



POLITECNICO
MILANO 1863

DIPARTIMENTO DI ELETTRONICA
INFORMAZIONE E BIOINGEGNERIA



Exercise Session 1

Performance, Amdahl's Law, Pipeline

Advanced Computer Architectures

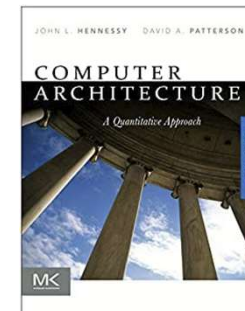
10th March 2024

Davide Conficconi <davide.conficconi@polimi.it>

Important Things: Material (EVERYTHING OPTIONAL)

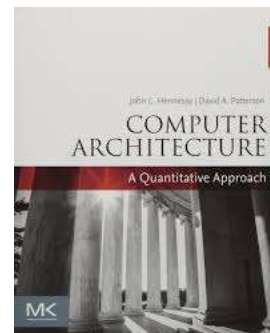
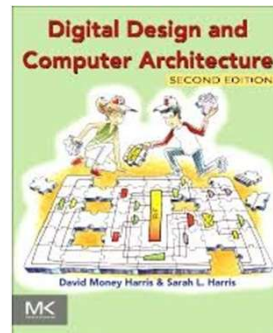
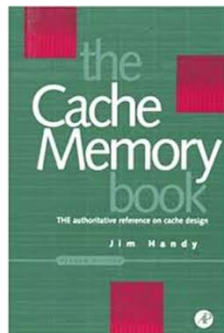
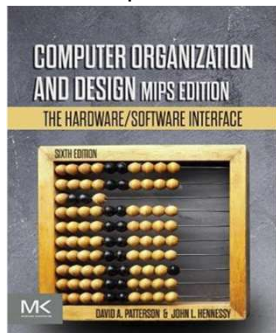
<https://webeep.polimi.it/course/view.php?id=14754>

Textbook: Hennessy and Patterson, Computer Architecture: A Quantitative Approach

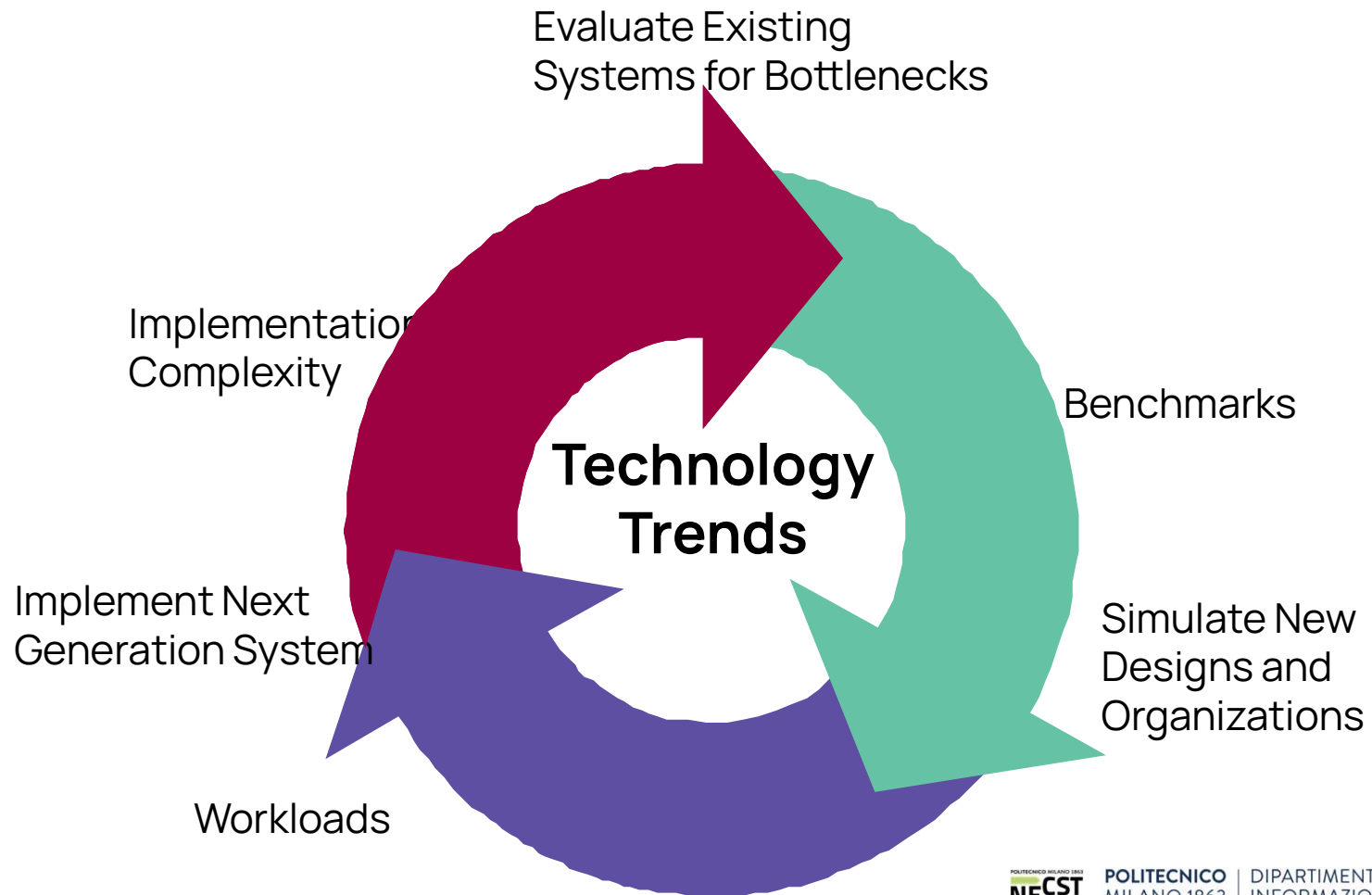


book about
also topics

Other Interesting Reference



Recall: Computer Engineering Methodology



Exe 1: Throughput vs Response time

(or latency)

Recall: Throughput vs Response time

- Two Metrics:

- Computer system user

- Minimize elapsed time for program execution:

response time: execution time = $\text{time}_{\text{end}} - \text{time}_{\text{start}}$
(latency)

- Computer center manager

- Maximize completion rate = #jobs/sec

throughput: total amount of work done in a given time

Exe 1: Throughput vs Response time



Exe 1: Throughput vs Response time

What will happen if...



