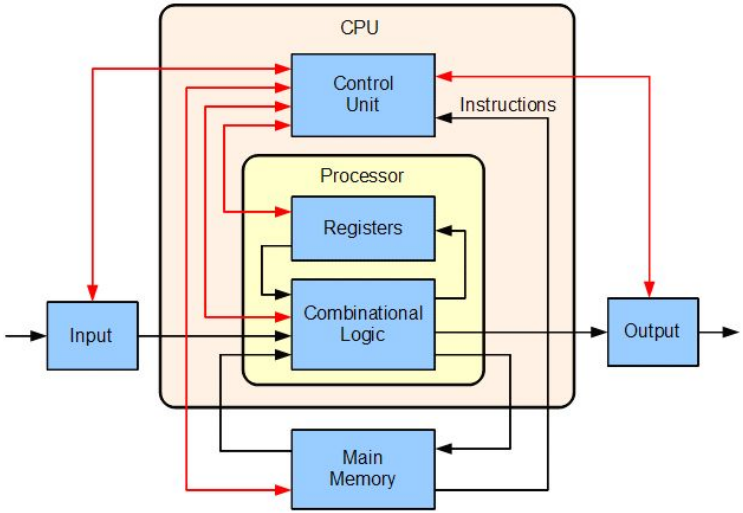
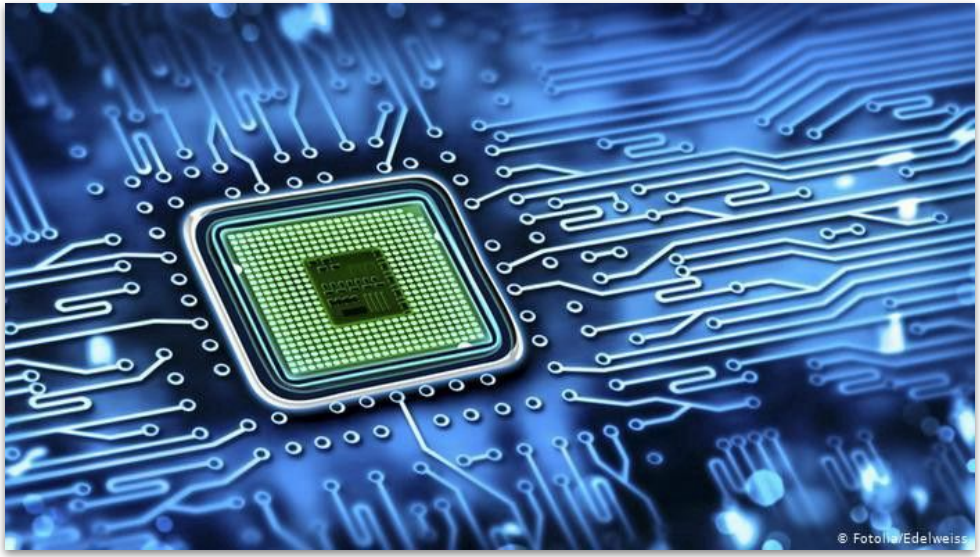
Review: CP#2, Part3
CSC 656 Fall 2025
28 Oct 2025





San Francisco State University

Today

Review:

- CP#2
- DM cache, multi-word blocks exercise
- In-class partnering activities to review Part 3

Reading for today:

P&H:

- 5.1, Introduction (Memory)
- 5.2, Memory Technologies
- 5.3, Cache Basics
- 5.4, Measuring and Improving Cache Performance

Upcoming deadlines/dates:

- Thu 30 Oct 2025: Quiz#3

CP#2 Review

CP#2 Overview

Study of how memory access patterns influence performance

Three different methods that implement different memory access patterns

Methods:

- Run at varying problem size
- Measure runtime
- Compute derived performance metrics: MFLOPS, % mem b/w used, latency

Findings: memory access patterns indeed impact performance

Codes

Sum_direct: sum up values of loop index variable

Sum_vector: sum up values in an array A[] using linear traversal

Sum_indirect: sum up values in an array A[] using a random traversal

Sample machine specs:
- Arithmetic op: 1ns
- Memory op: 5 ns
- CPU clock: 3 GHz

For each algorithm:
- How many arithmetic ops?
- How many memory accesses?
- Estimated runtime for N=256 Mi

Raw Data: F24, Perlmutter CPU node, -O3

```
1 # 10/2025, Perlmutter CPU node, -O1
2 Problem size, sum direct, sum vector, sum indirect
3 8388608, 0.0024, 0.0035,  0.0264
4 16777216, 0.0050, 0.0059, 0.0552
5 33554432, 0.0097, 0.0120, 0.1374
6 67108864, 0.0193, 0.0236, 0.5292
7 134217728, 0.0384, 0.0445, 1.9923
8 268435456, 0.0766, 0.0859, 4.2317
```

Metrics:

MFLOPS – millions of floating point operations/second

# of operations / time(sec) → number of operations/second

(# of operations / 1e6) / time(secs) → Millions of operations/second

How many operations do these codes perform?

