Course on: “Advanced Computer Architectures”

Performance Evaluation

Prof. Cristina Silvano
Politecnico di Milano
email: 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 (pmph)
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 mph / 610 mph = 2.2 times faster
    = 6.5 hours / 3 hours
- Throughput of Concorde vs. Boeing 747 ?
  - Concord is 178,200 pmph / 286,700 pmph = 0.62 “times faster”
  - Boeing is 286,700 pmph / 178,200 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) 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"* $\Rightarrow \frac{\text{execution time (B)}}{\text{execution time (A)}} = 1 + \frac{n}{100}$

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

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

Clock cycles

- T_{CLK} = Period or clock cycle time = Time between two consecutive clock pulses
  - Seconds per cycle
- f_{CLK}= Clock frequency = Clock cycles per second = 1 / T
- 1 Hz = 1 / sec
- Examples:
  - The clock frequency of **500 MHz** corresponds to a clock cycle time: $1 / (500 * 10^6) = 2 * 10^{-9} = 2$ nsec
  - The clock frequency of **1 GHz** corresponds to a clock cycle time: $1 / (10^9) = 1 * 10^{-9} = 1$ nsec

Execution time or CPU Time

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

- 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*
  - *To increase the clock frequency*

CPU Time

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

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

where **CPI** = Clock Cycles / Instruction Count

IPC = 1 / CPI

CPU Time

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

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

$\Rightarrow$ **CPU time** = $\sum_{i=1}^{n} (CPI_i \times I_i) \times T_{CLK}$

CPI = $\sum_{i=1}^{n} (CPI_i \times F_i)$ Where $F_i = \frac{I_i}{IC}$

"instruction frequency"

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

Example

	Frequency	Clock Cycles
ALU	43%	1
Load	21%	4
Store	12%	4
Branch	12%	2
Jump	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:

CPI = 0.43 * 1 + 0.21 * 4 + 0.12 * 4 + 0.12 * 2 + 0.12 * 2 = 2.23

CPU time = IC * CPI * T_{clock} = 100 * 2.23 * 2 ns = 446 ns

MIPS Millions of instructions per second

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

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

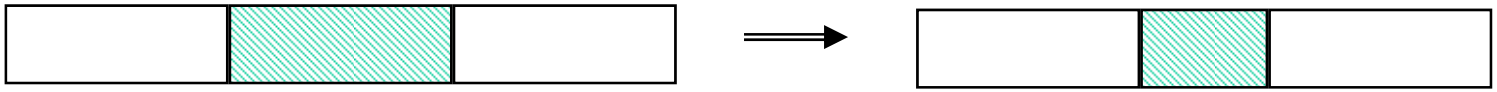
$$\textbf{MIPS} = \frac{f_{CLK}}{CPI \times 10^6}$$

Amdahl’s Law

How to evaluate the speedup

Speedup due to enhancement E:

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



Suppose that enhancement E accelerates a fraction F of the task by a factor S and the remainder of the task is unaffected then,