Exe 1: Throughput vs Response time

What will happen if...

(1) we replace with a faster version?

(1)



Exe 1: Throughput vs Response time

What will happen if...

(1) we replace with a faster version?

(2) We add multiple parallel systems for independent tasks?

throughput will increase
if our workloads can be
parallelized

(1)



(2)



Case 1: Scale-up

(1)



Case 2: Scale-out

(2)

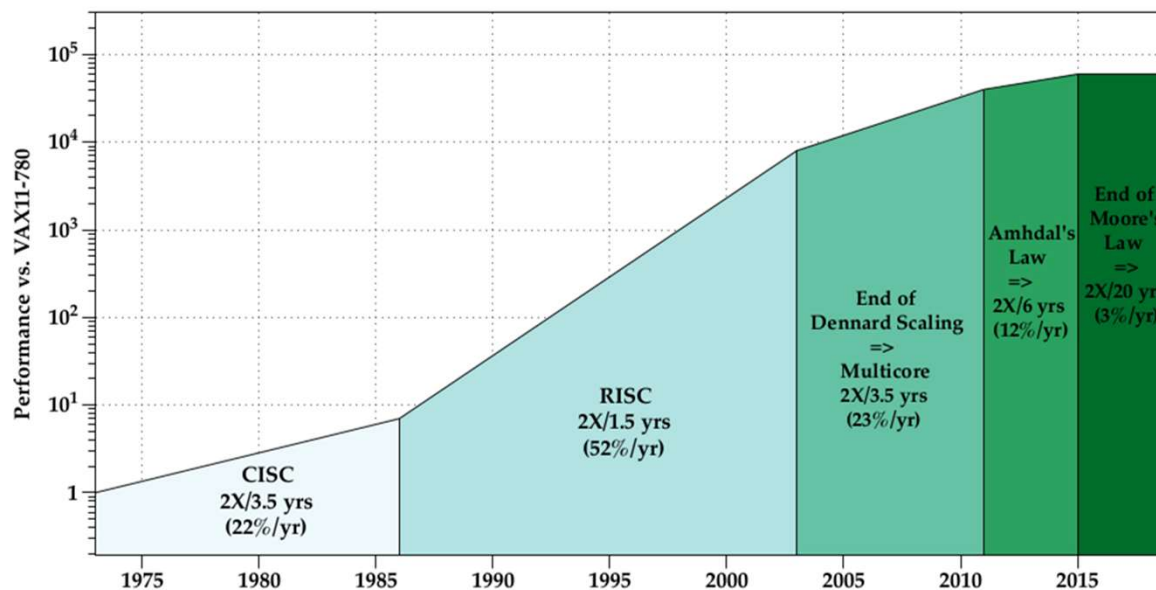




Recall: Issues as new opportunities

Programming has become very difficult
Impossible to balance all constraints manually

- More computational horse-power than ever before
 - Cores are free (almost ... <https://doi.org/10.1145/2000064.2000108>)
 - Energy (i.e., perf/joule) is **ALWAYS** a primary concern → Scaling (strong vs weak) <https://www.kth.se/blogs/pdc/2018/11/scalability-strong-and-weak-scaling>



Adapted from E. Del Sozzo. On how to effectively target fpgas from domain specific tools. 2019.
Data from: J. L. Hennessy and D. A. Patterson. Computer architecture: a quantitative approach 6th edition. Elsevier, 2018.

Recall: Some Factors Affecting Performance

Recall: Some Factors Affecting Performance

Algorithm complexity and data set

Compiler

Instruction set

Available operations

Operating system

Clock rate

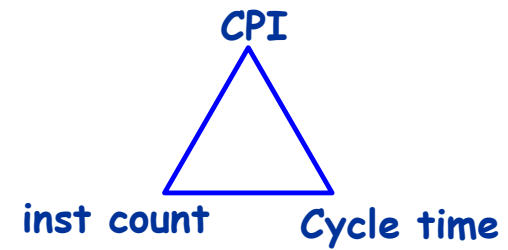
Memory system performance

I/O system performance and overhead

Recall: CPU time

- Instruction Count, IC
 - Instructions executed, not static code size
 - Determined by algorithm, compiler, Instruction Set Architecture
- Cycles per instructions, CPI
 - Determined by ISA and CPU organization
 - Overlap among instructions (pipelining) reduces this term
- Time/cycle
 - Determined by technology, organization and circuit design

Recall: Performance equation



	Inst. Count	CPI	Clock Rate
Program	X		
Compiler	X	(X)	
Instr. Set	X	X	
Organization		X	X
Technology			X

Exe 2: Performance Problem

Exe 2: Performance Problem

Consider two CPUs: CPU1 and CPU2.

CPU1 has clock cycle of 2 ns while CPU2 has an operating frequency of 700MHz.

Exe 2: Performance Problem

Consider two CPUs: CPU1 and CPU2.

CPU1 has clock cycle of 2 ns while CPU2 has an operating frequency of 700MHz.

Given the following frequencies of occurrence of the instructions for the two CPUs

Exe 2: Performance Problem

Consider two CPUs: CPU1 and CPU2.

CPU1 has clock cycle of 2 ns while CPU2 has an operating frequency of 700MHz.

Given the following frequencies of occurrence of the instructions for the two CPUs

Operation type	Frequency	CPU1 CYCLE	CPU2 CYCLE
A	0.3	2	2
B	0.1	3	3
C	0.2	4	3
D	0.3	2	2
E	0.1	4	3

Exe 2: Questions

Operation type	Frequency	CPU1 CYCLE	CPU2 CYCLE
A	0.3	2	2
B	0.1	3	3
C	0.2	4	3
D	0.3	2	2
E	0.1	4	3

(amount of time you see that
operation on your program, that is
a percentage of time)

- A. Compute the average CPI for CPU1 and CPU2
- B. Which is the fastest CPU?

$$\text{CPI}(\text{CPU1}): 0.3 \cdot 2 + 0.1 \cdot 3 + 0.2 \cdot 4 + 0.3 \cdot 2 + 0.1 \cdot 4 = 2.7$$

$$\text{CPI}(\text{CPU2}): 0.3 \cdot 2 + 0.1 \cdot 3 + 0.2 \cdot 3 + 0.3 \cdot 2 + 0.1 \cdot 3 = 2.4$$

$$\text{Time}(\text{CPUi}) = \text{CPI}(\text{CPUi}) \cdot \text{TimeClockCycle}(\text{CPUi})$$

Exe 2: AVG CPIs

Operation type	Frequency	CPU1 CYCLE	CPU2 CYCLE
A	0.3	2	2
B	0.1	3	3
C	0.2	4	3
D	0.3	2	2
E	0.1	4	3

Recall:
$$CPI = \frac{\text{Clock cycles}}{\text{Instruction}}$$

$$CPI = \sum_{i=1}^n CPI_i * F_i \quad \text{where} \quad F_i = \frac{I_i}{\text{Instruction Count}} \quad \begin{array}{l} \text{(num of instructions of type i)} \\ \text{(tot num of instruct)} \end{array}$$

In case there is the need for compute the frequency of the instructions

Exe 2.b: Which is the fastest CPU?

