High-Performance Scientific Computing
Lecture 8: Single-thread Performance

MATH-GA 2011 / CSCI-GA 2945 · October 24, 2012
Today

Tool of the day: Installing software

Closer to the machine

Making things go faster
Bits and pieces

- HW4: tonight / early tomorrow
- HW6: due Saturday (ask for ext’n early)
- Last homework → project work after that
- Might issue problem sets for entertainment
Outline

Tool of the day: Installing software

Closer to the machine

Making things go faster
Demo time
Outline

Tool of the day: Installing software

Closer to the machine
   Machine Language
   Memory

Making things go faster
A Basic Processor

- Address ALU
- Register File
- Flags
- Data ALU
- Address ALU
- Memory Interface
- Address Bus
- Data Bus
- Internal Bus
- Control Unit
- PC
- Data ALU

Insn. fetch

(loosely based on Intel 8086)

Software Closer to the machine Faster
A Basic Processor

- Address ALU
- Register File
- Flags
- Data ALU
- Address ALU
- Internal Bus
- Memory Interface
- Address Bus
- Data Bus
- Data ALU

Bonus Question:
What's a bus?

(loosely based on Intel 8086)
Tool of the day: Installing software

Closer to the machine
  Machine Language
  Memory

Making things go faster
**A Very Simple Program**

```c
int a = 5;
int b = 17;
int z = a * b;
```

### Things to know:

- **Addressing modes** (Immediate, Register, Base plus Offset)
- **0xHexadecimal**
- “AT&T Form”: (we’ll use this)
  ```
  <opcode><size> <source>, <dest>
  ```
Software Closer to the machine Faster
4: c7 45 f4 05 00 00 00 movl $0x5,−0xc(%rbp)
b: c7 45 f8 11 00 00 00 movl $0x11,−0x8(%rbp)
12: 8b 45 f4 mov −0xc(%rbp),%eax
15: 0f af 45 f8 imul −0x8(%rbp),%eax
19: 89 45 fc mov %eax,−0x4(%rbp)
1c: 8b 45 fc mov −0x4(%rbp),%eax

Software Closer to the machine Faster
"Intel Form": (you might see this on the net) 
<opcode> <sized dest>, <sized source>

Goal: Reading comprehension.

Don’t understand an opcode? 
Google “<opcode> intel instruction".
int main()
{
    int y = 0, i;
    for (i = 0; y < 10; ++i)
        y += i;
    return y;
}

Things to know:

- **Condition Codes (Flags)**: Zero, Sign, Carry, etc.
- **Call Stack**: Stack frame, stack pointer, base pointer
- **ABI**: Calling conventions
Web demo

http://assembly.ynh.io/demo
demo time
Other web-based assembly viewers

- http://assembly.ynh.io/
  [https://github.com/ynh/cpp-to-assembly]
- http://gcc.godbolt.org/
- http://llvm.org/demo/
DIY demo

Assembly comprehension/optimizer
Outline

Tool of the day: Installing software

Closer to the machine
  Machine Language
  Memory

Making things go faster
What is... a Memory Interface?

**Memory Interface** gets and stores binary words in off-chip memory.

Smallest granularity: Bus width

Tells outside memory
- “where” through *address bus*
- “what” through *data bus*

Computer main memory is “Dynamic RAM” (*DRAM*): Slow, but small and cheap.
How does computer memory work?

One (reading) memory transaction (simplified):

Observation: Access (and addressing) happens in bus-width-size "chunks".
How does computer memory work?

One (reading) memory transaction (simplified):

Processor

Memory

D0..15

A0..15

R/\bar{W}

CLK

Observation: Access (and addressing) happens in bus-width-size "chunks".
How does computer memory work?

One (reading) memory transaction (simplified):

Processor <-> Memory

Observation: Access (and addressing) happens in bus-width-size "chunks".
How does computer memory work?

One (reading) memory transaction (simplified):

Processor → Memory

CLK
R/W
A0..15
D0..15

Observation: Access (and addressing) happens in bus-width-size “chunks”.

Software  Closer to the machine  Faster
How does computer memory work?

One (reading) memory transaction (simplified):

![Diagram showing processor and memory connected with signals D0..15, A0..15, R/∅, CLK, illustrating memory access and addressing in bus-width-size “chunks.”]
How does computer memory work?

One (reading) memory transaction (simplified):
How does computer memory work?

One (reading) memory transaction (simplified):

Observation: Access (and addressing) happens in bus-width-size “chunks”.