$$\text{ExTime(with E)} = ((1\text{-}F) + F/S) \times \text{ExTime(without E)}$$

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

Amdahl's Law

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

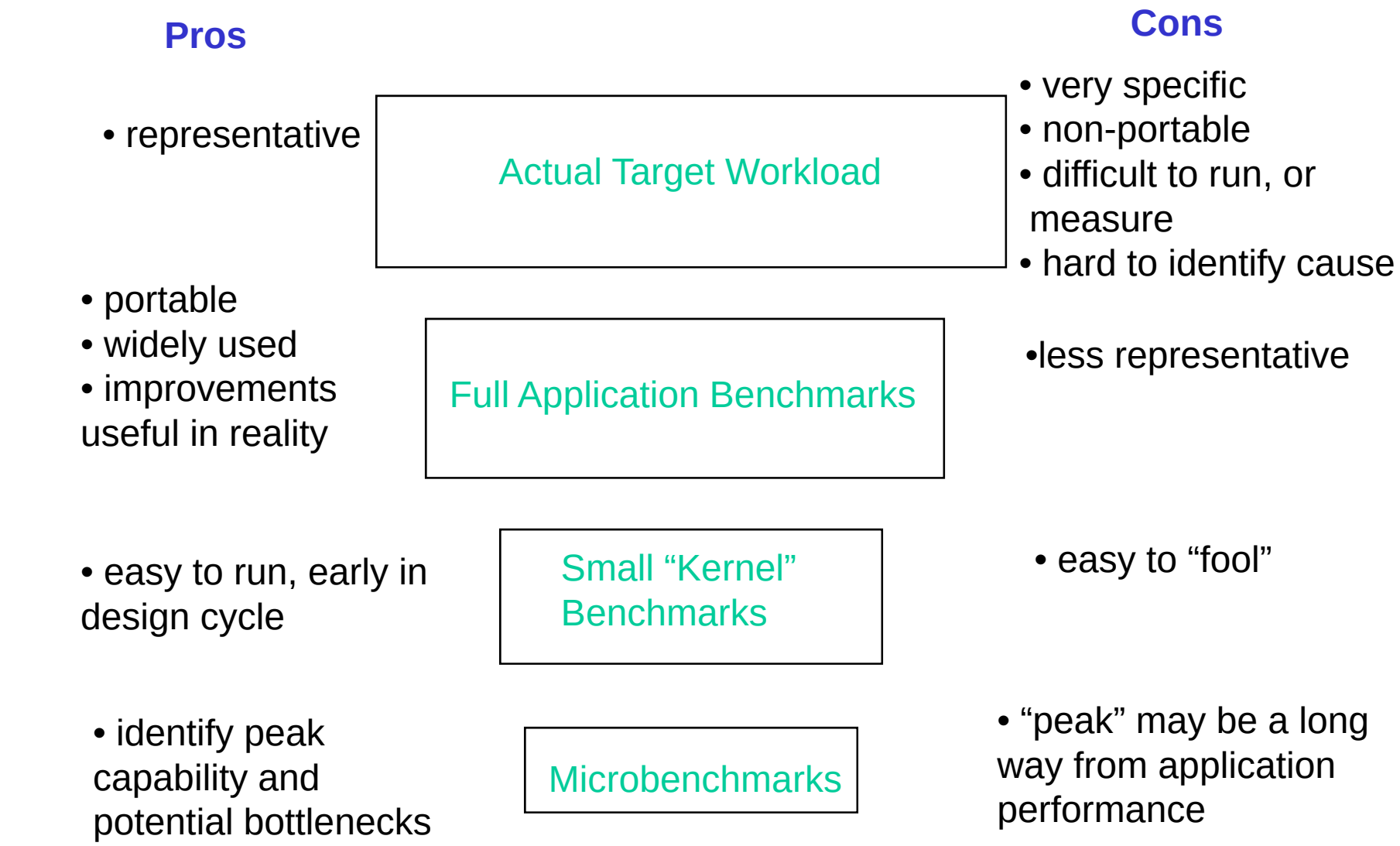
$$\textbf{Speedup}_{\textbf{overall}} = \frac{\text{ExTime}_{old}}{\text{ExTime}_{new}} = \frac{1}{(1 - \text{Fraction}_E) + \frac{\text{Fraction}_E}{\text{Speedup}_E}}$$