Exe 2.b: Which is the fastest CPU?

Recall: "X is n times faster than Y" means

$$\frac{Performance(X)}{Performance(Y)} = \frac{Exe(Y)}{Exe(X)}$$

$$\text{CPU time} = \left(\sum_{i=1}^n IC_i \times CPI_i \right) \times \text{Clock cycle time}$$



Exe 3: Amdahl's Law --> one of the most important performance laws

Recall Given an application →

$$Speedup_{overall} = \frac{Executiontime_{old}}{Executiontime_{new}} = \frac{1}{(1 - Fraction_{enhanced} + \frac{Fraction_{enhanced}}{Speedup_{enhanced}})}$$

Best you could ever hope to do:

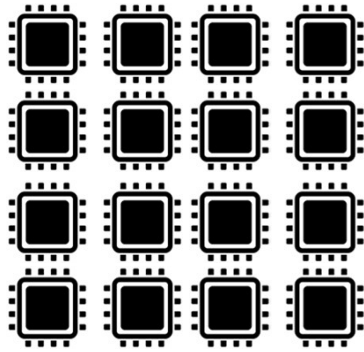
$$Speedup_{overall} = \frac{1}{(1 - Fraction_{enhanced})}$$



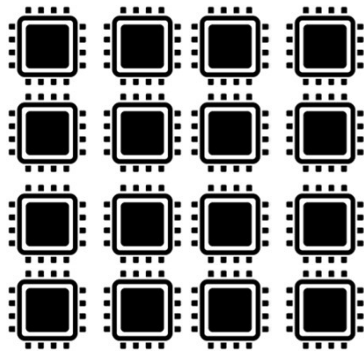
Example of good trade-off analysis considering also costs:

[In-Datcenter Performance Analysis of a Tensor Processing Unit](#)

Exe 3 : Amdahl's Law



Exe 3 : Amdahl's Law



Exe 3 : Amdahl's Law

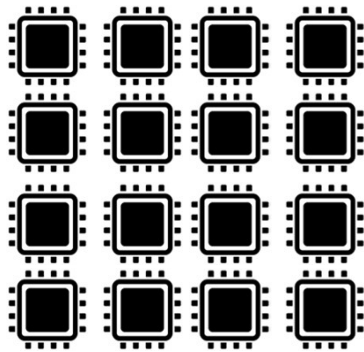


Image Processing task on FPGA is $2.86x^{[1]}$ times faster
100W (TDP) vs 30.85 W

[1] Conficconi, D., D'Amese, E., Del Sozzo, E., Sciuto, D., & Santambrogio, M. D. "A framework for customizable fpga-based image registration accelerators." The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. 2021.

Exe 3 : Questions

- A. With what percentage of processing will adding FPGA result in a speedup of 2?

- B. (At home, if you want, Enjoy :D) Draw a graph that plots the speedup as a percentage of the computation spent performing the image processing task. Label y-axis *< Net speedup >* and x-axis *< Percent image processing >*

Exe 3 : Questions

- A. With what **percentage of processing** will adding FPGA result in a **speedup of 2**?
- B. (At home, if you want, Enjoy :D) Draw a graph that plots the speedup as a percentage of the computation spent performing the image processing task. Label y-axis < Net speedup > and x-axis < Percent image processing >

Exe 3.a : Find the processing fraction

Exe 3: Questions

- A. With what percentage of processing will adding FPGA result in a speedup of 2?
- B. (At home, if you want, Enjoy :D) Draw a graph that plots the speedup as a percentage of the computation spent performing the image processing task. Label y-axis *< Net speedup >* and x-axis *< Percent image processing >*

Exe 3.b : Graph solution

Performance Scaling

How does the **overall performance scale** if we further **increase** the **speedup**?

Let's consider an application where the **number** of used **processors/threads linearly increases** the **performance** of the parallelizable portion

Modern Problems Require Modern Formulae

$$Speedup_{overall} = \frac{1}{(1 - Fraction_{enhanced} + \frac{Fraction_{enhanced}}{Speedup_{enhanced}})}$$