- Sum_direct:
- Sum_vector:
- Sum_indirect:

CP#2 MFLOP/s

AMD Milan Peak: 39.2 GFLOP/s /core

Sum_direct:

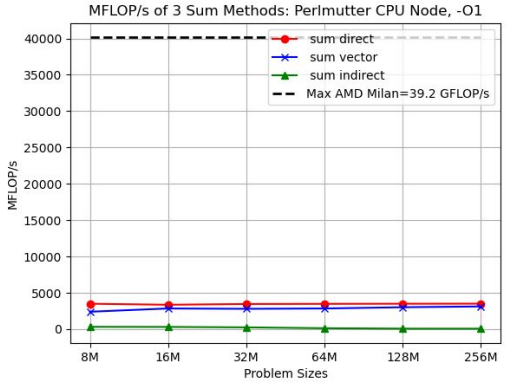
- Steady at ~3.4 GFLOP/s

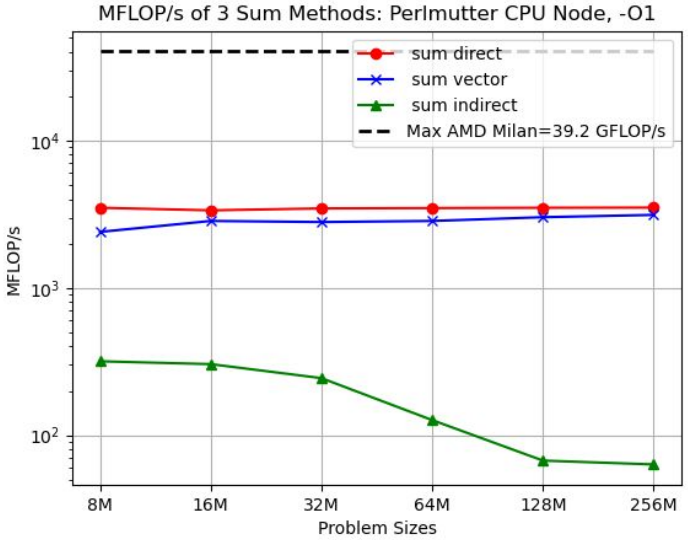
Sum_vector:

- Steady-ish at ~2.8 GFLOP/s

Sum_indirect:

- ~317 MFLOP/s at N=8M
- ~63 MFLOP/s at N=256M





Metrics:

% mem bandwidth utilized

# of bytes transferred / time(sec) → your bytes/second

(# of bytes transferred / 1e9) / time(secs) → your GB/sec

(# of bytes transferred / 1e9) / time(secs) / max_GB/sec * 100% = your %

How many bytes moved by each of these codes?

- Sum_direct:
- Sum_vector:
- Sum_indirect:

CP#2 % Mem Bandwidth

AMD Milan Peak mem b/w: 204.8 GB/s

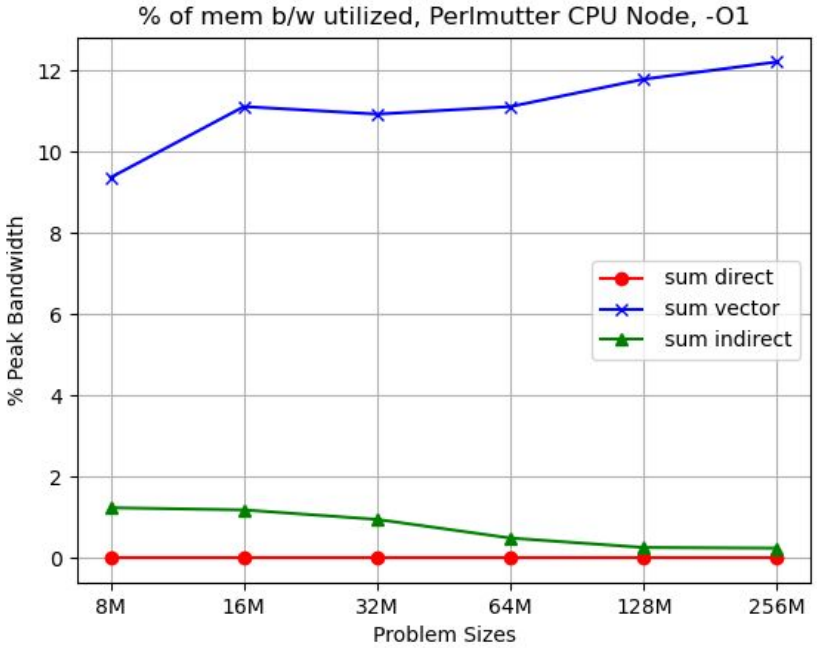
Sum_direct: no memory accesses

Sum_vector:

- 9-12% of peak (19-24 GB/s)
- Level off at ~12% of peak

Sum_indirect:

- Starts at ~1.2% of peak
- Performance drops with increasing problem size



Metrics:

Average memory latency

time(sec) / (number of memory accesses) → time(sec) per access

But want avg memory latency in *nanoseconds*

time(sec) * 1e9 / (number of mem accesses) → time(ns) per access

How many memory accesses by each of these codes?