Example

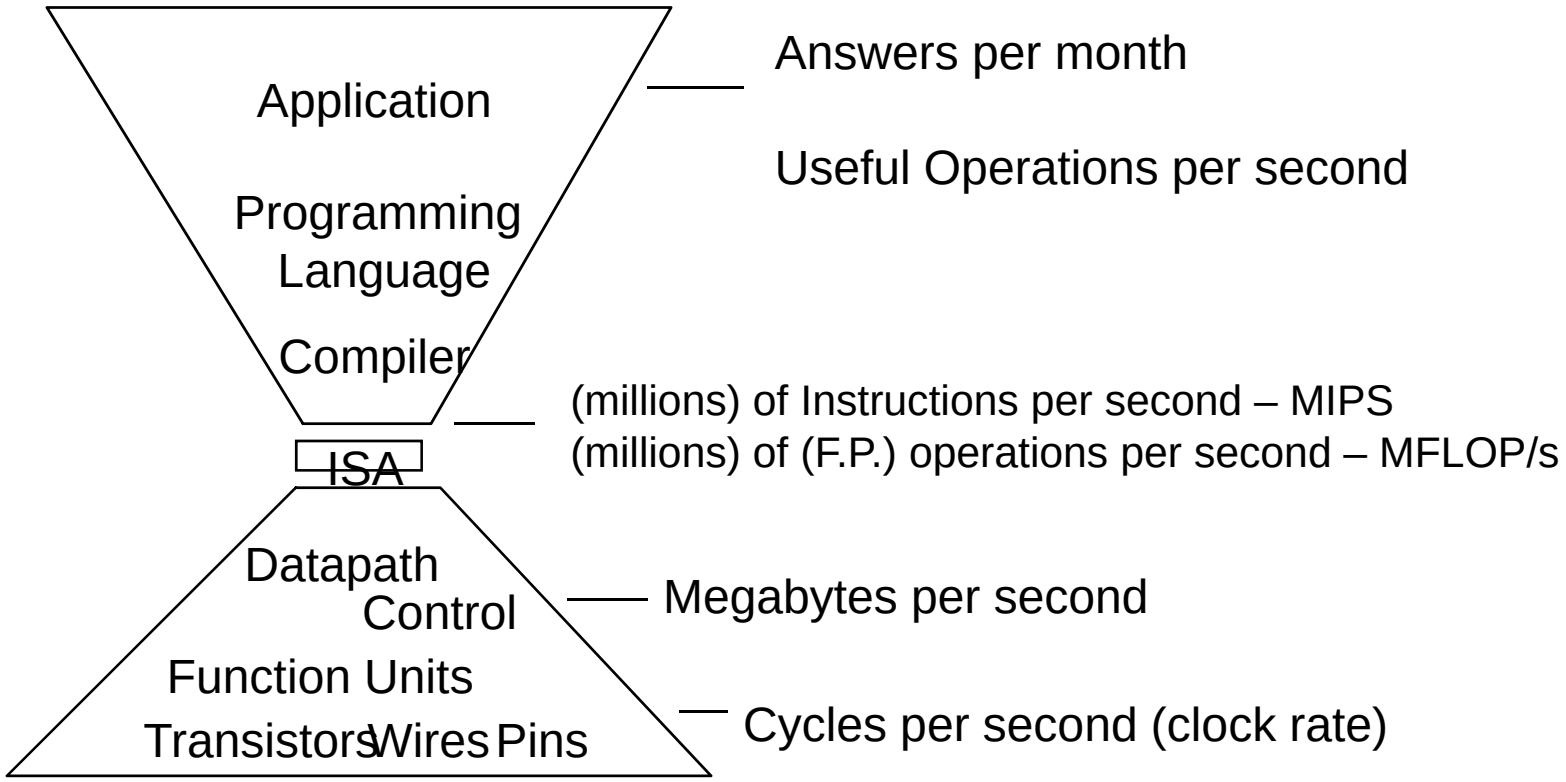
- Let us consider an enhancement for a CPU resulting ten time faster on computation than the original one but the original CPU is busy with computation only 40% of the time. What is the overall speedup gained by introducing the enhancement?
- **Solution:** Application of Amdahl's Law where:
  - **Fraction_E** = 0.4
  - **Speedup_E** = 10
- The overall speed up is given by:

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

Basis of Evaluation



Metrics of performance



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

Aspects of CPU Performance

CPU time= IC x CPI x T_{CLK}

	instr count	CPI	clock rate
Program	X		
Compiler	X	X	
Instr. Set	X	X	X
Organization		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 imbalance among the pipeline stages and overhead in the control of the pipeline.
  - Imbalance among pipeline stages reduces performance since the clock can run no faster than the time needed for the slowest pipe stage.
  - Pipeline overhead arises from pipeline register delay and clock skew.