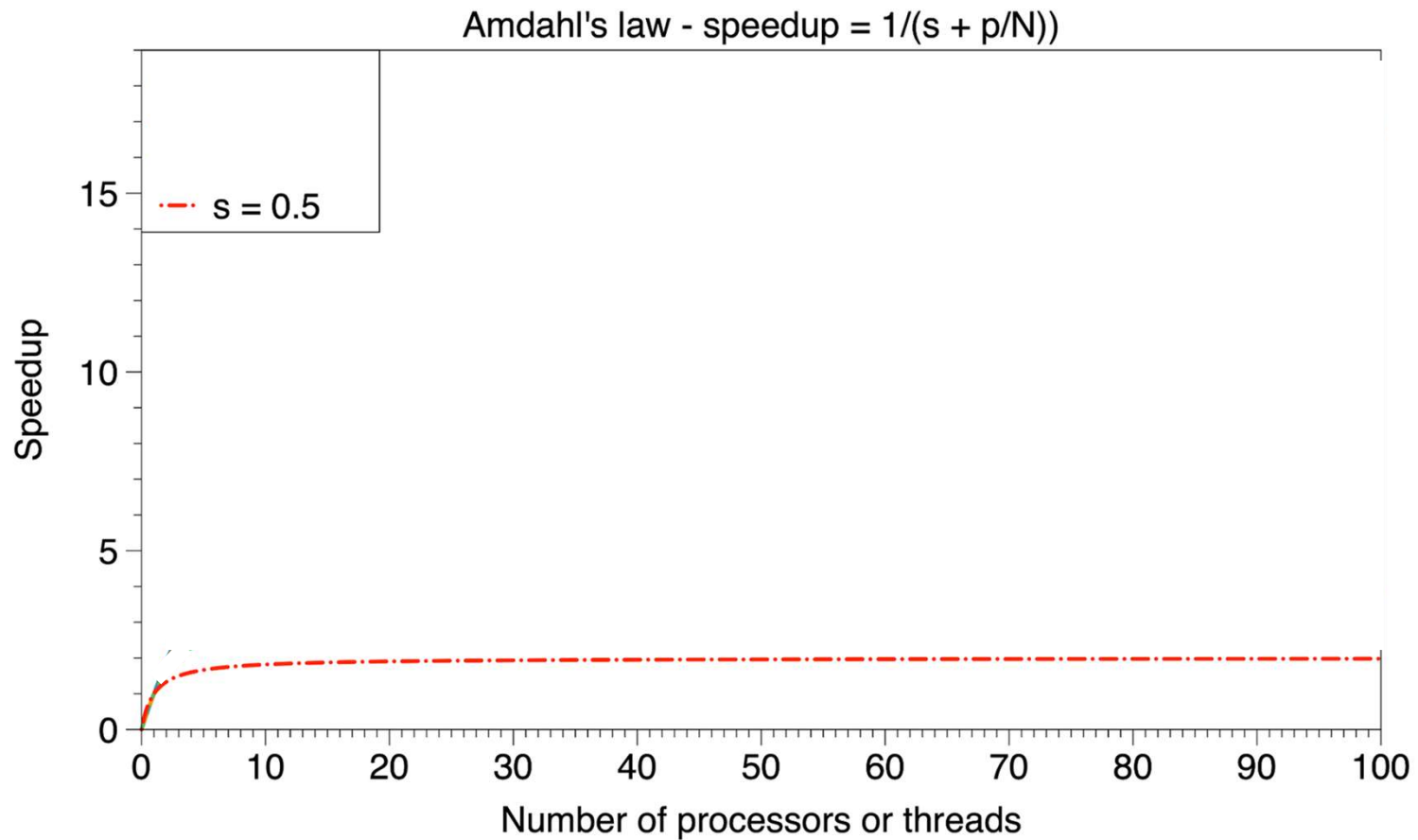
$$speedup_{overall} = 1/(s + p/N)$$

s = serial part = $1 - Fraction_{enhanced}$

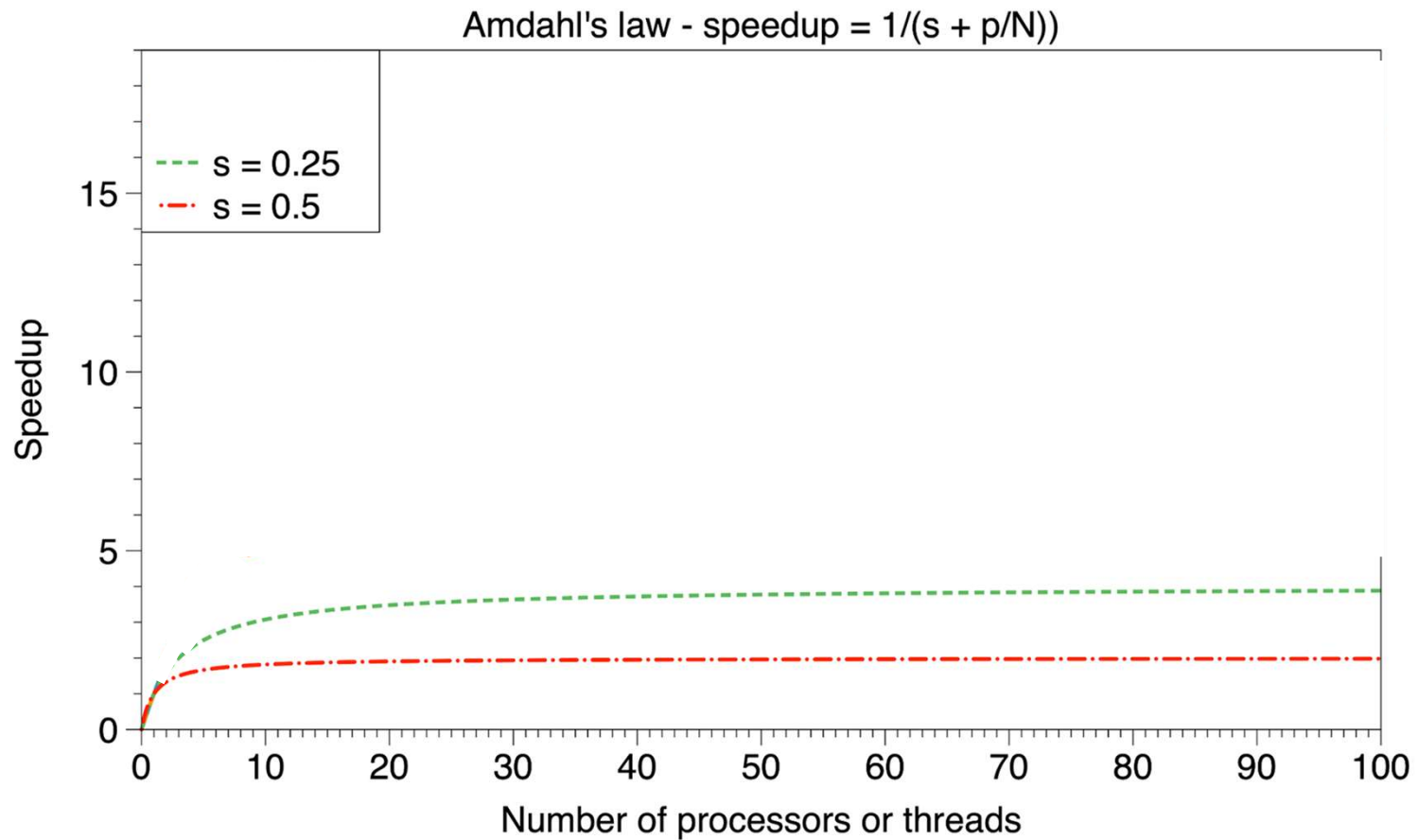
$p = 1 - s$ = parallelizable part = $Fraction_{enhanced}$

