

Course on: "Advanced Computer Architectures"

# Performance Evaluation



Prof. Cristina Silvano  
Politecnico di Milano  
email: [cristina.silvano@polimi.it](mailto:cristina.silvano@polimi.it)

# **Basic concepts and performance metrics**

# Performance

- Purchasing perspective
  - given a collection of machines, which has the
    - best performance?
    - least cost?
    - best performance / cost?
- Design perspective
  - faced with design options, which has the
    - best performance improvement?
    - least cost?
    - best performance / cost?
- Both require
  - basis for comparison
  - metrics for evaluation
- Our goal is to understand cost & performance implications of architectural choices

# Two notions of “performance”

Plane	DC to Paris	Speed	Passengers	Throughput (p×mph)
Boeing 747	6.5 hours	610 mph	470	286,700
BAD/Sud Concorde	3 hours	1350 mph	132	178,200

## Which has higher performance?

- Time to do the task (Execution Time)
  - Execution time, response time, **latency**
- Number of jobs done per day, hour, sec, ns (Performance)
  - **Throughput**, bandwidth
- Response time and throughput often are in opposition

# Example

- Time of Concorde vs. Boeing 747?
  - Concord is  $1350 \text{ mph} / 610 \text{ mph} = 2.2$  times faster  
= 6.5 hours / 3 hours
- Throughput of Concorde vs. Boeing 747 ?
  - Concord is  $178,200 \text{ pmph} / 286,700 \text{ pmph} = 0.62$  “times faster”
  - Boeing is  $286,700 \text{ pmph} / 178,200 \text{ pmph} = 1.60$  “times faster”
- Boeing is 1.6 times (“60%”) faster in terms of throughput
- Concord is 2.2 times (“120%”) faster in terms of flying time

We will focus primarily on execution time for a single job

Lots of instructions in a program => Instruction throughput important!

# Definitions

- *"X is n% faster than Y"*  $\Rightarrow \frac{\text{execution time (y)}}{\text{execution time (x)}} = 1 + \frac{n}{100}$

$$\text{performance}(x) = \frac{1}{\text{execution\_time}(x)}$$

- *"X is n% faster than Y"*  $\Rightarrow \frac{\text{performance}(x)}{\text{performance}(y)} = 1 + \frac{n}{100}$

# Performance Improvement

- Performance improvement means increment:
  - Higher is better
- Execution time (or response time) improvement means decrement:
  - Lower is better

# Example

If machine A executes a program in 10 sec and machine B executes same program in 15 sec:

*A is 50% faster than B or A is 33% faster than B?*

## Solution:

- The statement A is  $n\%$  faster than B can be expressed as:
- *"A is  $n\%$  faster than B"*  $\Rightarrow \frac{\text{execution time (B)}}{\text{execution time (A)}} = 1 + \frac{n}{100}$

$$\Rightarrow n = \frac{\text{execution time (B)} - \text{execution time (A)}}{\text{execution time (A)}} * 100$$

$$(15 - 10)/10 * 100 = 50 \Rightarrow \mathbf{A \text{ is } 50\% \text{ faster than B.}}$$



# Clock cycles

- $T_{\text{CLK}}$  = Period or clock cycle time [seconds]
- $f_{\text{CLK}}$  = Clock frequency = Clock cycles per second

$$f_{\text{CLK}} = 1 / T_{\text{CLK}}$$

Where: 1 Hz = 1 / sec

- Examples:
  - $f_{\text{CLK}} = 500 \text{ MHz}$  corresponds to  $T_{\text{CLK}} = 1 / (500 * 10^6) = 2 * 10^{-9} = 2 \text{ nsec}$
  - $f_{\text{CLK}} = 1 \text{ GHz}$  corresponds to  $T_{\text{CLK}} = 1 / (10^9) = 1 * 10^{-9} = 1 \text{ nsec}$

# Execution time or CPU Time

$$\text{CPU time} = \text{Clock Cycles} \times T_{\text{CLK}} = \frac{\text{Clock Cycles}}{f_{\text{CLK}}}$$

- To optimize performance means to reduce the execution time (or CPU time):
  - *To reduce the number of clock cycles per program*
  - *To reduce the clock period  $T_{\text{clk}}$*
  - *To increase the clock frequency  $f_{\text{CLK}}$*

# CPU Time

$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \text{IC} \times \text{CPI} \times T_{\text{CLK}} = \frac{\text{IC} \times \text{CPI}}{f_{\text{CLK}}}$$

