CS 354 - Machine Organization & Programming
Tuesday, October 29th, 2019

Midterm Exam 2 (~18%): Thursday, November 7th, 7:15 - 9:15 pm

Project p4A (~2%): DUE at 10 pm on Tuesday, November 5th
Project p4B (~4%): Assigned later this week
Homework hw4 (1.5%): DUE at 10 pm on Thursday, October 31st

Last Time

Set Associative Cache
Replacement Policies
Fully Associative Cache
Writing to Caches
Cache Performance Metrics
Cache Parameters and Performance

Today

Impact of Stride
Memory Mountain
C, Assembly, & Machine Code
Low-level View of Data
Registers
Instructions - MOV, PUSH, POP

Next Time

More Instructions and Operands
Read: B&O 3.5, 3.6

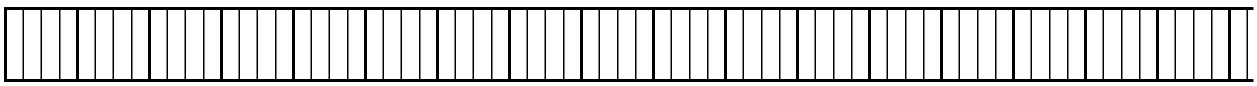
Impact of Stride

Stride Misses

Example:

```
int initArray(int a[][8], int rows) {
   for (int i = 0; i < rows; i++)
      for(int j = 0; j < 8; j++)
         a[i][j] = i * j;
}
```

→ Draw a diagram of the memory layout of the first two rows of `a`:



Assume: `a` is aligned with cache blocks and <u>is too big to fit entirely into the cache</u>
words are 4 bytes, block size is 16 bytes
direct-mapped cache is initially empty, write allocate used

→ Indicate the order elements are accessed in the table below and mark H for hit or M for miss:

a[i][j]	j = 0	1	2	3	4	5	6	7
i = 0								
1								
...								

→ Now exchange the `i` and `j` loops mark the table again:

a[i][j]	j = 0	1	2	3	4	5	6	7
i = 0								
1								
...								

Memory Mountain

Independent Variables

stride - 1 to 16 double words step size used to scan through array
size - 2K to 64 MB arraysize