N = number of processors or threads = $Speedup_{enhanced}$

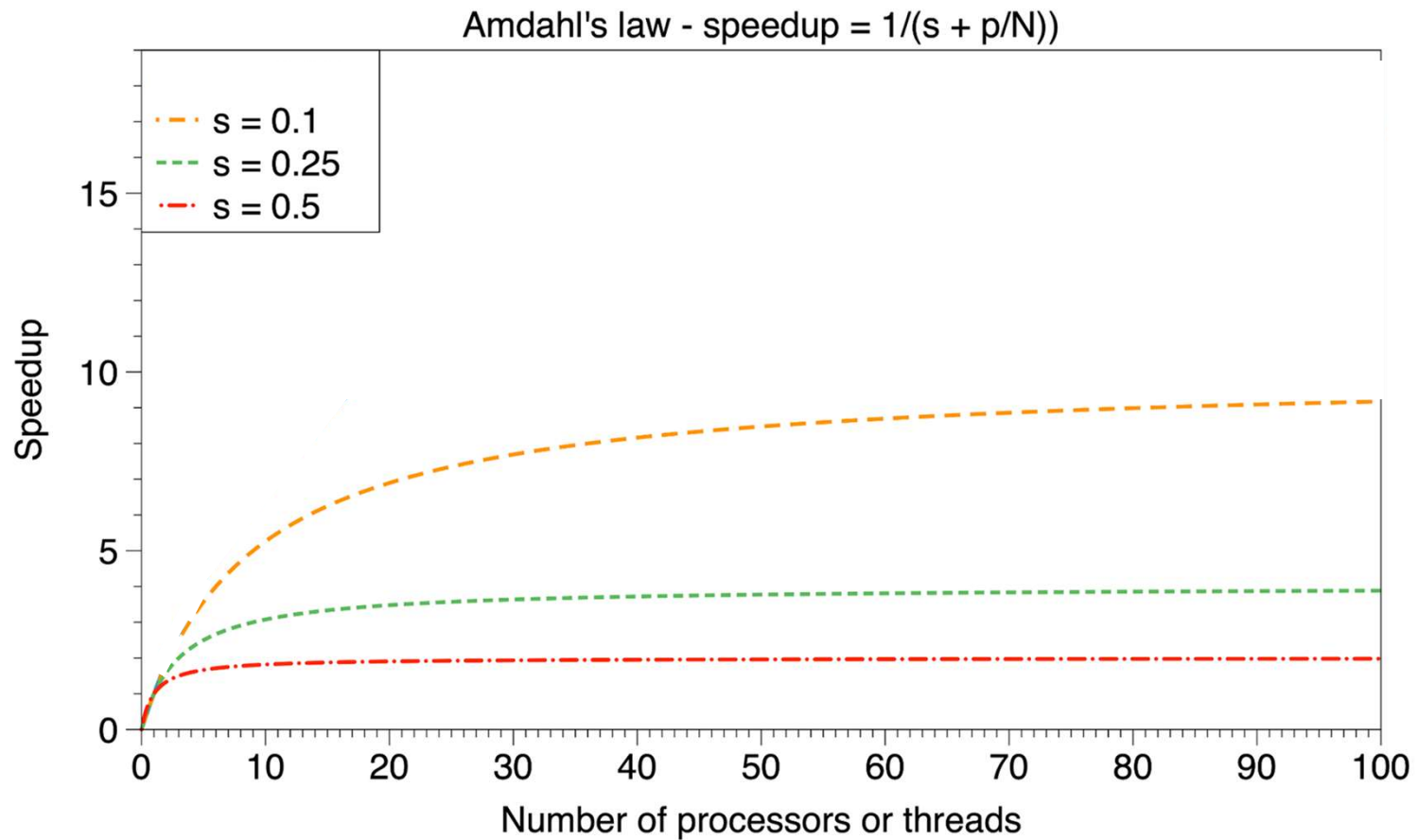
Speedup Scaling vs N/s Scaling



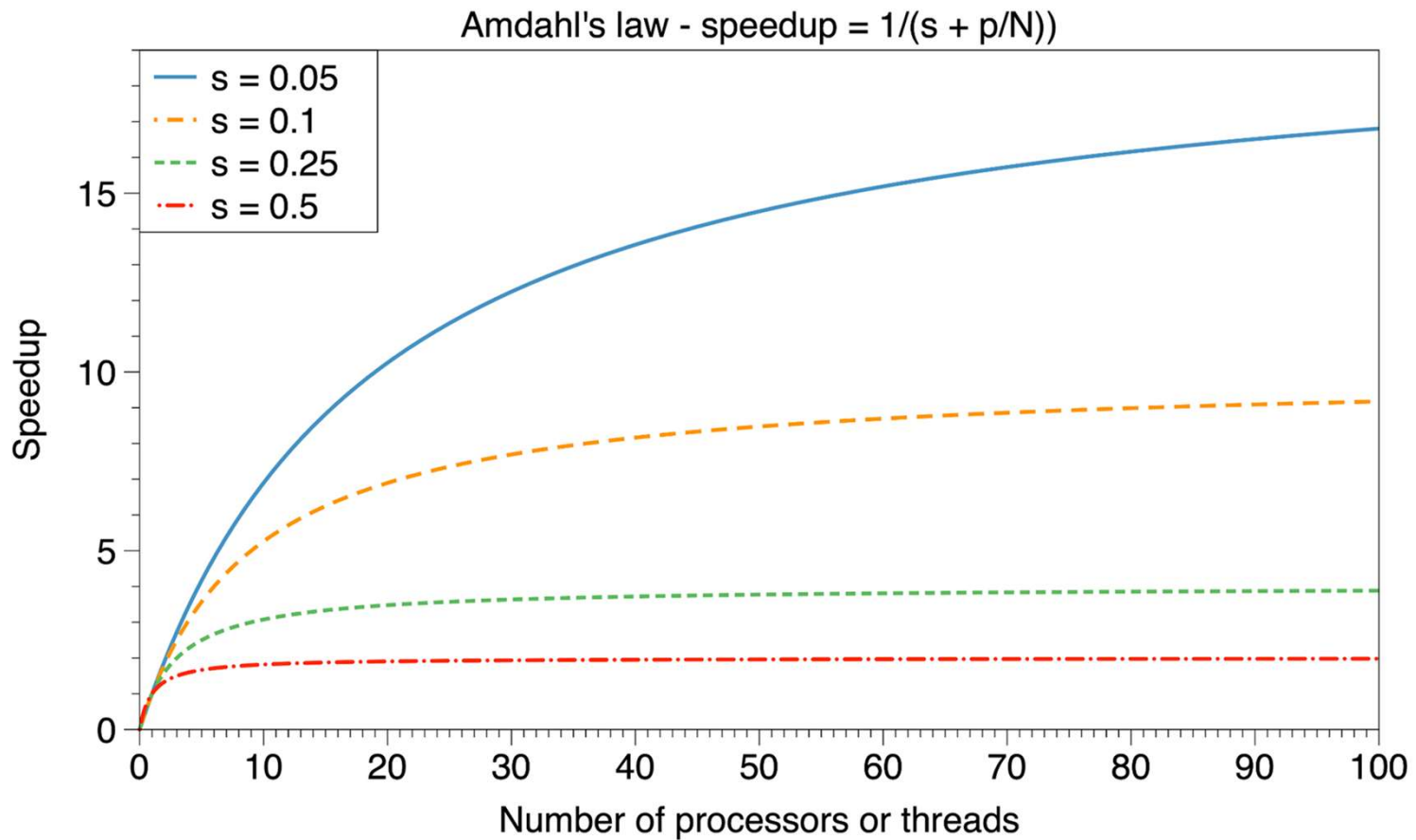
Speedup Scaling vs N/s Scaling



Speedup Scaling vs N/s Scaling



Speedup Scaling vs N/s Scaling



To Sum Up

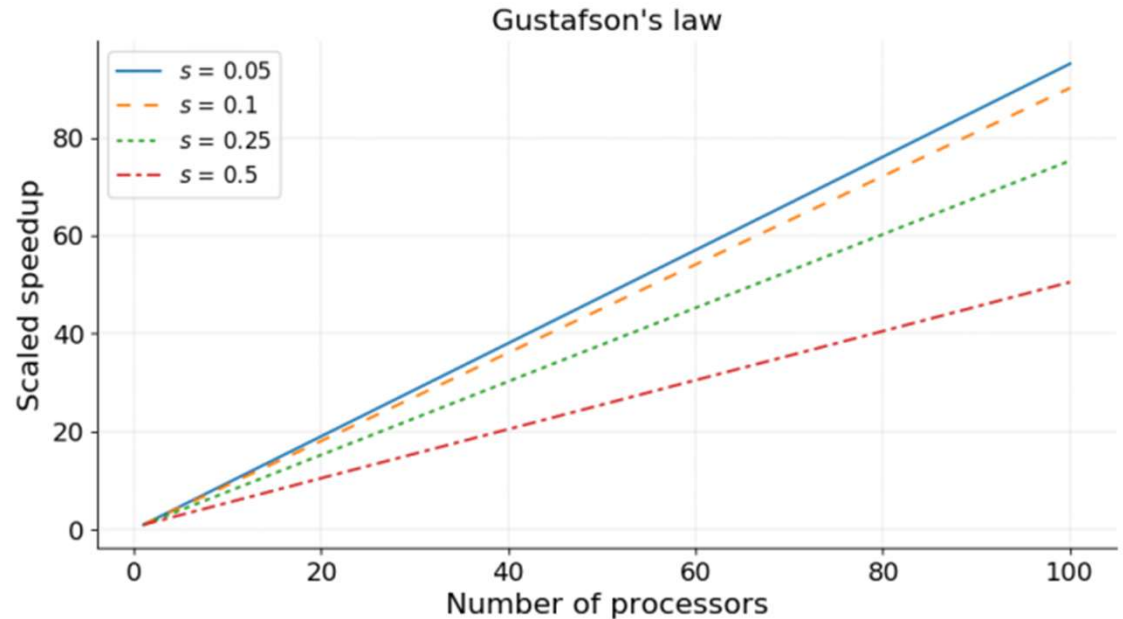
Amdahl's law states that, for a **fixed problem size**, the upper limit of speedup is determined by the serial fraction of the code → **strong scaling**

$$speedup = 1/(s + p/N)$$

Something More about Performance Scaling

Gustafson's law → weak scaling (problems that scales with size)

$$\text{scaled speedup} = s + p \times N$$