- Where **Clock Per Instruction** is given by:

$$\text{CPI} = \text{Clock Cycles} / \text{Instruction Count}$$

- The **Instruction Per Clock** is given by: **IPC = 1 / CPI**

# CPU Time

$$\text{CPU time} = \text{Clock Cycles} \times T_{\text{CLK}} = \frac{\text{Clock Cycles}}{f_{\text{CLK}}}$$

$$\text{Where: Clock Cycles} = \sum_{i=1}^n (\text{CPI}_i \times I_i)$$

$$\Rightarrow \text{CPU time} = \sum_{i=1}^n (\text{CPI}_i \times I_i) \times T_{\text{CLK}} \quad \text{thus being CPU time} = \text{IC} \times \text{CPI} \times T_{\text{CLK}}$$

$$\text{CPI} = \sum_{i=1}^n (\text{CPI}_i \times F_i)$$

$$\text{Where } F_i = \frac{I_i}{\text{IC}}$$

"instruction frequency"

$$\Rightarrow \text{CPU time} = \text{IC} \times \text{CPI} \times T_{\text{CLK}} = \text{IC} \times \sum (\text{CPI}_i * F_i) \times T_{\text{CLK}}$$

# Example

	Frequency	Clock Cycles
<b>ALU</b>	43%	1
<b>Load</b>	21%	4
<b>Store</b>	12%	4
<b>Branch</b>	12%	2
<b>Jump</b>	12%	2

- Evaluate the CPI and the CPU time to execute a program composed of 100 instructions mixed as in the table by using 500 MHz clock frequency:

$$\mathbf{CPI} = 0.43 * 1 + 0.21 * 4 + 0.12 * 4 + 0.12 * 2 + 0.12 * 2 = 2.23$$

$$\mathbf{CPU\ time} = IC * CPI * T_{clk} = 100 * 2.23 * 2\text{ ns} = 446\text{ ns}$$

# MIPS Millions of instructions per second

$$\text{MIPS} = \frac{\text{Instruction Count}}{\text{Execution time} \times 10^6}$$

$$\text{Where: Execution time} = \frac{\text{IC} \times \text{CPI}}{f_{\text{CLK}}}$$

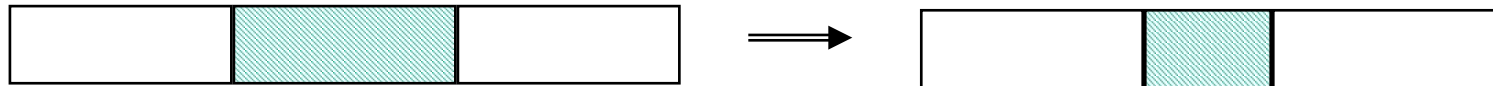
$$\text{MIPS} = \frac{f_{\text{CLK}}}{\text{CPI} \times 10^6}$$

# Amdahl's Law

# How to evaluate the speedup

Speedup due to enhancement E:

$$\text{Speedup}(E) = \frac{\text{ExTime w/o } E}{\text{ExTime w/ } E} = \frac{\text{Performance w/ } E}{\text{Performance w/o } E}$$



Assume that an enhancement E accelerates a fraction F of the execution time of the task by a factor S and the remainder of the task is unaffected then,

$$\text{ExTime}(w/ E) = ((1-F) + F/S) \times \text{ExTime}(w/o E)$$

$$\text{Speedup}(w/ E) = \frac{1}{(1-F) + F/S}$$



# Amdahl's Law

- Basic idea: Make the most common case fast
- **Amdahl's Law:** The performance improvement obtainable from using some faster execution modes is limited by the fraction of the time that the faster mode is used. Let us assume:
  - **Fraction<sub>E</sub>** the fraction of the computation time in the original machine that can be converted to take advantage of the enhancement
  - **Speedup<sub>E</sub>** the improvement gained by the enhanced execution mode
- The overall speed up is given by:

$$\text{Speedup}_{\text{overall}} = \frac{\text{ExTime}_{\text{old}}}{\text{ExTime}_{\text{new}}} = \frac{1}{(1 - \text{Fraction}_E) + \frac{\text{Fraction}_E}{\text{Speedup}_E}}$$

# Example

- Let us consider an enhancement for a CPU resulting 10 time faster on computation than the original one but the original CPU is busy with computation only 40% of the time.

What is the overall speedup gained by introducing the enhancement?