- Sum_direct:
- Sum_vector:
- Sum_indirect:

CP#2 Memory Latency

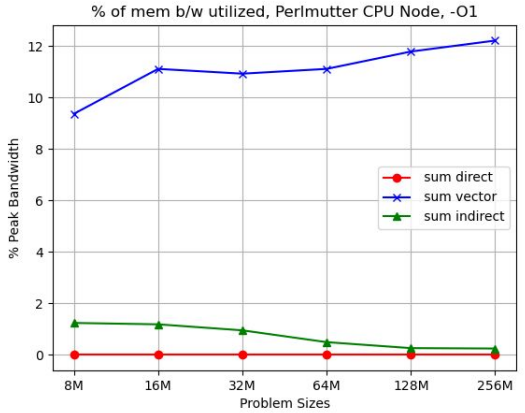
Sum_direct: no memory accesses

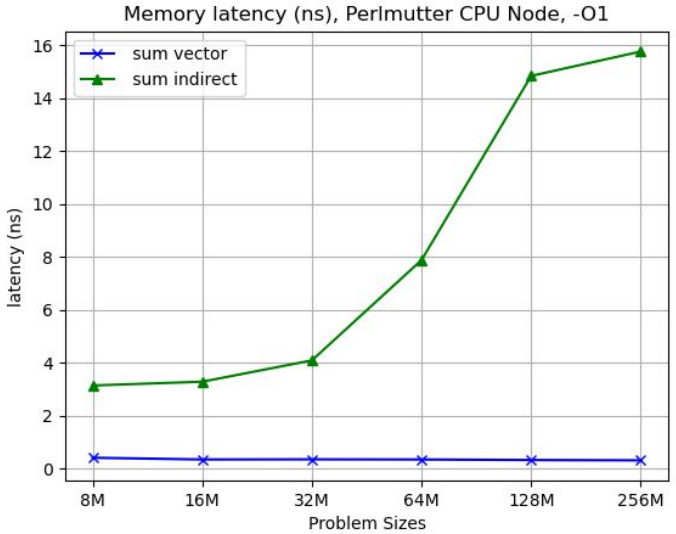
Sum_vector:

- Steady at ~0.40 ns

Sum_indirect:

- Starts at ~3 ns
- Increasing latency with increasing problem size





San Francisco State University

Analysis questions

1. What types of operations are more expensive and why?
2. Computational rate: which of the 3 codes has best MFLOPS, and why?
3. Mem b/w usage: which has better b/w utilization, and why?
4. Memory latency: which has lower memory latency, and why?

Where did things go wrong?

Sum_indirect implementation

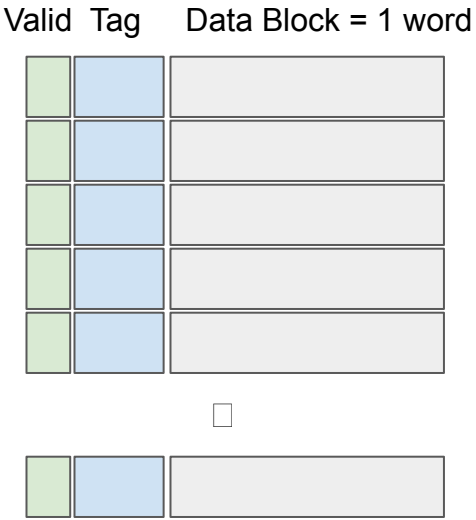
Computing metrics: MFLOPs, % mem b/w utilized, memory latency

Multi-word Cache Blocks in Direct-mapped Caches

Single Word Cache Blocks

Up until now, our direct-mapped cache consists of:

- Single-word blocks for holding data
- An n-bit tags field
  - How many bits are needed?
- A 1-bit Valid field



Multi-Word Cache Blocks

Before:

- Our DM cache blocks were 1 word

Now our direct-mapped cache contains

- Multi-word blocks for holding data
- An n-bit tags field
- A 1-bit Valid field

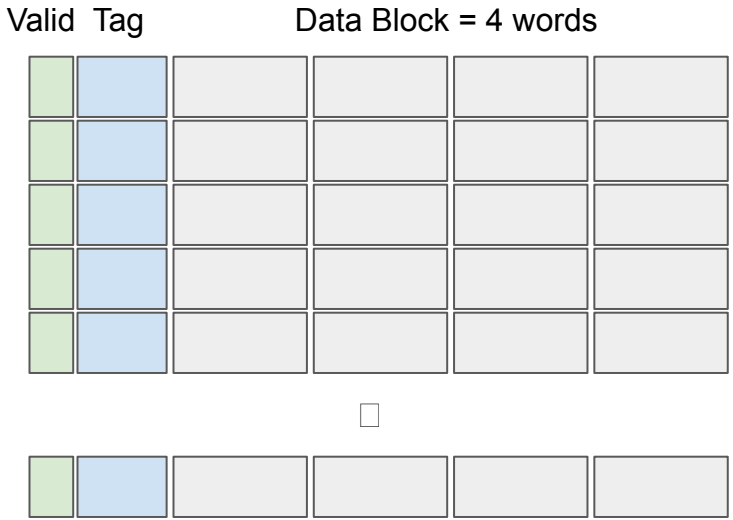


Image source: P&H 5.3

Increasing the Cache Block Size

As the block size *increases*:

- Th
  be
  tha
  de
- Th
  inc
  ha
  from lower in the memory hierarchy

What is the size of the L1 cache line on Perlmutter's AMD EPYC 7763 Milan processors?

Looking at this chart, why do you think the designers chose that value?

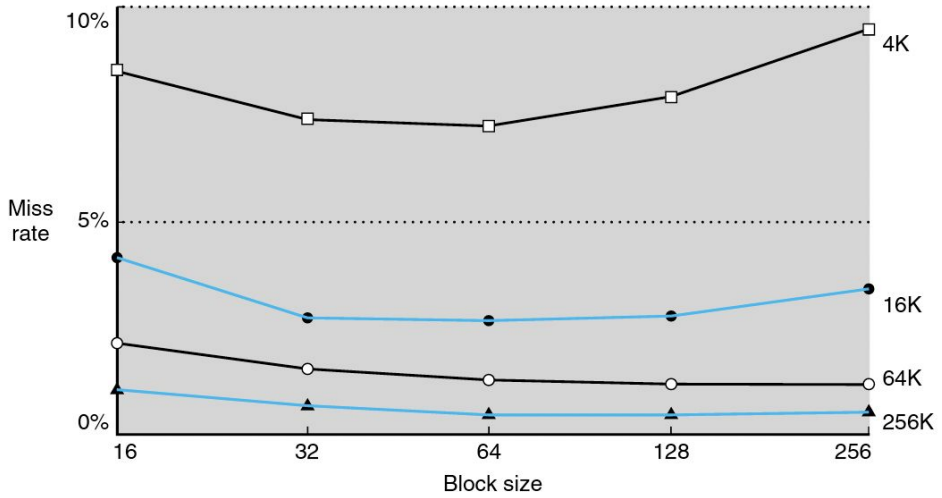


FIGURE 5.11 Miss rate versus block size. Note that the miss rate actually goes up if the block size is too large relative to the cache size. Each line represents a cache of different size. (This figure is independent of associativity, discussed soon.) Unfortunately, SPEC CPU2000 traces would take too long if block size were included, so this data is based on SPEC92.