Software Closer to the machine Faster
DRAM

Software  Closer to the machine  Faster
Key: each cell is tiny → many of them!
Samsung 1 Gib DDR3 die
Tool of the day: Installing software

Closer to the machine

Making things go faster
  Overview
  The Memory Hierarchy
  Pipelines
  How about actually doing work?
Tool of the day: Installing software

Closer to the machine

Making things go faster

Overview

The Memory Hierarchy

Pipelines

How about actually doing work?
We know how a computer works!

All of this can be built in about 4000 transistors.
(e.g. MOS 6502 in Apple II, Commodore 64, Atari 2600)

So what exactly is Intel doing with the other 623,996,000 transistors?

Answer:
We know how a computer works!

All of this can be built in about 4000 transistors. (e.g. MOS 6502 in Apple II, Commodore 64, Atari 2600)

So what exactly is Intel doing with the other 623,996,000 transistors?

Answer: *Make things go faster!*
Go-fast widgets

All this go-faster technology: **hard to see**.

Most of the time:
- program fast,
- programmer happy.

Sometimes that’s not the case.
All this go-faster technology: **hard to see.**

Most of the time:
- program fast,
- programmer happy.

Sometimes that’s not the case.

**Goal now:** Break each widget in an understandable way.
Outline

Tool of the day: Installing software

Closer to the machine

Making things go faster
  Overview
  The Memory Hierarchy
  Pipelines
  How about actually doing work?
Source of Slowness: Memory

Memory is slow.

Distinguish two different versions of “slow”:

- Bandwidth
- Latency

→ Memory has long latency, but can have large bandwidth.

Size of die vs. distance to memory: big!

Dynamic RAM: long intrinsic latency!
Source of Slowness: Memory

Memory is slow.

Distinguish two different versions of “slow”:
- Bandwidth
- Latency

→ Memory has long latency, but can have large bandwidth.

Size of die vs. distance to memory: big!

Dynamic RAM: long intrinsic latency!

Idea:
Put a look-up table of recently-used data onto the chip.

→ “Cache”
The Memory Hierarchy

Hierarchy of increasingly bigger, slower memories:

- Registers: 1 kB, 1 cycle
- L1 Cache: 10 kB, 10 cycles
- L2 Cache: 1 MB, 100 cycles
- DRAM: 1 GB, 1000 cycles
- Virtual Memory (hard drive): 1 TB, 1 M cycles
The Memory Hierarchy

Hierarchy of increasingly bigger, slower memories:

- Registers
  - 1 kB, 1 cycle
- L1 Cache
  - 10 kB, 10 cycles
- L2 Cache
  - 1 MB, 100 cycles
- DRAM
- Virtual Memory (hard drive)

Second red/blue pebble game: played by cache controller

What is a *working set*?

How might *data locality* factor into this?
Cache: Actual Implementation

Demands on cache implementation:
- Fast, small, cheap, low power
- Fine-grained
- High “hit”-rate (few “misses”)

Problem:
Goals at odds with each other: Access matching logic expensive!

Solution 1: More data per unit of access matching logic
→ Larger “Cache Lines”

Solution 2: Simpler/less access matching logic
→ Less than full “Associativity”

Other choices: Eviction strategy, size
Cache: Associativity

Direct Mapped

<table>
<thead>
<tr>
<th>Memory</th>
<th>Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

Software Closer to the machine Faster
Cache: Associativity

Direct Mapped

<table>
<thead>
<tr>
<th>Memory</th>
<th>Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

2-way set associative

<table>
<thead>
<tr>
<th>Memory</th>
<th>Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
</tr>
</tbody>
</table>

Miss rate versus cache size on the Integer portion of SPEC CPU2000 [Cantin, Hill 2003]

Software Closer to the machine Faster
Cache: Associativity

Direct Mapped

<table>
<thead>
<tr>
<th>Memory</th>
<th>Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

2-way set associative

<table>
<thead>
<tr>
<th>Memory</th>
<th>Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

Miss rate versus cache size on the Integer portion of SPEC CPU2000 [Cantin, Hill 2003]
Cache: Associativity

Direct Mapped

<table>
<thead>
<tr>
<th>Memory</th>
<th>Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

2-way set associative

<table>
<thead>
<tr>
<th>Memory</th>
<th>Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Cache: Associativity

Direct Mapped

2-way set associative

Miss rate versus cache size on the Integer portion of SPEC CPU2000 [Cantin, Hill 2003]

Software Closer to the machine Faster
Cache: Associativity

Direct Mapped

<table>
<thead>
<tr>
<th>Memory</th>
<th>Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

... 