- Solution:** Application of Amdahl's Law where:
  - Fraction<sub>E</sub>** = 0.4
  - Speedup<sub>E</sub>** = 10
- The overall speed up is given by:

$$\text{Speedup}_{\text{overall}} = \frac{1}{(1 - \text{Fraction}_E) + \frac{\text{Fraction}_E}{\text{Speedup}_E}} = \frac{1}{0.6 + 0.04} = 1.56$$

# Basis of Evaluation

## Pros

- representative
- portable
- widely used
- improvements useful in practice
- easy to run, early in design cycle
- identify peak capability and potential bottlenecks

Actual Target Workload

Full Application Benchmarks

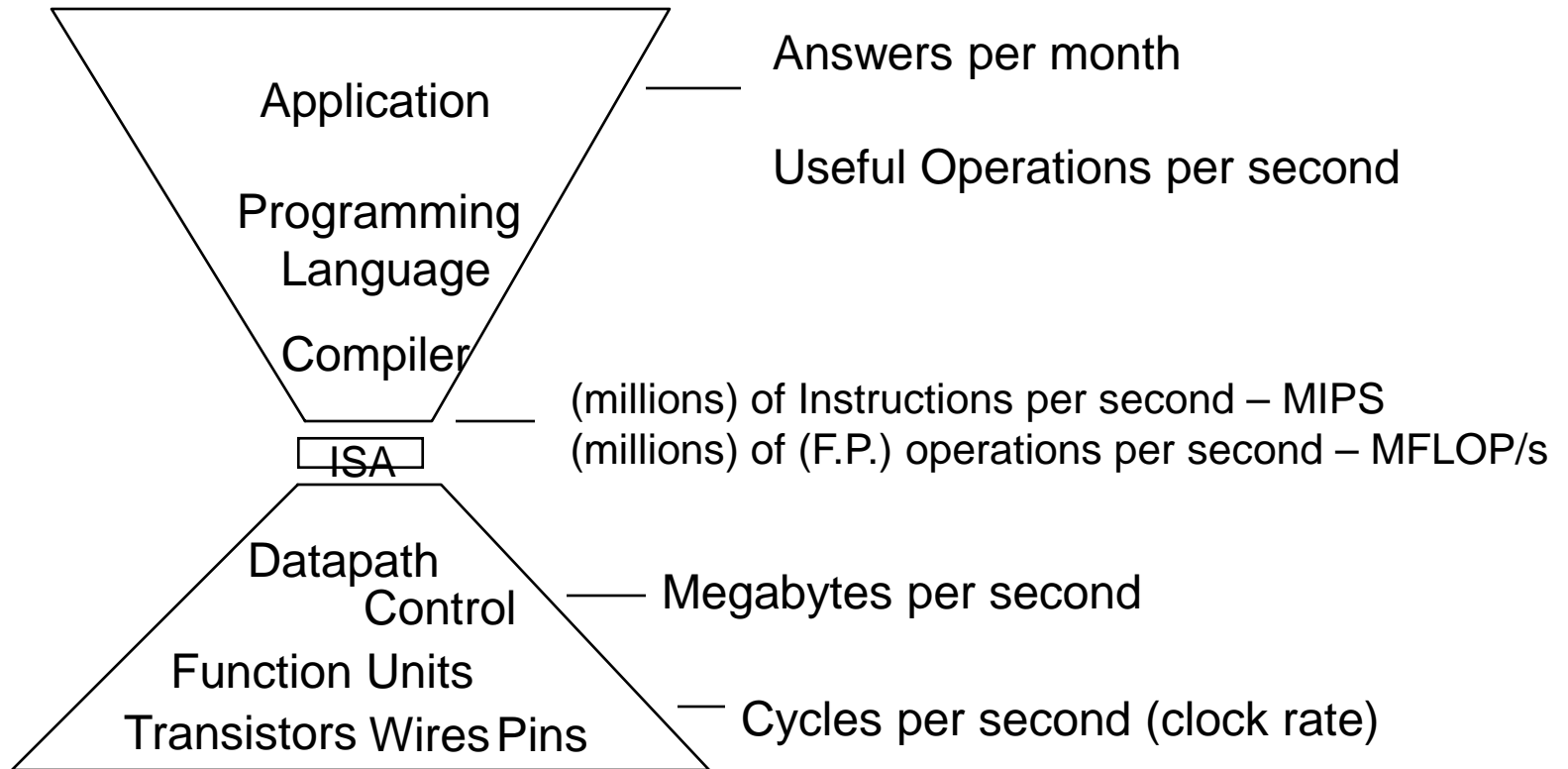
Small “Kernel”  
Benchmarks

Microbenchmarks

## Cons

- very specific
- non-portable
- difficult to run, or measure
- hard to identify cause
- less representative
- easy to “fool”
- “peak” may be a long way from application performance