Image source: P&H 5.3

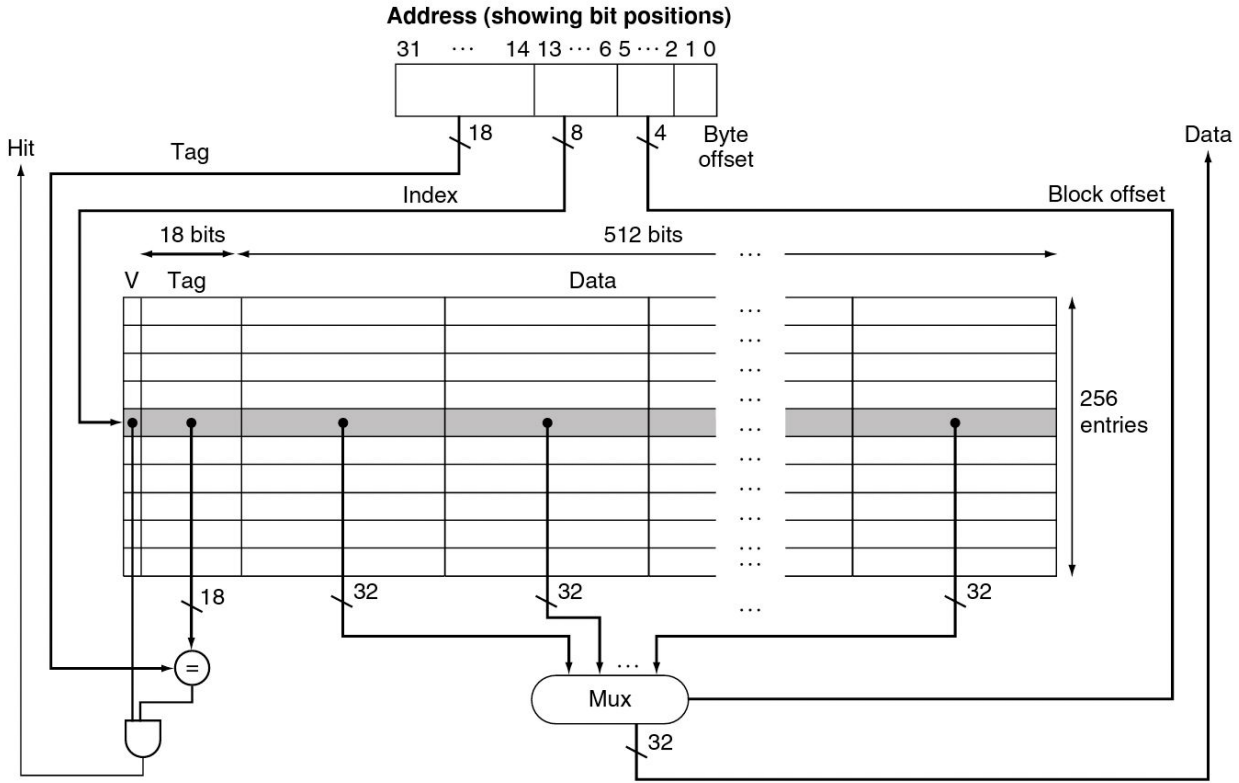
Intrinsity FastMATH Data Cache

Configuration:

- 256 entries (blocks)
- 16 words/block

Questions:

- How many A_m bits needed for cache index?
- How wide is the tags field?
- How do we figure out which word within a block we are accessing?
- What address range is referred to by a single tag value?



Activity Review:
Direct-mapped Caches, Multi-word Blocks

CSC 656-01 F25 Activity: Direct-mapped caches, multi-word blocks
Due: Sun 26 Oct 2025 23:59 PDT, No Late Submissions

Given:

- A direct-mapped cache of 8 cache blocks, each of which is two words in size.
- A list of 8-bit memory address transactions (memory addresses are *byte addresses*):
  - 3, 180, 2, 44, 43, 191, 88, 190, 14, 181, 186, 253

Questions:

1. How many bits wide is the block offset?
2. How many bits wide is the cache index?
3. How many bits wide is the tags field?
4. Create a table showing the state of the cache after each memory transaction. The table will have 12 rows, one per memory transaction, and 7 columns: memory address in decimal (base 10), memory address in binary, tags value in binary after update, cache index in binary, block offset in binary, byte offset in binary, and a flag indicating if the transaction is a hit (H) or miss (M).

Your table should have columns that look something like this:

Mem.addr.10	Mem.addr.2	Tags	Cache Index	Block Offset	Byte offset	H/M Flag
3	0000 0011	XXX	XXX	XXX	XX	XX
…						
…						

index in binary, block offset in binary, byte offset in binary, and a flag indicating if the transaction is a hit (H) or miss (M).

Your table should have columns that look something like this:

Mem.addr.10	Mem.addr.2	Tags	Cache Index	Block Offset	Byte offset	H/M Flag
3	0000 0011	XXX	XXX	XXX	XX	XX
…						
…						

	Mem address	Tags bits 6,7	Cache indx bits 3,4,5	Block offset bit 2	byte offset bits 0,1	H/M	Note
3	00 000 0 11	00	000	0	11	M	
180	10 110 1 00	10	110	1	00	M	
2	00 000 0 10	00	000	0	10	H	See 3

index in binary, block offset in binary, byte offset in binary, and a flag indicating if the transaction is a hit (H) or miss (M).

Your table should have columns that look something like this:

Mem.addr.10	Mem.addr.2	Tags	Cache Index	Block Offset	Byte offset	H/M Flag
3	0000 0011	XXX	XXX	XXX	XX	XX
…						
…						

	Mem address	Tags bits 6,7	Cache indx bits 3,4,5	Block offset bit 2	byte offset bits 0,1	H/M	Note
3	00 000 0 11	00	000	0	11	M	
180	10 110 1 00	10	110	1	00	M	
2	00 000 0 10	00	000	0	10	H	See 3
44	00 101 1 00	00	101	1	00	M	
43	00 101 0 11	00	101	0	11	H	See 44
191	10 111 1 11	10	111	1	11	M	

… remaining 6 entries of the table omitted for brevity

In-class practice problems

Cache configuration

Assume:

- Direct-mapped cache, multi-word blocks
- 1KiB data size
- Blocks contain 16 bytes
- 32-bit memory addresses (MIPS standard)

Derive/compute:

- Mapping of address bits to:
  - Byte offset
  - Block offset
  - Cache index
  - Tags
- Total # bits storage required for this cache

Code sample: What is the cache miss rate?

```
2 int32_t sum(int32_t A[], int32_t N)
3 {
4     int32_t result=0;
5     for (int32_t i=0; i<N; i++)
6         result += A[i];
7     return N;
8 }
9
```

Same cache configuration as previous question:

- Direct-mapped cache, multi-word blocks
- 1KiB data size
- Blocks contain 16 bytes

Compute/derive:

- What are the cache hit/miss rates for this code on that particular cache?