2-way set associative

<table>
<thead>
<tr>
<th>Memory</th>
<th>Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

... 

Software  Closer to the machine  Faster
Cache: Associativity

- **Direct Mapped**

  - Memory:
    - 0
    - 1
    - 2
    - 3
    - 4
    - 5
    - 6
    - ...
  
  - Cache:
    - 0
    - 1
    - 2
    - 3

- **2-way set associative**

  - Memory:
    - 0
    - 1
    - 2
    - 3
    - 4
    - 5
    - 6
    - ...
  
  - Cache:
    - 0
    - 1
    - 2
    - 3

Miss rate versus cache size on the Integer portion of SPEC CPU2000 [Cantin, Hill 2003]

Software  Closer to the machine  Faster
Cache: Associativity

Direct Mapped

<table>
<thead>
<tr>
<th>Memory</th>
<th>Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

2-way set associative

<table>
<thead>
<tr>
<th>Memory</th>
<th>Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Cache: Associativity

Direct Mapped

<table>
<thead>
<tr>
<th>Memory</th>
<th>Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

2-way set associative

<table>
<thead>
<tr>
<th>Memory</th>
<th>Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

Miss rate versus cache size on the Integer portion of SPEC CPU2000 [Cantin, Hill 2003]

Software Closer to the machine Faster
Miss rate versus cache size on the Integer portion of SPEC CPU2000 [Cantin, Hill 2003]
CPUID demo time
Updating every \( k \)th integer

```c
int go(unsigned count, unsigned stride )
{
    const unsigned array_size = 64 * 1024 * 1024;
    int *ary = (int *) malloc( sizeof(int) * array_size );

    for (unsigned it = 0; it < count; ++it)
    {
        for (unsigned i = 0; i < array_size ; i += stride)
            ary[i] *= 17;
    }

    int result = 0;
    for (unsigned i = 0; i < array_size ; ++i)
        result += ary[i];

    free(ary);
    return result ;
}
```

Original benchmarks by Igor Ostrovsky
Updating every $k$th integer

```c
int go(unsigned count, unsigned stride)
{
    const unsigned array size = 64 * 1024 * 1024;
    int *ary = (int *) malloc(sizeof(int) * array size);
    for (unsigned it = 0; it < count; ++it)
    {
        for (unsigned i = 0; i < array size; i += stride)
        {
            ary[i] = 17;
        }
    }
    int result = 0;
    for (unsigned i = 0; i < array size; ++i)
    {
        result += ary[i];
    }
    free(ary);
    return result;
}
```

Original benchmarks by Igor Ostrovsky

<table>
<thead>
<tr>
<th>Stride</th>
<th>Time [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
<td>0.08</td>
</tr>
<tr>
<td>0.01</td>
<td>0.07</td>
</tr>
<tr>
<td>0.02</td>
<td>0.06</td>
</tr>
<tr>
<td>0.03</td>
<td>0.05</td>
</tr>
<tr>
<td>0.04</td>
<td>0.04</td>
</tr>
<tr>
<td>0.05</td>
<td>0.03</td>
</tr>
<tr>
<td>0.06</td>
<td>0.02</td>
</tr>
<tr>
<td>0.07</td>
<td>0.01</td>
</tr>
<tr>
<td>0.08</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Software Closer to the machine Faster
Measuring bandwidths

```c
int go(unsigned array_size, unsigned steps)
{
    int *ary = (int *) malloc(sizeof(int) * array_size);
    unsigned asm1 = array_size - 1;

    for (unsigned i = 0; i < 100*steps;)
    {
        #define ONE ary[(i++*16) & asm1] ++;
        #define FIVE ONE ONE ONE ONE ONE
        #define TEN FIVE FIVE
        #define FIFTY TEN TEN TEN TEN TEN
        #define HUNDRED FIFTY FIFTY HUNDRED
    }

    int result = 0;
    for (unsigned i = 0; i < array_size; ++i)
        result += ary[i];

    free(ary);
    return result;
}
```

