter = tmp_C;$

Execution Time

- ✓ Expected new value of counter: 3
- ✗ **Actual** new value of counter: **2 !**

☞ ***counter*** **may result with a value from {1,2,3}**

Why synchronization is necessary

- Let each process increment 3 times the *counter*
- Consider the following execution (*counter* is initially 0):

Process p_A	Process p_B
$tmp_A = counter;\ tmp_A = tmp_A + 1;$	
	$tmp_B = counter;\ tmp_B = tmp_B + 1;$
$counter = tmp_A;$	
$tmp_A = counter;\ tmp_A = tmp_A + 1;$	
$counter = tmp_A;$	
	$counter = tmp_B;$
$tmp_A = counter;\ tmp_A = tmp_A + 1;$	
	$tmp_B = counter;\ tmp_B = tmp_B + 1;$
	$counter = tmp_B;$
	$tmp_B = counter;\ tmp_B = tmp_B + 1;$
	$counter = tmp_B;$
$counter = tmp_A;$	

Execution Time

- ✓ Expected new value of counter: 9
- ✘ **Actual** new value of counter: **2 !**

☞ ***counter*** **may result with a value from {2,…,9}**

Why synchronization is necessary

- What values can the *counter* take:
  - when 2 process increments it without synchronization and each process increments it 10 times?
    - {2, …, 20}
  - when 2 process increments it without synchronization and each process increments it k times?
    - {2, …, 2*k}
  - when 3 processes increments it without synchronization and each process increments the counter k times?
    - {2, …, 3*k}
  - when n processes increments it without synchronization and each process increments the counter k times?
    - {2, …, n*k}

The End - Questions