If you want more information **JUST FOR CURIOSITY**:

- <https://www.kth.se/blogs/pdc/2018/11/scalability-strong-and-weak-scaling/>
- <https://dl.acm.org/doi/10.1145/42411.42415>



Exe 4 : Pipelining and Performance

Recall and some ref

Web simulators:

(Not tested) [MIPS simulator from unisi](#) and [RISC-V simulator from unisi](#)

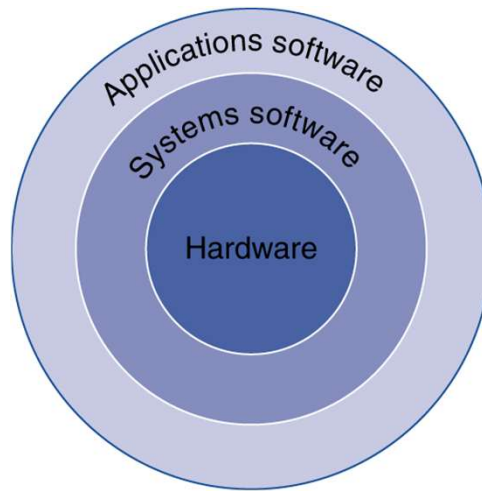
<https://tinyurl.com/aca-grid25>

<https://docs.google.com/spreadsheets/d/1vqTGDF7TNgOfmMDSYCP2vqisW3uBdiuXvvZo0hvc4tA/edit?usp=sharing>



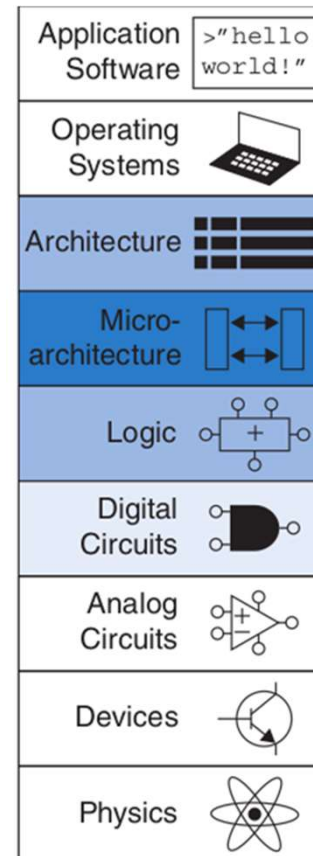
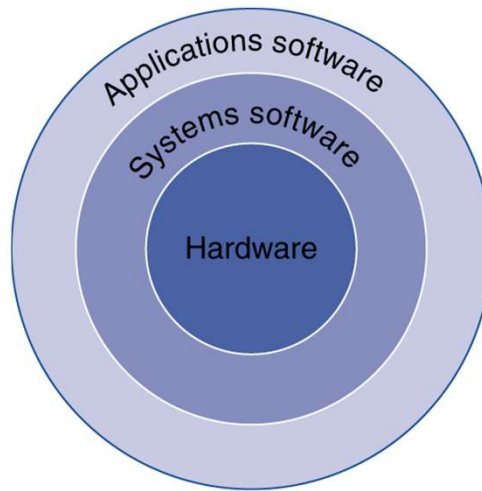
Recall and some ref