Performance Metrics

IC = Instruction Count

 # Clock Cycles = IC + # Stall Cycles + 4

CPI = Clock Per Instruction = # Clock Cycles / IC =
(IC + # Stall Cycles + 4) / IC

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

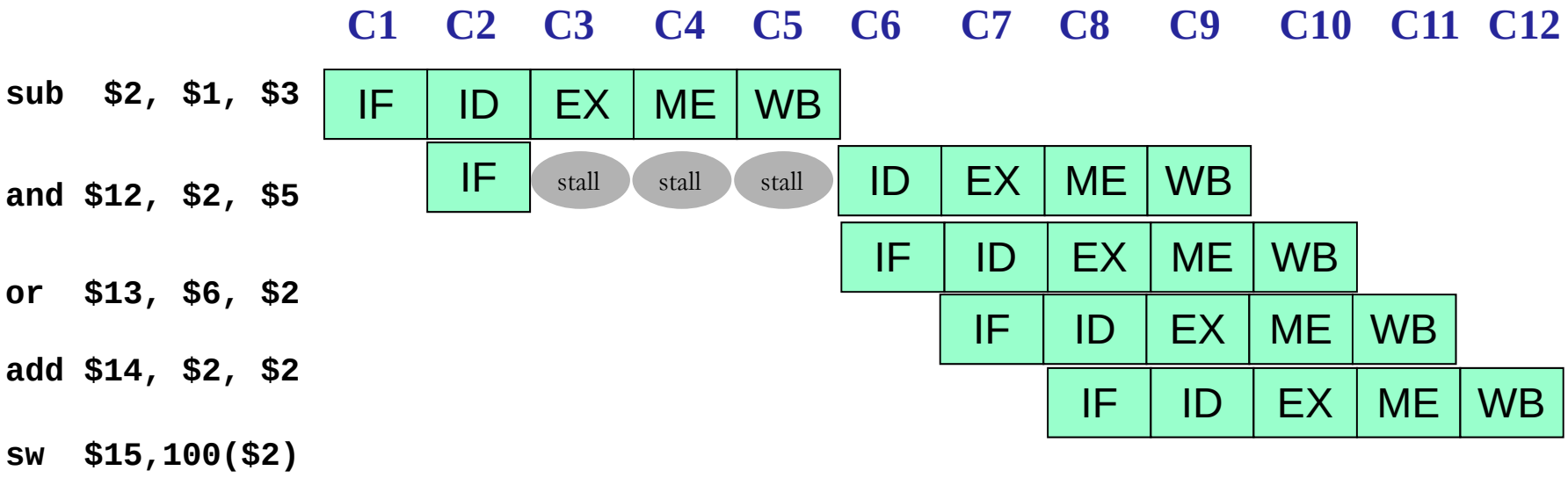
Example

IC = Instruction Count = 5

 # Clock Cycles = IC + # Stall Cycles + 4 = 5 + 3 + 4 = 12

CPI = Clock Per Instruction = # Clock Cycles / IC = 12 / 5 = 2.4

MIPS = f_{clock} / (CPI * 10^6) = 500 MHz / 2.4 * 10^6 = 208.3



Performance Metrics (2)

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

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

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

$$\mathbf{CPI}_{\mathbf{per\_iter}} = (IC_{per\ iter} + \#\ Stall\ Cycles_{per\_iter} + 4) / IC_{per\_iter}$$

$$= (m + k + 4) / m$$

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

Asymptotic Performance Metrics

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

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

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

$$\mathbf{CPI}_{\mathbf{AS}} = \lim_{n \to \infty} (IC_{AS} + \#\ \text{Stall Cycles}_{AS} + 4) / IC_{AS}$$

$$= \lim_{n \to \infty} (m*n + k*n + 4) / m*n$$

$$= (m + k) / m$$

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

Performance Issues in Pipelining

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

Ave. CPI Pipe = Ideal CPI + Pipe Stall Cycles per Instruction
= 1 + Pipe Stall Cycles per Instruction

- Pipe Stall Cycles per Instruction are due to Structural Hazards + Data Hazards + Control Hazards + Memory Stalls

Performance Issues in Pipelining

$$\textbf{Pipeline Speedup} = \frac{\text{Ave. Exec. Time Unpipelined}}{\text{Ave. Exec. Time Pipelined}} = \frac{\text{Ave. CPI Unp.}}{\text{Ave. CPI Pipe}} \times \frac{\text{Clock Cycle Unp.}}{\text{Clock Cycle Pipe}} =$$

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 two processors can be equal, so:

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

- Simple case: All instructions take the same number of cycles, which must also equal to the number of pipeline stages (called pipeline depth):

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

- If there are no pipeline stalls (ideal case), this leads to the intuitive result that pipelining can improve performance by the depth of the pipeline.