Code sample: estimate performance

```
2 int32_t sum(int32_t A[], int32_t N)
3 {
4     int32_t result=0;
5     for (int32_t i=0; i<N; i++)
6         result += A[i];
7     return N;
8 }
9
```

Same cache configuration:

- Direct-mapped cache, multi-word blocks
- 1KiB data size
- Blocks contain 16 bytes

Machine configuration/specs:

- CPU clock: 1 GHz
- Arithmetic operation: 2 clocks
- Cache hit time: 2 clocks
- Miss penalty: 100 clocks

Problem size: N=1e9

What is the estimated runtime?

Food for thought

How does performance change as a function of cache configuration?

Quiz#3 Musings

Quiz#3 Logistics and Rules

When: Thu 10/30/2025, 2:00-3:15pm (regular class time)

Where: via the usual CSC 656 Lecture zoom link

- You must do your own work on the quiz, the work you submit must be yours and yours alone
- You must be logged into zoom the entire time you are taking the quiz.

Quiz#3 Permitted Resources

Permitted:

1-page front-back “Cheat Sheet” – your own notes

PDFs of chapters from the Computer Organization textbook

PDFs of lecture slides

Calculator

Strongly discouraged:

Google searches

Random content on the web

Generative AI: ChatGPT, etc.

Remember the CSC 656 policy on the use of GenAI. It will be enforced on all quizzes, homeworks, and programming projects.

Forbidden/Verboten:

Consulting with others in any way about the quiz while the quiz is in session

Quiz#3, ctd

All questions should be easy/straightforward to answer if you've participated in the lectures, the Q&A, and completed the activities, problem sets, and coding project

You will be required to reason about concepts: test of learning, not memorization

Part 3 Overview and Summary

CSC 656 Course Topical Outline and Schedule (tentative)

Weeks 1-2-3	Instruction set architectures (ISA): MIPS, ARM, x86
Weeks 4-5-6	Digital logic fundamentals data and control pathways, processor organization, pipelining
Weeks 7-8-9	Memories and Caches, Performance Analysis
Weeks 10-11-12	Shared-memory parallelization on CPUs, vectorization
Weeks 13-14-15	Shared-memory parallelization on GPUs, Advanced processor architecture

Part 3 Topics

Performance analysis

- What factors contribute to code performance?
- How do we measure performance?
- How do we improve performance?

The memory hierarchy

- Caches and their impact on performance
- Optimizing code to take advantage of the memory hierarchy

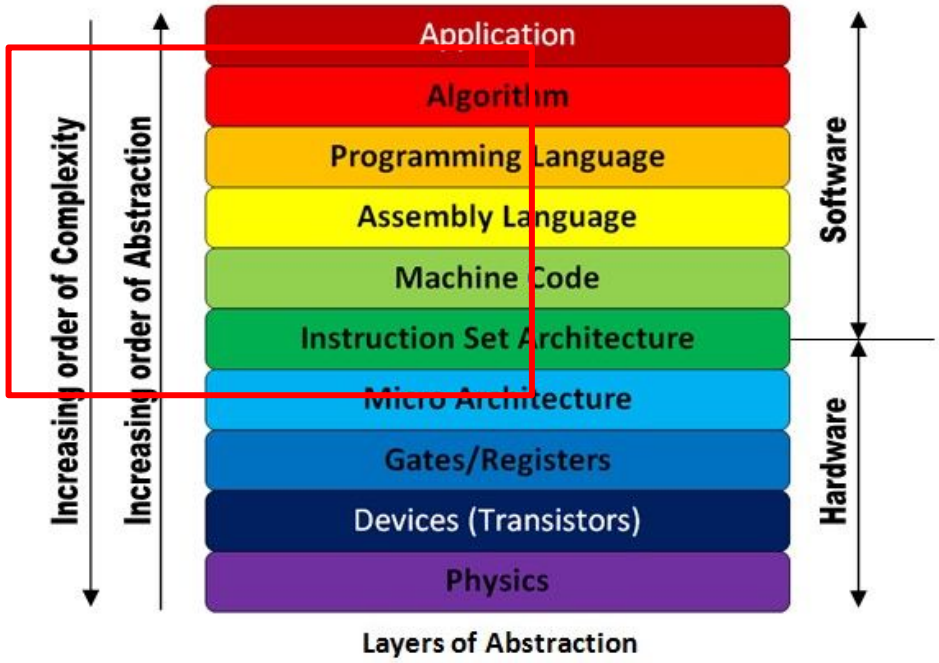


Image source: electronics.stackexchange.com

Defining “Performance” based on Execution Time

$$\text{Performance}_X = \frac{1}{\text{Execution time}_X}$$

Performance inversely proportional to execution time

As execution time → *infinity,* Performance → 0

As execution time → 0, Performance → infinity

CPU Performance

$$\text{CPU execution time for a program} = \text{CPU clock cycles for a program} \times \text{Clock cycle time}$$

Elapsed time is a function of:

- Clock speed – the faster the clock, the less time required per instruction
- Total number of clock cycles needed for a program
  - How many instructions
  - Number of clocks per instruction – varies as a function of instruction type
  - Recall differences between lw instruction (800 ps) and beq instruction (500 ps)

How many clock cycles needed for a program?

$$\text{CPU clock cycles} = \text{Instructions for a program} \times \text{Average clock cycles per instruction}$$

Instructions for a program:

- The total number of instructions in your program: arithmetic, memory access, etc.