<https://tinyurl.com/aca-grid25>

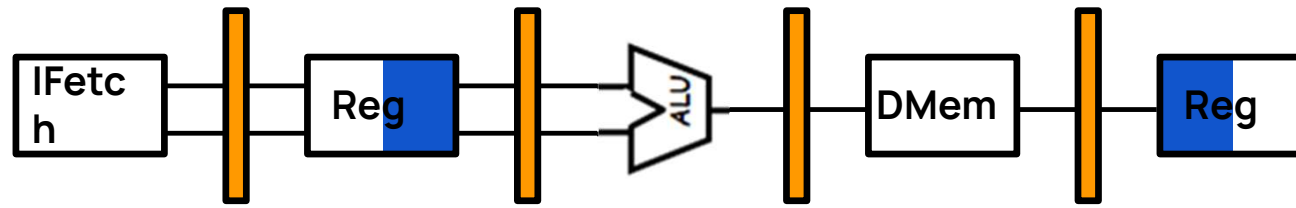


Recall and some ref

<https://tinyurl.com/aca-grid25>

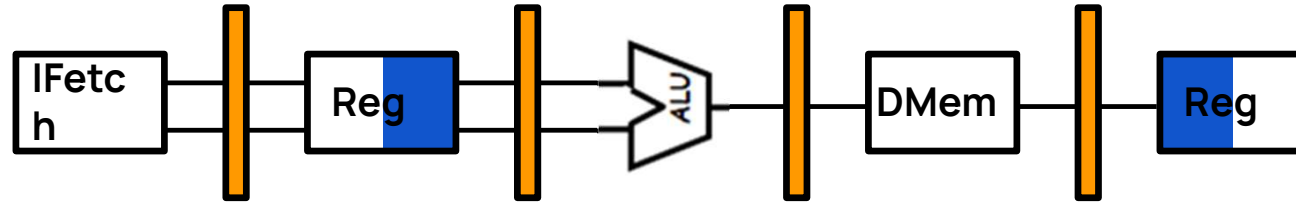


Exe 4 : Pipelining and Performance



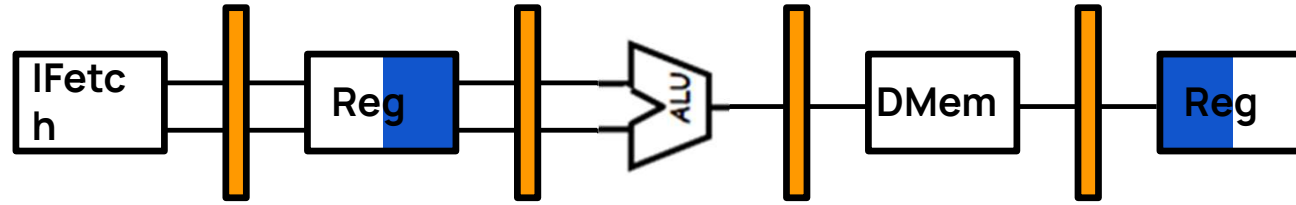
only integer computations!

Exe 4 : Pipelining and Performance



lw \$1, OFF(\$2)
addi \$3, \$1, 4
sub \$4, \$1, \$2
addi \$2, \$1, -8
sw \$4, OFF(\$2)

Exe 4 : Pipelining and Performance



lw \$1, OFF(\$2)
addi \$3, \$1, 4
sub \$4, \$1, \$2
addi \$2, \$1, -8
sw \$4, OFF(\$2)

store doesn't write in the register file
thus the writeback stage is useless for it

IF	ID	EX	ME	WB
Instruction Fetch	Instruction Decode	Execution	Memory Access	Write Back

ALU Instructions: **op \$x, \$y, \$z**

Instr. Fetch & PC Increm.	Read of Source Regs. \$y and \$z	ALU Op. (\$y op \$z)		Write Back Destinat. Reg. \$x
---------------------------	----------------------------------	----------------------	--	-------------------------------

Load Instructions: **lw \$x, offset(\$y)**

Instr. Fetch & PC Increm.	Read of Base Reg. \$y	ALU Op. (\$y+offset)	Read Mem. M(\$y+offset)	Write Back Destinat. Reg. \$x
---------------------------	-----------------------	----------------------	-------------------------	-------------------------------

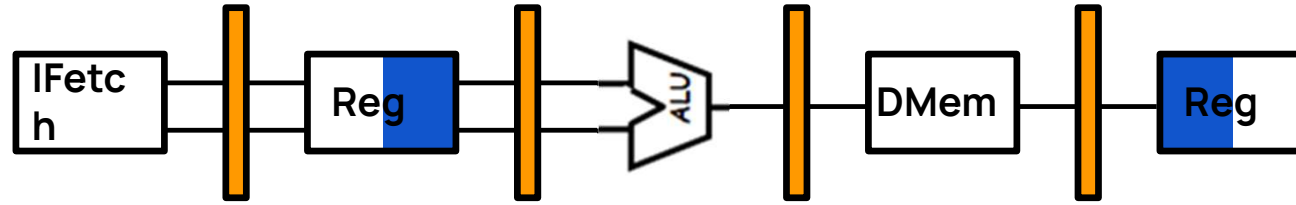
Store Instructions: **sw \$x, offset(\$y)**

Instr. Fetch & PC Increm.	Read of Base Reg. \$y & Source \$x	ALU Op. (\$y+offset)	Write Mem. M(\$y+offset)	
---------------------------	------------------------------------	----------------------	--------------------------	--

Conditional Branches: **beq \$x, \$y, offset**

Instr. Fetch & PC Increm.	Read of Source Regs. \$x and \$y	ALU Op. (\$x-\$y) & (PC+4+offset)	Write PC	
---------------------------	----------------------------------	-----------------------------------	----------	--