Dependent Variable

read throughput - 0 to 7000 MB/s

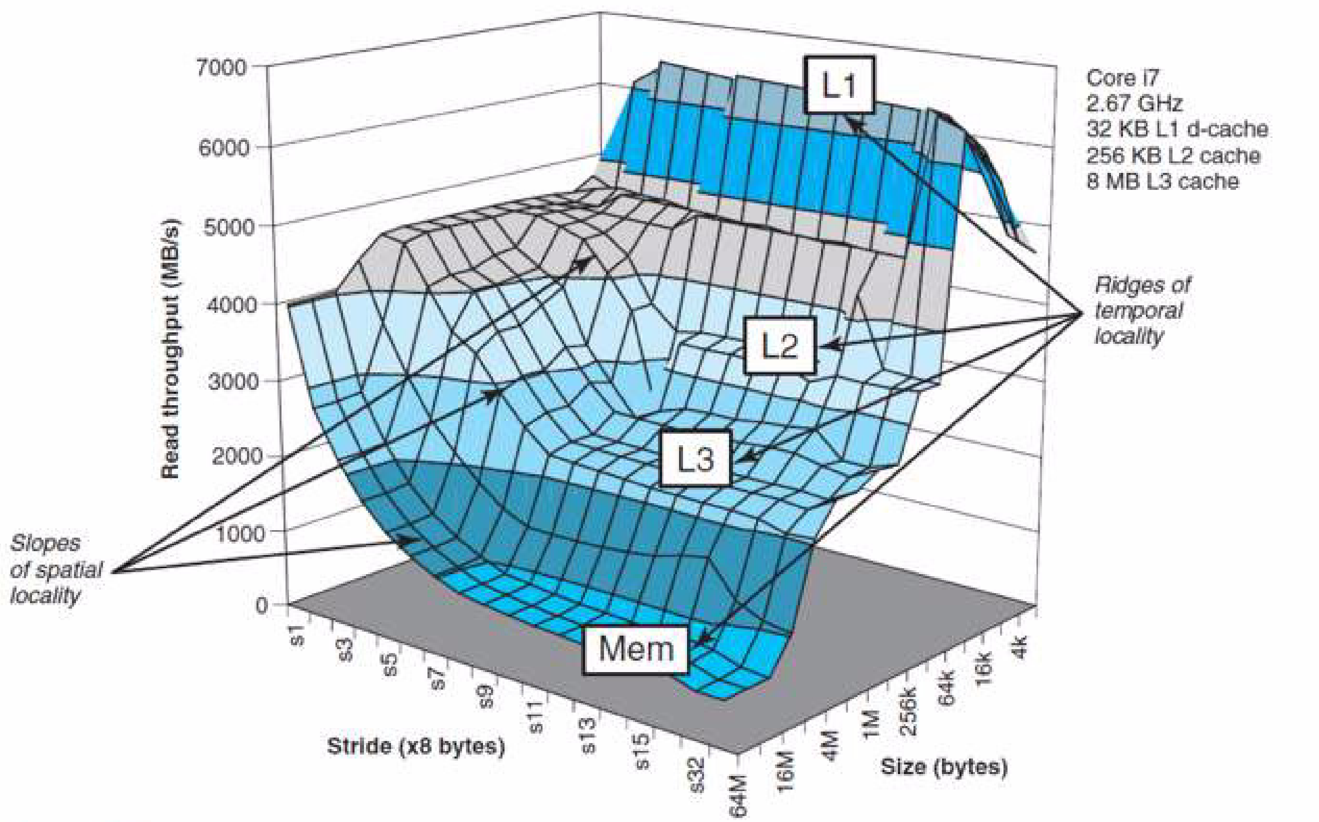


Figure 6.43 The memory mountain.

Computer Systems, A Programmer's Perspective
Second Edition, Bryant and O'Hallaron

Temporal Locality Impacts

Spatial Locality Impacts

❋ *Memory access speed is not characterized*

C, Assembly, & Machine Code

C Function	Assembly (AT&T)	Machine (hex)
`int accum = 0;`		
`int sum(int x, int y)`	`sum:`	
`{`	`pushl %ebp`	`55`
	`movl %esp, %ebp`	`89 e5`
	`movl 12(%ebp), %eax`	`8b 45 0C`
`int t = x + y;`	`addl 8(%ebp), %eax`	`03 45 08`
`accum += t;`	`addl %eax, accum`	`01 05 ?? ?? ?? ??`
`return t;`	`popl %ebp`	`5D`
`}`	`ret`	`C3`

C

→ What aspects of the machine does C hide from us?

Assembly (ASM)

→ What ISA (Instruction Set Architecture) are we studying?

→ What does assembly remove from C source?

→ Why Learn Assembly?

Machine Code (MC)

→ How many bytes long is an IA-32 instructions?

Low-Level View of Data

C's View

-
-

Machine's View

❋ *Memory contains bits that do not*

→ How does a machine know what it's getting from memory?

1.

2.

Assembly Data Formats

C	IA-32	Assembly Suffix	Size in bytes
`char`	byte		
`short`	word		
`int`	double word		
`long int`	double word		
`char*`	double word		
`float`	single precision		
`double`	double prec		
`long double`	extended prec		

❋ *In IA-32 a word*

Registers

What? Registers

◆

◆

General Registers

	bit 31 … 16	15 … 8	7 … 0
`%eax`		`%ax` `%ah`	`%al`
`%ecx`		`%cx` `%ch`	`%cl`
`%edx`		`%dx` `%dh`	`%dl`
`%ebx`		`%bx` `%bh`	`%bl`
`%esi`		`%si`	
`%edi`		`%di`	
`%esp`		`%sp`	
`%ebp`		`%bp`	

Program Counter `%eip`

Condition Code Registers

Instructions - MOV, PUSH, POP

What? These are instructions to

Why?

How?

instruction class	operation	description
MOV S, D		
MOVS S, D		
MOVZ S, D		
pushl S		
popl D		

Practice with Data Formats

→ What data format suffix should replace the _ given the registers used?

```
1. mov_  %eax, %esp
2. push_  $0xFF
3. mov_  (%eax), %dx
4. mov_  (%esp, %edx, 4), %dh
5. mov_  0x800AFFE7, %bl
6. mov_  %dx, (%eax)
7. pop_  %edi
```