Average clock cycles per instruction, or CPI:

- #CPU clock cycles / # instructions for program = Average clock cycles/instruction
- #CPU clock cycles may be estimated or observed

Analyzing Performance By Looking at CPI

Given that

- CPI = #CPU clock cycles / # instructions for program
- Cost of arithmetic = X ns, cost of memory access = Y ns
- Assume X << Y

Which program will have a greater CPI? Why?

Program A:

Let N = 1M words
Computes B[i] = sqrt(A[i])

Program B:

Let N = 1M words
Computes sum += A[i]

“Classic” CPU Performance Equation

$$\text{CPU time} = \frac{\text{Instruction count} \times \text{CPI}}{\text{Clock rate}}$$

Three factors influencing performance:

- The clock rate. A faster clock reduces CPU time (improves performance)
- The number of instructions being executed
- The average time (clocks) required to execute each instruction

Reasoning About Performance

```
t = 0
for i in range(0,N):
    t += A[i]
```

Workload: read in bytes from memory, addi 1 to each item

Assume problem size N=3,000,000,000 (3 billion)

Computer A:

- CPU clock = 2 Ghz
- Memory bandwidth = 1 GB/s

Computer B:

- CPU clock = 1.5 Ghz
- Memory bandwidth = 2 GB/s

Which machine executes the problem more quickly?

Simple performance models:

Model #1 ("Classic):

t = total program cycles * CPI / clock rate

Model #2:
Execution time = $t_f + t_m$

Where:

- t_f = time for arithmetic operations
- t_m = time for memory accesses

Estimate Runtime for N=3,000,000 (3 billion)

```
t = 0
for i in range(0,N):
    t += A[i]
```

Computer A:

- CPU clock = 2 GHz
- Memory bandwidth = 1 GB/s

t_f = time for arithmetic

→ 3B items / 2.0B clocks/sec = 1.5 sec

t_m = time for memory accesses

→ 3 B items / 1 GB/s = 3 sec

Total time = 1.5 sec + 3 sec = 4.5 sec

Computer B:

- CPU clock = 1.5 GHz
- Memory bandwidth = 2 GB/s

t_f = time for arithmetic

→ 3B items / 1.5B clocks/sec = 2.0 sec

t_m = time for memory accesses

→ 3B items / 2 GB/s = 1.5 sec

Total time = 3.5 sec

HW#4 - Use a Simple Model to Estimate Performance

Assume:

- CPU Clock = 1 Ghz
- 1 OP requires 5 clock cycles (+/-, conditional, etc.)
- 1 memory access requires 100 clock cycles (Read or Write)

Questions:

- How many arithmetic instructions required for 3 different algorithms?
- How many memory accesses required for 3 different algorithms?
- How many clocks required for 3 different algorithms?
- What is the CPI (cycles per instruction) required for 3 different algorithms
- Prediction of execution time for this problem size on this platform

The Three Algorithms:

- Sum direct
- Sum vector
- Sum indirect

Comments:

- *Do not include print stmt or problem setup instructions in your analysis*
- *For "free": accessing loop index variable, incrementing loop index variable, accessing "accumulator"*

Image source: P&H 5.1

Caches and the Memory Hierarchy

Cache: a safe place for storing things

Memory cache: layer of storage between main memory and the processor: the cache is smaller and faster than the main memory

Other types of cache:

- Disk cache
- Web browser cache
- …

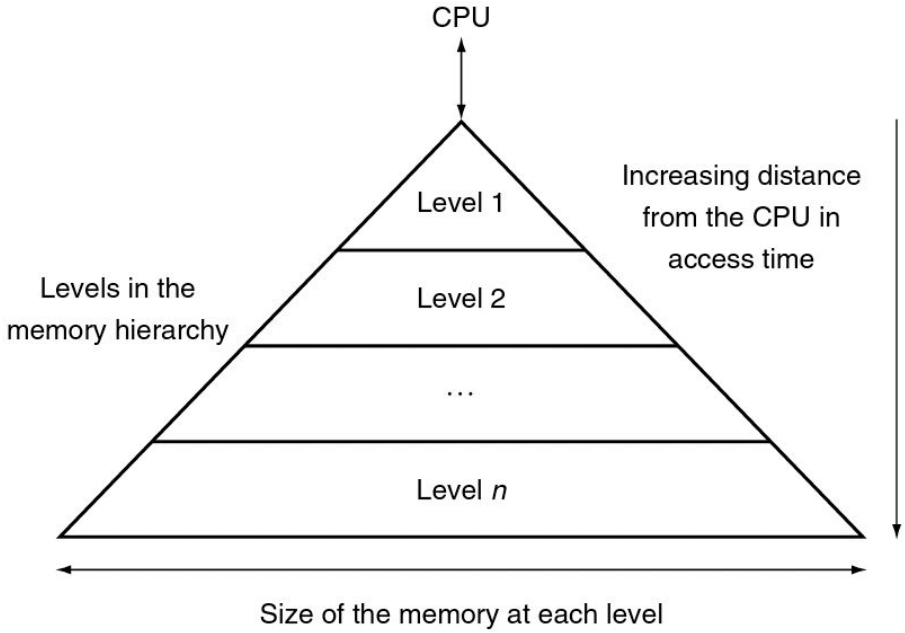
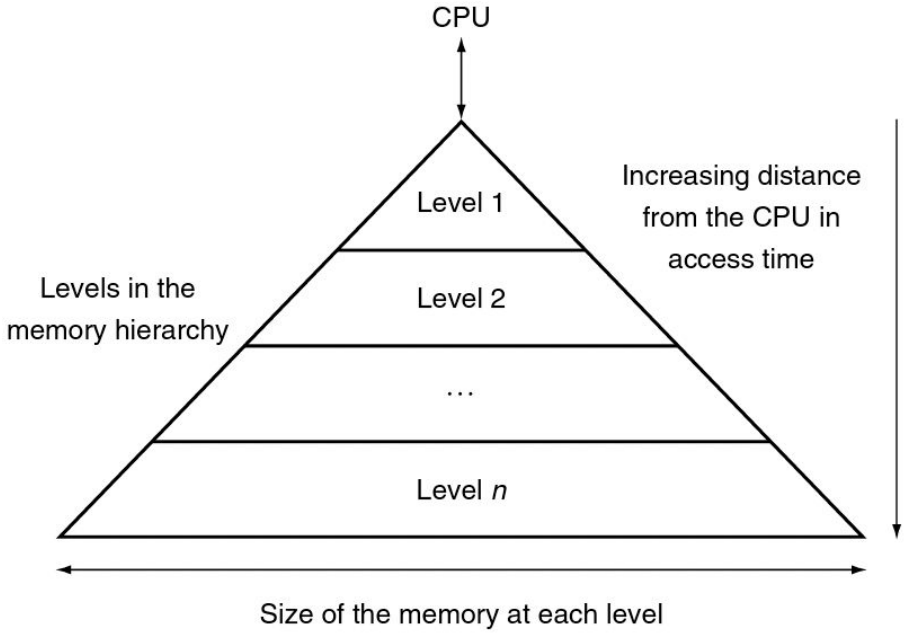


Image source: P&H 5.1

Caches and the Memory Hierarchy

Memory hierarchies take advantage of spatial and temporal locality:

- Temporal locality: keeping more recently accessed items close to the processor
- Spatial locality: by moving blocks of contiguous data through the memory hierarchy



Caches and Terminology

Hit: an access found in the upper level
Miss: an access not found in the upper level

hit rate (or hit ratio) = no. of hits / no. of accesses
miss rate (or miss ratio) = no. of misses / no. of accesses

hit time = time for a hit
miss time = time for a miss
miss penalty = miss time - hit time

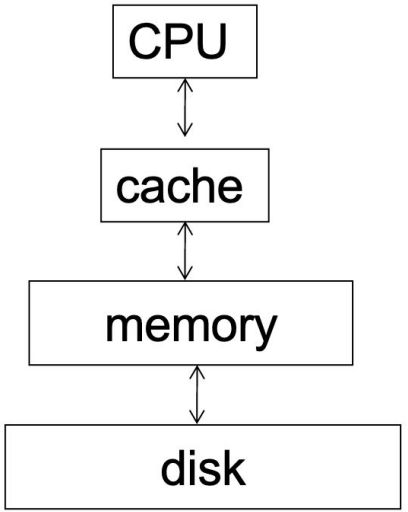


Image source: P&H 5.4

Average Memory Access Time: Example

$$\text{AMAT} = \text{Time for a hit} + \text{Miss rate} \times \text{Miss penalty}$$

Assume:

- Processor clock cycle time: 1ns
- Cache access time: 2 cycles
- Miss penalty: 100 cycles

What is average memory access time when:

- Miss rate = 5%?
- Miss rate = 95%?

When miss rate = 5%:

AMAT = hit time + miss rate * miss penalty

= 2 cycles + 0.05 * 100 cycles
= 7 cycles (or 7 ns)

When miss rate = 95%:

AMAT = hit time + miss rate * miss penalty

= 2 cycles + 0.95 * 100 cycles
= 97 cycles

Direct-mapped caches

Most direct-mapped caches use *modulo mapping* :

Cache address = (memory block address) modulo (number of blocks in the cache)

Cache address = memory address % cache size in blocks

When the cache size is a power of 2, then module can be computed by using the low-order $\log_2$ bits of the address

Terminology:

- Blocks vs. bytes vs. words
- For now, let's assume blocks == words