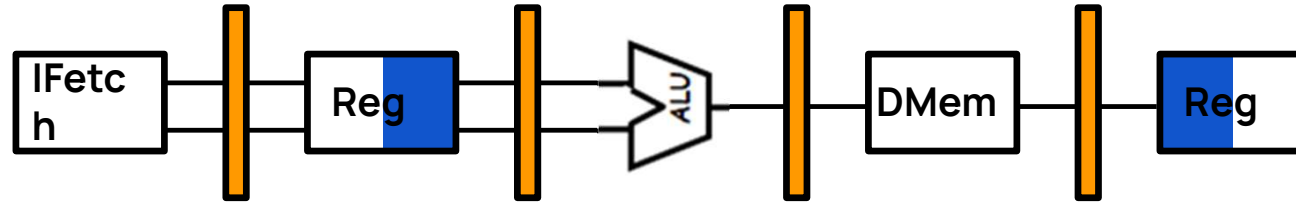
Exe 4 : Pipelining and Performance



lw \$1, OFF(\$2)
addi \$3, \$1, 4
sub \$4, \$1, \$2
addi \$2, \$1, -8
sw \$4, OFF(\$2)

No optimization in the **MIPS** pipeline (e.g., forwarding paths) just our “optimization” (i.e., RF access R/W)

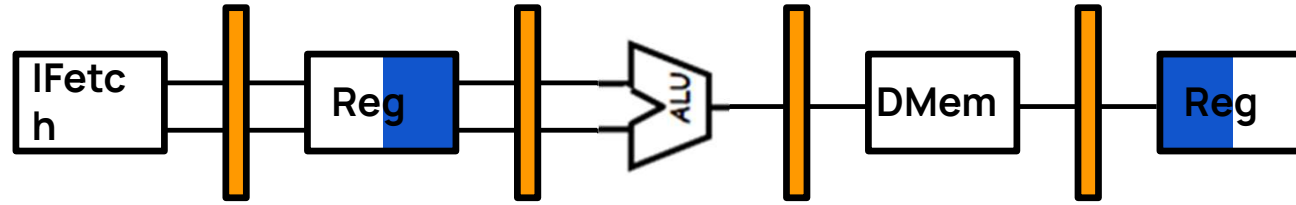
Exe 4 : Pipelining and Performance



lw \$1, OFF(\$2)
addi \$3, \$1, 4
sub \$4, \$1, \$2
addi \$2, \$1, -8
sw \$4, OFF(\$2)

No optimization in the **MIPS** pipeline (e.g., forwarding paths) just our “optimization” (i.e., RF access R/W)
The processor has a clock cycle of 2ns

Exe 4 : Pipelining and Performance



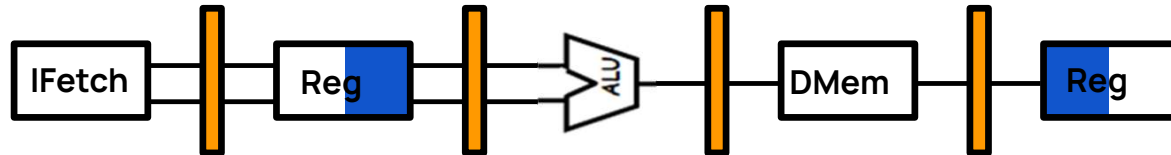
lw \$1, OFF(\$2)
addi \$3, \$1, 4
sub \$4, \$1, \$2
addi \$2, \$1, -8
sw \$4, OFF(\$2)

No optimization in the **MIPS** pipeline (e.g., forwarding paths) just our “optimization” (i.e., RF access R/W)

The processor has a clock cycle of 2ns

- A. Draw the pipeline schema and highlight possible hazards
- B. Represent the real execution (Insert the stalls)
- C. Calculate IC, CPI, MIPS

Exe 4.a : Ideal case




CC 0

Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
lw \$1, OFF(\$2)															
addi \$3, \$1, 4															
sub \$4, \$1, \$3															
addi \$2, \$1, -8															
sw \$5, OFF(\$2)															

Recall: Type of Data Hazard

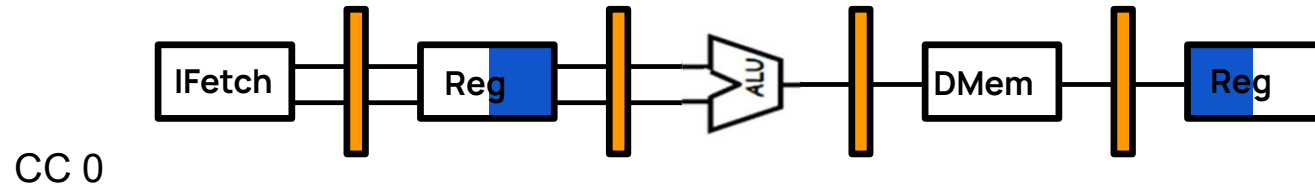
Read After Write (RAW)

Instr_j tries to read operand before Instr_i writes it

 I: add **r1**, r2, r3
J: sub r4, **r1**, r3

Caused by a “[Dependence](#)” (in compiler nomenclature). This hazard results from an actual need for communication.

Exe 4.a : The Hazard



Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
lw \$1, OFF(\$2)															
addi \$3, \$1, 4															
sub \$4, \$1, \$3															
addi \$2, \$1, -8															
sw \$5, OFF(\$2)															

Recall: Data Hazards: Possible Solutions

- Compilation Techniques:
 - Insertion of nop (no operation) instructions
 - Instructions Scheduling to avoid that correlating instructions are too close
 - The compiler tries to insert independent instructions among correlating instructions
 - When the compiler does not find independent instructions, it insert nops.
- Hardware Techniques:
 - Insertion of “bubbles” or stalls in the pipeline
 - Data Forwarding or Bypassing

Exe 4.c : Performance

Calculate **IC**, **CPI**, **MIPS**





POLITECNICO
MILANO 1863

DEPARTMENT OF ELECTRONICS
INFORMATION AND BIOENGINEERING

Thanks for your attention

Davide Conficconi <davide.conficconi@polimi.it>

Acknowledgements

E. Del Sozzo, Marco D. Santambrogio, D. Sciuto

Part of this material comes from:

- Keynotes at HPCA/CGO/PPoPP 2025 from C. Leirson and C. Young
- Hennessy and Patterson [Turing Lecture](#)
- “Computer Organization and Design” and “Computer Architecture A Quantitative Approach” Patterson and Hennessy books
- “Digital Design and Computer Architecture” Harris and Harris

and are **properties of their respective owners**