Performance of Branch Schemes

- What is the performance impact of conditional branches?

$$\textbf{Pipeline Speedup} = \frac{\text{Pipeline Depth}}{1 + \text{Pipe Stall Cycles per Instruction due to Branches}}$$

$$= \frac{\text{Pipeline Depth}}{1 + \text{Branch Frequency} \times \text{Branch Penalty}}$$

Performance evaluation of the memory hierarchy

Memory Hierarchy: Definitions

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

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

- **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 the data in the upper level with respect to the total number of memory accesses

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

- **Hit Rate + Miss Rate = 1**
- **Miss Penalty :** time needed to access the lower level and to replace the block in the upper level
- **Miss Time :**

  $$\textbf{Miss Time} = \textbf{Hit Time} + \textbf{Miss Penalty}$$

- **Hit Time << Miss Penalty**

Average Memory Access Time

AMAT = Hit Rate * Hit Time + Miss Rate * Miss Time

where:

Miss Time = Hit Time + Miss Penalty

Hit Rate + Miss Rate = 1

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

Performance evaluation: Impact of memory hierarchy on CPU_{time}

$\mathbf{CPU_{time}}$ = (CPU exec cycles + Memory stall cycles) x T_{CLK}

where: T_{CLK} = T clock cycle time period
CPU exec cycles = IC x CPI_{exec}
IC = Instruction Count
(CPI_{exec} includes ALU and LD/STORE instructions)
Memory stall cycles = IC x Misses per instr x Miss Penalty

$\Rightarrow$ $\mathbf{CPU_{time}}$ = IC x (CPI_{exec} + Misses per instr x Miss penalty) x T_{CLK}
where:
Misses per instr = Memory accesses per instruction x Miss rate

$\Rightarrow$ $\mathbf{CPU_{time}}$ = IC x (CPI_{exec} + MAPI x Miss rate x Miss penalty) x T_{CLK}

Performance evaluation: Impact of memory hierarchy on CPU_{time}

$\mathbf{CPU_{time}} = IC \times (CPI_{exec} + MAPI \times \text{Miss rate} \times \text{Miss penalty}) \times T_{CLK}$

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

$\mathbf{CPU_{time}} = IC \times CPI_{exec} \times T_{CLK}$

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

$\mathbf{CPU_{time}} = IC \times (CPI_{exec} + MAPI \times \text{Miss penalty}) \times T_{CLK}$

Performance evaluation: Impact of memory hierarchy and pipeline stalls on CPU_{time}

$$\mathbf{CPU_{time}} = IC \times (CPI_{exec} + MAPI \times Miss\ rate \times Miss\ penalty) \times T_{CLK}$$

Putting all together: Let us also consider the stalls due to pipeline hazards:

$$\mathbf{CPU_{time}} = IC \times (CPI_{exec} + \mathbf{Stalls\ per\ instr} + MAPI \times Miss\ rate \times Miss\ penalty) \times T_{CLK}$$

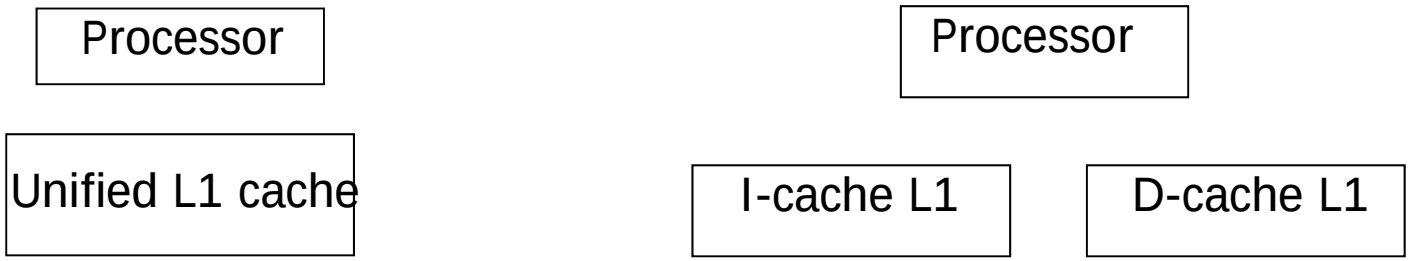
Cache Performance

- Average Memory Access Time:

  AMAT= Hit Time + Miss Rate * 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

- Average Memory Access Time for Separate I$ & D$

  AMAT= % Instr. (Hit Time + I$ Miss Rate * Miss Penalty) +
  % Data (Hit Time + D$ Miss Rate * Miss Penalty)

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

Unified vs Separate I$ & D$: Example of comparison

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

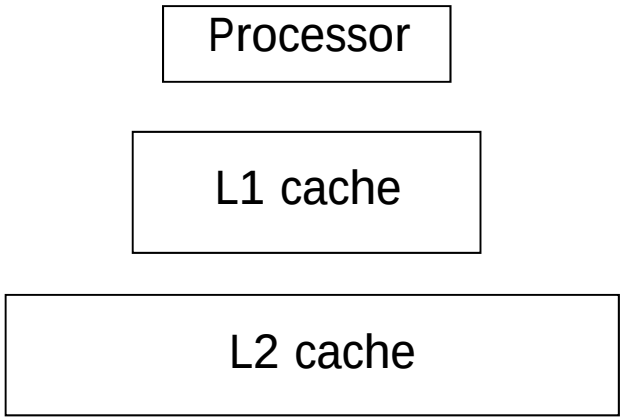
$AMAT_{Harvard}$=75%x(1+0.64%x50)+25%x(1+6.47%x50) = 2.05

$AMAT_{Unified}$=75%x(1+1.99%x50)+25%x(1+1+1.99%x50)= 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

AMAT = Hit Time$_{L1}$ + Miss Rate$_{L1}$ x Miss Penalty$_{L1}$
where:

Miss Penalty$_{L1}$ = Hit Time$_{L2}$ + Miss Rate$_{L2}$ x Miss Penalty$_{L2}$

$\Rightarrow$ **AMAT** = Hit Time$_{L1}$ + Miss Rate$_{L1}$ x (Hit Time$_{L2}$ + Miss Rate$_{L2}$ x 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*:
    Miss $rate_{L1}$ for L1 and 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 ***Miss $Rate_{L1}$,*** while for L2, it is ***(Miss $Rate_{L1}$ x 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$_{L2}$ =

  (40 /1000) x (20 /40) = 2%

AMAT for L1 and L2 Caches

AMAT = Hit Time_{L1} + Miss Rate_{L1} x (Hit Time_{L2} + Miss Rate_{L2} x Miss Penalty_{L2})

$\Rightarrow$ **AMAT** = Hit Time_{L1} + Miss Rate_{L1} x Hit Time_{L2} + $\text{Miss Rate}_{L1\ L2}$ x Miss Penalty_{L2}

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 =

$Misses_{L1}$ per instr X $Hit\ Time_{L2}$ +

$Misses_{L2}$ per instr X $Miss\ Penalty_{L2}$

Impact of L1 and L2 on CPU_{time}

$\mathbf{CPU_{time}}$ = IC x (CPI_{exec} + Memory stall cycles per instr) x T_{CLK}

where:

Memory stall cycles per instr =
$Misses_{L1}$ per instr X Hit $Time_{L2}$ +
$Misses_{L2}$ per instr X Miss $Penalty_{L2}$

$Misses_{L1}$ per instr = Memory Accesses Per Instr x $Miss\ Rate_{L1}$

$Misses_{L2}$ per instr = Memory Accesses Per Instr x $Miss\ Rate_{L1\ L2}$

$\Rightarrow$ $\mathbf{CPU_{time}}$ = IC x (CPI_{exec} + MAPI x MR_{L1} x HT_{L2} + MAPI x $MR_{L1\ L2}$ x MP_{L2}) x T_{CLK}

References

- Chapter 1 of Text Book:

  J. Hennessey, D. Patterson,

  "Computer Architecture: a quantitative approach"
  5 th Edition, Morgan-Kaufmann Publishers.