# Metrics of performance



Each metric has a place and a purpose, and each can be misused

# Aspects of CPU Performance

$$\text{CPU time} = \text{IC} \times \text{CPI} \times T_{\text{CLK}}$$

	instr count	CPI	clock rate
Program	X		
Compiler	X	X	
Instr. Set	X	X	X
Organization (e.g., datapath or pipeline)		X	X
Technology			X

# **Performance evaluation in pipelined processors**

# Performance Issues in Pipelining

Pipelining increases the CPU instruction **throughput** (number of instructions completed per unit of time), but it does not reduce the execution time (latency) of a single instruction

- Pipelining usually slightly increases the latency of each instruction due to the imbalance among the pipeline stages and the overhead in the control of the pipeline.
  - Imbalance among pipeline stages reduces performance since the clock period cannot be shorter than the time needed for the slowest pipe stage.
  - Pipeline overhead arises from pipeline register delay and clock skew.
  - All instructions should go through the same number of pipeline stages

# Performance Metrics

**IC** = Instruction Count

**CPI** = Cycles Per Instruction

**IPC** = Instructions Per Clock =  $1 / \text{CPI}$

**# Clock Cycles** =  $\text{IC} + \# \text{ Stall Cycles} + 4$  for a 5-stage pipeline as MIPS or RISC-V

**CPI** =  $\# \text{ Clock Cycles} / \text{IC} = (\text{IC} + \# \text{ Stall Cycles} + 4) / \text{IC}$

**MIPS** =  $f_{\text{clock}} / (\text{CPI} * 10^6)$



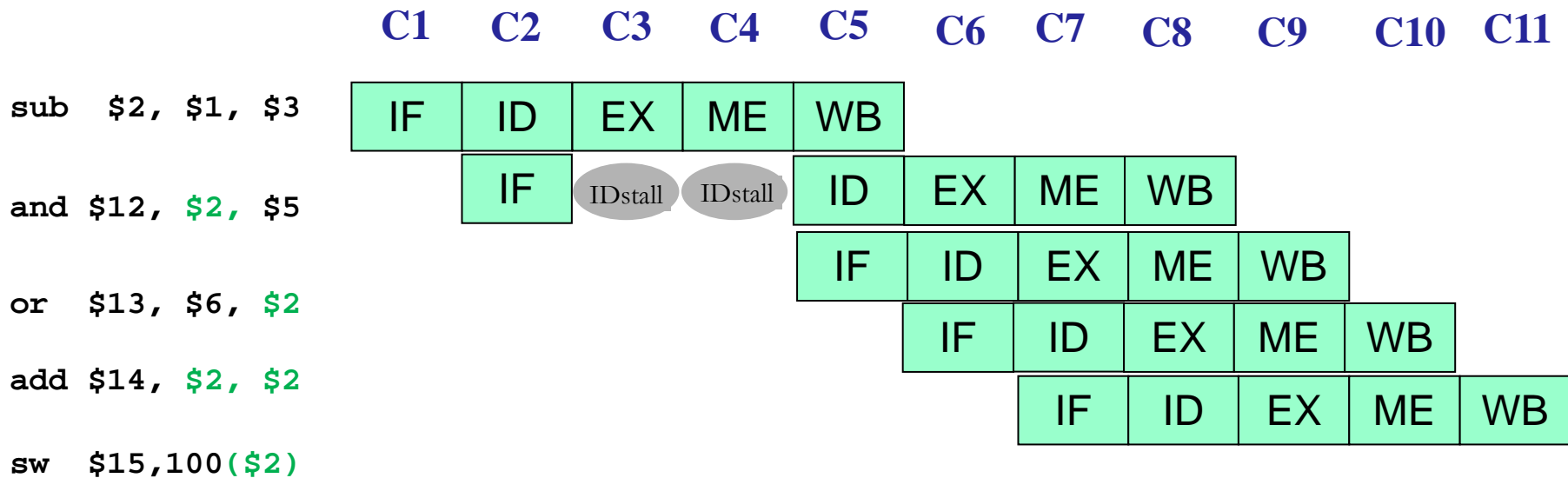
# Example

**IC** = Instruction Count = 5

**# Clock Cycles** = IC + # Stall Cycles + 4 = 5 + 2 + 4 = 11

**CPI** = Clock Per Instruction = # Clock Cycles / IC = 11 / 5 = 2.2

**MIPS** =  $f_{\text{clock}} / (\text{CPI} * 10^6) = 500 \text{ MHz} / (2.2 * 10^6) = 227$



## Performance Metrics (2)

- Let us consider **n** iterations of a loop composed of **m** instructions per iteration requiring **k** stalls per iteration