Original benchmarks by Igor Ostrovsky
Measuring bandwidths

```c
int go(unsigned array_size, unsigned steps)
{
    int *ary = (int *) malloc(sizeof(int) * array_size);
    unsigned asm1 = array_size - 1;
    for (unsigned i = 0; i < 100 * steps;)
    {
        #define ONE ary[(i++ * 16) & asm1] ++;
        #define FIVE ONE ONE ONE ONE ONE
        #define TEN FIVE FIVE
        #define FIFTY TEN TEN TEN TEN TEN
        #define HUNDRED FIFTY FIFTY HUNDRED
    }
    int result = 0;
    for (unsigned i = 0; i < array_size; ++i)
        result += ary[i];
    free(ary);
    return result;
}
```

Original benchmarks by Igor Ostrovsky
Another mystery

```c
int go(unsigned array_size, unsigned stride, unsigned steps)
{
    char *ary = (char *) malloc(sizeof(int) * array_size);

    unsigned p = 0;
    for (unsigned i = 0; i < steps; ++i)
    {
        ary[p] ++;
        p += stride;
        if (p >= array_size)
            p = 0;
    }

    int result = 0;
    for (unsigned i = 0; i < array_size; ++i)
    {
        result += ary[i];
    }

    free(ary);
    return result ;
}
```

Original benchmarks by Igor Ostrovsky

Software  Closer to the machine  Faster
```c
int go(unsigned array_size, unsigned stride, unsigned steps)
{
    char *ary = (char *) malloc(sizeof(int) * array_size);
    unsigned p = 0;
    for (unsigned i = 0; i < steps; ++i)
    {
        ary[p] ++;
        p += stride;
        if (p == array_size)
            p = 0;
    }
    int result = 0;
    for (unsigned i = 0; i < array_size; ++i)
        result += ary[i];
    free(ary);
    return result;
}
```

Original benchmarks by Igor Ostrovsky

<table>
<thead>
<tr>
<th>Array Size [MB]</th>
<th>Stride [bytes]</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>5</td>
</tr>
<tr>
<td>200</td>
<td>10</td>
</tr>
<tr>
<td>300</td>
<td>15</td>
</tr>
<tr>
<td>400</td>
<td>20</td>
</tr>
<tr>
<td>500</td>
<td></td>
</tr>
<tr>
<td>600</td>
<td></td>
</tr>
</tbody>
</table>

Software Closer to the machine Faster
Learned a lot about caches.

Also learned:

Honest measurements are *hard*.

A good attempt:
http://www.bitmover.com/lmbench/

Instructions:
Programming for the Hierarchy

How can we rearrange programs to friendly to the memory hierarchy?

Examples:

- Large vectors $x$, $a$, $b$

  Compute

  $$x \leftarrow x + 3a - 5b.$$
Programming for the Hierarchy

How can we rearrange programs to friendly to the memory hierarchy?

Examples:

- Large vectors $x, a, b$
  
  Compute

  $$x ← x + 3a − 5b.$$  

- Matrix-Matrix Multiplication
Tool of the day: Installing software

Closer to the machine

Making things go faster
  Overview
  The Memory Hierarchy
  Pipelines
  How about actually doing work?
Source of Slowness: Sequential Operation

- **IF**: Instruction fetch
- **ID**: Instruction Decode
- **EX**: Execution
- **MEM**: Memory Read/Write
- **WB**: Result Writeback

Software Closer to the Machine: Faster
Solution: Pipelining

Software Closer to the machine Faster
Pipelining

(MIPS, 110,000 transistors)
Pipelines generally help performance—but not always.

Possible issue: Dependencies... 

- ...on memory
- ...on previous computation
- ...on branch outcomes

“Solution”: Bubbling
Issues with Pipelines

Pipelines generally help performance—but not always.

Possible issue: Dependencies...
- ...on memory
- ...on previous computation
- ...on branch outcomes

“Solution”: Bubbling

For branches: could guess...?
Performance mystery demo time
New concept: Instruction-level parallelism ("Superscalar")

Software Closer to the machine Faster
New concept: Instruction-level parallelism ("Superscalar")
More Pipeline Mysteries
Outline

Tool of the day: Installing software

Closer to the machine

Making things go faster
  Overview
  The Memory Hierarchy
  Pipelines
  How about actually doing work?
Floating point performance demo
Questions?
Image Credits

- DRAM: Wikipedia (CC)
- DRAM die: chipworksrealchips.com / Samsung
- Basic cache: Wikipedia (CC)
- Cache associativity: based on Wikipedia (CC)
- Cache associativity vs miss rate: Wikipedia (CC)
- Cache Measurements: Igor Ostrovsky
- Pipelining: Wikipedia (CC)
- Bubbly Pipeline: Wikipedia (CC)