$$\mathbf{IC}_{\text{per\_iter}} = m$$

$$\mathbf{\# \text{ Clock Cycles}}_{\text{per iter}} = \mathbf{IC}_{\text{per\_iter}} + \mathbf{\# \text{ Stall Cycles}}_{\text{per\_iter}} + 4$$

$$\begin{aligned}\mathbf{CPI}_{\text{per\_iter}} &= (\mathbf{IC}_{\text{per iter}} + \mathbf{\# \text{ Stall Cycles}}_{\text{per\_iter}} + 4) / \mathbf{IC}_{\text{per\_iter}} \\ &= (m + k + 4) / m\end{aligned}$$

$$\mathbf{MIPS}_{\text{per\_iter}} = f_{\text{clock}} / (\mathbf{CPI}_{\text{per\_iter}} * 10^6)$$

# Asymptotic Performance Metrics

- Let us consider **n** iterations of a loop composed of **m** instructions per iteration requiring **k** stalls per iteration:

$$\mathbf{IC}_{AS} = \text{Instruction Count}_{AS} = m * n$$

$$\mathbf{\# \text{ Clock Cycles}} = \text{IC}_{AS} + \# \text{ Stall Cycles}_{AS} + 4$$

$$\begin{aligned}\mathbf{CPI}_{AS} &= \lim_{n \rightarrow \infty} ( \text{IC}_{AS} + \# \text{ Stall Cycles}_{AS} + 4 ) / \text{IC}_{AS} \\ &= \lim_{n \rightarrow \infty} ( m * n + k * n + 4 ) / m * n \\ &= (m + k) / m\end{aligned}$$

$$\mathbf{MIPS}_{AS} = f_{\text{clock}} / (\text{CPI}_{AS} * 10^6)$$

# Performance Issues in Pipelining

- The *ideal CPI* on a pipelined processor would be *1*, but stalls cause the pipeline performance to degrade from the ideal performance, so we have:

$$\begin{aligned}\text{Ave. CPI Pipe} &= \text{Ideal CPI} + \text{Stall Cycles per Instruction} \\ &= 1 + \text{Stall Cycles per Instruction}\end{aligned}$$

- **Stall Cycles per Instruction** are due to:

Structural Hazards + Data Hazards + Control Hazards + Memory Stalls

# Performance Issues in Pipelining

$$\text{Pipeline Speedup} = \frac{\text{Avg. Exec. Time Unpipelined}}{\text{Avg. Exec. Time Pipelined}} =$$

$$= \frac{\text{Avg. CPI}_{\text{Unp}} \times T_{\text{clk,unp}}}{\text{Avg. CPI}_{\text{Pipe}} \times T_{\text{clk,pipe}}}$$

# Performance Issues in Pipelining

- If we ignore the cycle time overhead of pipelining and we assume the stages are perfectly balanced, the clock cycle time of unpipelined/pipelined processors can be equal, so: (Speedup =  $CPI_{Unp} / CPI_{Pipe}$ ,  $CPI_{Pipe} = 1 + \text{Stall cycles per instr.}$ )

$$\text{Pipeline Speedup} = \frac{\text{Avg. CPI Unp.}}{1 + \text{Stall Cycles per Instruction}}$$

- If we assume that each instruction takes the same number of cycles, which must be equal to the # of pipeline stages (called pipeline depth):

$$\text{Pipeline Speedup} = \frac{\text{Pipeline Depth}}{1 + \text{Stall Cycles per Instruction}}$$

If there were no pipeline stalls (ideal case), this leads to the intuitive result that pipelining improves performance by the depth of the pipeline:

*Pipeline Speedup = Pipeline Depth*

# Performance of Branch Schemes

- What is the performance impact of conditional branches?

$$\begin{aligned}\text{Pipeline Speedup} &= \frac{\text{Pipeline Depth}}{1 + \text{Pipe Stall Cycles per Instruction due to Branches}} \\ &= \frac{\text{Pipeline Depth}}{1 + \text{Branch Frequency} \times \text{Branch Penalty}}\end{aligned}$$

# **Performance evaluation of the memory hierarchy**



# Memory Hierarchy: Definitions

In a stack of memories, the one on the lowest level is the slowest one (likely it is also the biggest, e.g., hard disk), while the one on the highest level (the topmost) is the fastest one (and also the smallest, e.g., a L1 cache, or a CPU register)

- **Hit:** data found in a block of the upper level
- **Hit Rate:** Number of memory accesses finding data in the upper level memory with respect to the total number of memory accesses:

$$\text{Hit Rate} = \frac{\text{\# hits}}{\text{\# memory accesses}}$$

- **Hit Time:** time to access the data in the upper level of the hierarchy, including the time needed to decide if the attempt of access will result in a hit or miss

# Memory Hierarchy: Definitions

- **Miss:** the data must be taken from the lower level
- **Miss Rate:** number of memory accesses not finding data in the upper level with respect to the total number of memory accesses:

$$\text{Miss Rate} = \frac{\# \text{ misses}}{\# \text{ memory accesses}}$$

- By definition: **Hit Rate + Miss Rate = 1**
- **Miss Time = Hit Time + Miss Penalty**  
**Miss Penalty** is the time needed to access the lower level and to replace the block in the upper level  
Typically, we have that: **Hit Time << Miss Penalty**

# Average Memory Access Time (AMAT)

$$\text{AMAT} = \text{Hit Rate} * \text{Hit Time} + \text{Miss Rate} * \text{Miss Time}$$

Being:  $\text{Miss Time} = \text{Hit Time} + \text{Miss Penalty}$

$$\Rightarrow \text{AMAT} = \text{Hit Rate} * \text{Hit Time} + \text{Miss Rate} * (\text{Hit Time} + \text{Miss Penalty})$$

$$\Rightarrow \text{AMAT} = (\text{Hit Rate} + \text{Miss Rate}) * \text{Hit Time} + \text{Miss Rate} * \text{Miss Penalty}$$

By definition:  $\text{Hit Rate} + \text{Miss Rate} = 1$

$$\Rightarrow \text{AMAT} = \text{Hit Time} + \text{Miss Rate} * \text{Miss Penalty}$$

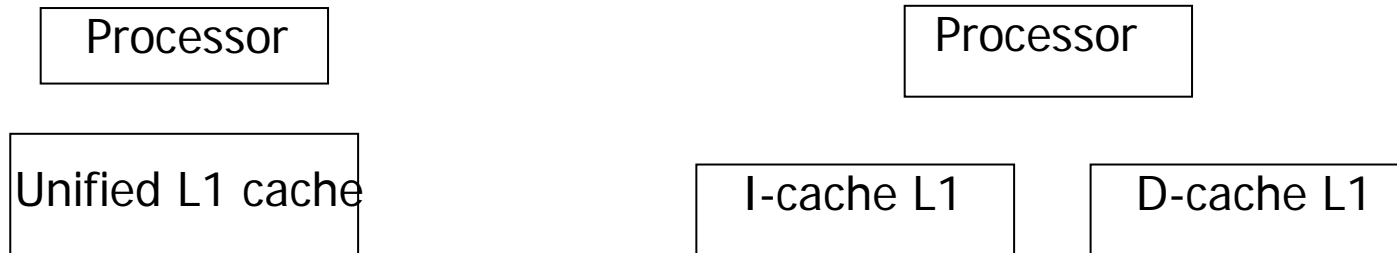
# Cache Performance

- Average Memory Access Time:

$$\text{AMAT} = \text{Hit Time} + \text{Miss Rate} * \text{Miss Penalty}$$

- *How to improve cache performance?*
  1. Reduce the hit time
  2. Reduce the miss rate
  3. Reduce the miss penalty

# Unified Cache vs Separate I\$ & D\$ (Harvard architecture)



*To better exploit the locality principle*

- For separate I\$ & D\$ (Harvard architecture):

$$\mathbf{AMAT}_{\text{Harvard}} = \% \text{ Instr. (Hit Time + Miss Rate I\$ * Miss Penalty)} + \\ \% \text{ Data (Hit Time + Miss Rate D\$ * Miss Penalty)}$$

- Usually: Miss Rate I\$ << Miss Rate D\$

# Unified Cache vs Separate I\$ & D\$: Example of comparison

## Assumptions:

- Harvard: 16KB I\$                      16KB D\$:  
I\$ Miss Rate=0.64%                  D\$ Miss Rate=6.47%
- 32KB Unified: Aggregate Miss Rate=1.99%
- 33% loads/stores (data ops)  
⇒ 75% accesses from instructions (1.0/1.33)  
⇒ 25% accesses from data (0.33/1.33)
- Hit time=1, Miss Penalty = 50
- Note: *data* hit has 1 more stall for unified cache (only one port)

## Which cache is better?

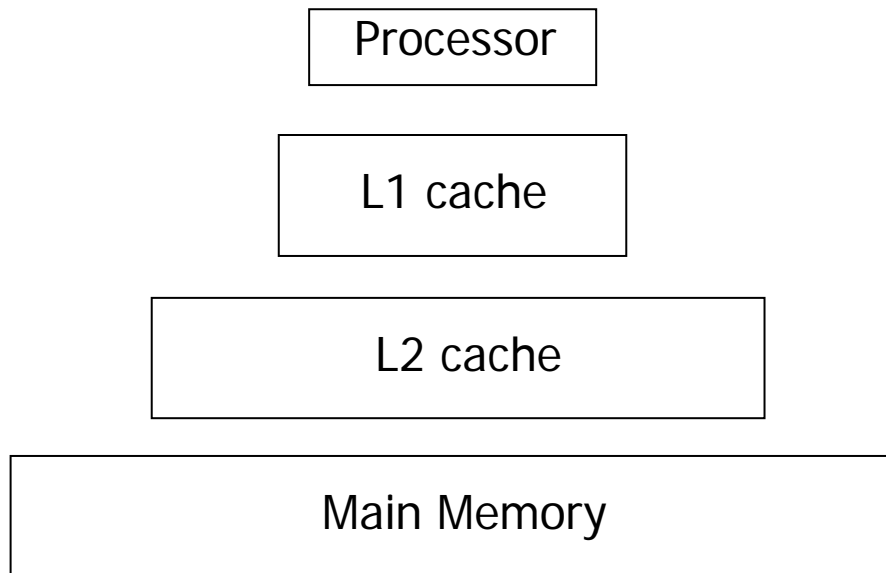
$$AMAT_{\text{Harvard}} = 75\% \times (1 + 0.64\% \times 50) + 25\% \times (1 + 6.47\% \times 50) = 2.05$$

$$AMAT_{\text{Unified}} = 75\% \times (1 + 1.99\% \times 50) + 25\% \times (1 + 1 + 1.99\% \times 50) = 2.24$$

# Miss Penalty Reduction: Second Level Cache

## Basic Idea:

- L1 cache small enough to match the fast CPU cycle time
- L2 cache large enough to capture many accesses that would go to main memory reducing the effective miss penalty



# AMAT for L1 and L2 Caches

$$\mathbf{AMAT} = \text{Hit Time}_{L1} + \text{Miss Rate}_{L1} \times \text{Miss Penalty}_{L1}$$

$$\text{where: } \text{Miss Penalty}_{L1} = \text{Hit Time}_{L2} + \text{Miss Rate}_{L2} \times \text{Miss Penalty}_{L2}$$

$$\Rightarrow \mathbf{AMAT} = \text{Hit Time}_{L1} + \text{Miss Rate}_{L1} \times (\text{Hit Time}_{L2} + \text{Miss Rate}_{L2} \times \text{Miss Penalty}_{L2})$$

$$\Rightarrow \mathbf{AMAT} = \text{Hit Time}_{L1} + \text{Miss Rate}_{L1} \times \text{Hit Time}_{L2} + \mathbf{\text{Miss Rate}_{L1 L2}} \times \text{Miss Penalty}_{L2}$$



# Local and global miss rates

Definitions:

- ***Local miss rate***: misses in this cache divided by the total number of memory accesses *to this cache*: the  $\text{Miss rate}_{L1}$  for L1 and the  $\text{Miss rate}_{L2}$  for L2
- ***Global miss rate***: misses in this cache divided by the total number of memory accesses generated by the CPU:
  - for L1, the global miss rate is still just  $\text{Miss Rate}_{L1}$
  - for L2, it is  $(\text{Miss Rate}_{L1} \times \text{Miss Rate}_{L2})$
- **Global miss rate** is what really matters: it indicates what fraction of memory accesses from CPU go all the way to main memory

# Example

- Let us consider a computer with a L1 cache and L2 cache memory hierarchy. Suppose that in 1000 memory references there are 40 misses in L1 and 20 misses in L2.
- *What are the various miss rates?*

**Miss Rate  $_{L1}$  = 40 / 1000 = 4% (either local or global)**

**Miss Rate  $_{L2}$  = 20 / 40 = 50%**

**Global Miss Rate for Last Level Cache (L2):**

**Miss Rate  $_{L1 L2}$  = Miss Rate $_{L1}$  x Miss Rate $_{L2}$  = 2%**

# Impact of memory hierarchy on CPU<sub>time</sub>

$$\text{CPU}_{\text{time}} = (\text{CPU}_{\text{exec}} \text{ cycles} + \text{Memory stall cycles}) \times T_{\text{CLK}}$$

where:  $T_{\text{CLK}} = T$  clock cycle time period

$$\text{CPU}_{\text{exec}} \text{ cycles} = \text{IC} \times \text{CPI}_{\text{exec}} \quad \text{where: IC} = \text{Instruction Count}$$

( $\text{CPI}_{\text{exec}}$  includes both ALU and LD/STORE instructions)

$$\text{Memory stall cycles} = \text{IC} \times \text{Misses per Instr} \times \text{Miss Penalty}$$

$$\Rightarrow \text{CPU}_{\text{time}} = \text{IC} \times (\text{CPI}_{\text{exec}} + \text{Misses per Instr} \times \text{Miss Penalty}) \times T_{\text{CLK}}$$

By definition, the **Misses per Instruction** are given by:

$$\begin{aligned} \text{Misses per Instr.} &= \frac{\# \text{ misses}}{\text{IC}} = \frac{\# \text{ mem. accesses}}{\text{IC}} \times \frac{\# \text{ misses}}{\# \text{ mem. accesses}} = \\ &= \text{Memory Accesses Per Instruction} \times \text{Miss Rate} \end{aligned}$$

$$\Rightarrow \text{CPU}_{\text{time}} = \text{IC} \times (\text{CPI}_{\text{exec}} + \text{MAPI} \times \text{Miss rate} \times \text{Miss Penalty}) \times T_{\text{CLK}}$$

# Impact of memory hierarchy on $\text{CPU}_{\text{time}}$

$$\text{CPU}_{\text{time}} = \text{IC} \times (\text{CPI}_{\text{exec}} + \text{MAPI} \times \text{Miss rate} \times \text{Miss penalty}) \times T_{\text{CLK}}$$

*Let us consider an ideal cache (100% hits):*

$$\text{CPU}_{\text{time}} = \text{IC} \times \text{CPI}_{\text{exec}} \times T_{\text{CLK}}$$

*Let us consider a system without cache (100% misses):*

$$\text{CPU}_{\text{time}} = \text{IC} \times (\text{CPI}_{\text{exec}} + \text{MAPI} \times \text{Miss penalty}) \times T_{\text{CLK}}$$

# Impact of memory hierarchy and pipeline stalls on $\text{CPU}_{\text{time}}$

$$\text{CPU}_{\text{time}} = \text{IC} \times (\text{CPI}_{\text{exec}} + \text{MAPI} \times \text{Miss rate} \times \text{Miss penalty}) \times T_{\text{CLK}}$$

*Putting all together...*

Let's also consider the stalls due to pipeline hazards:

$$\text{CPU}_{\text{time}} = \text{IC} \times (\text{CPI}_{\text{exec}} + \text{Stalls per instr} + \text{MAPI} \times \text{Miss rate} \times \text{Miss penalty}) \times T_{\text{CLK}}$$

# Memory stalls per instructions for L1 and L2 caches

- Average memory stalls per instructions:

**Memory stall cycles per instr** = Misses per instr x Miss Penalty

- Average memory stalls per instructions for L1 and L2 caches:

**Memory stall cycles per instr** =

Misses<sub>L1</sub> per instr X Hit Time<sub>L2</sub> + Misses<sub>L2</sub> per instr X Miss Penalty<sub>L2</sub>

# Impact of L1 and L2 on CPU<sub>time</sub>

$$\mathbf{CPU}_{\text{time}} = \text{IC} \times (\text{CPI}_{\text{exec}} + \text{Memory stall cycles per instr}) \times T_{\text{CLK}}$$

where:

Memory stall cycles per instr =

$$\text{Misses}_{L1} \text{ per instr} \times \text{Hit Time}_{L2} + \text{Misses}_{L2} \text{ per instr} \times \text{Miss Penalty}_{L2}$$

$$\text{Misses}_{L1} \text{ per instr} = \text{Memory Accesses Per Instr} \times \text{Miss Rate}_{L1}$$

$$\text{Misses}_{L2} \text{ per instr} = \text{Memory Accesses Per Instr} \times \text{Miss Rate}_{L1 L2}$$

$$\mathbf{CPU}_{\text{time}} = \text{IC} \times (\text{CPI}_{\text{exec}} + \text{MAPI} \times \text{MR}_{L1} \times \text{HT}_{L2} + \text{MAPI} \times \text{MR}_{L1 L2} \times \text{MP}_{L2}) \times T_{\text{CLK}}$$

# References

- Chapter 1 of the textbook:  
J. Hennessey, D. Patterson,  
*"Computer Architecture: a quantitative approach"*  
4<sup>th</sup> Edition, Morgan-Kaufmann Publishers.