Memory Hierarchy (I)

Hung-Wei Tseng
Outline

• Memory wall/gap problem
• Memory hierarchy
• Cache organization
Memory wall problem
The memory gap problem

CPU

DRAM-based main memory

lw $t2, 0($a0)
add $t3, $t2, $a1
addi $a0, $a0, 4
subi $a1, $a1, 1
bne $a1, LOOP
lw $t2, 0($a0)
add $t3, $t2, $a1

The access time of DRAM is around 50ns
100x to the cycle time of a 2GHz processor!

SRAM is as fast as the processor, but $$$

<table>
<thead>
<tr>
<th>Memory technology</th>
<th>Typical access time</th>
<th>$ per GiB in 2012</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRAM semiconductor memory</td>
<td>0.5–2.5 ns</td>
<td>$500–$1000</td>
</tr>
<tr>
<td>DRAM semiconductor memory</td>
<td>50–70 ns</td>
<td>$10–$20</td>
</tr>
<tr>
<td>Flash semiconductor memory</td>
<td>5,000–50,000 ns</td>
<td>$0.75–$1.00</td>
</tr>
<tr>
<td>Magnetic disk</td>
<td>5,000,000–20,000,000 ns</td>
<td>$0.05–$0.10</td>
</tr>
</tbody>
</table>
Memory’s impact

- Assume your processor takes only 1 cycle to process an instruction if the DRAM is the same speed as the processor. In fact, the latency of DRAM is 100 cycles.
  - Ignore all virtual memory overheads and processor optimizations
  - The application contains 20% instructions that perform data memory accesses
  - How many cycles do you need to process an instruction on average?
    A. ~ 10
    B. ~ 20
    C. ~ 100
    D. ~ 120
    E. ~ 200

\[
\text{average} = 1 + 1 \times 100 + 0.2 \times 100 = 121
\]
Why is C better than B

• How many of the following statements explains the reason why B outperforms C with compiler optimizations

① C has lower dynamic instruction count than B
  — C only needs one load, one add, one shift, the same amount of iterations
  — the same number being predicted.
  — the same amount of branches
  — Probably not. In fact, the load may have negative effect without architectural supports

② C has significantly lower branch mis-predictions than B

③ C has significantly fewer branch instructions than B

④ C can incur fewer data hazards

A. 0
B. 1
C. 2
D. 3
E. 4

Does this make sense if memory is so slow?
Memory hierarchy
The memory hierarchy

- **CPU**
- **Main Memory**
- **Secondary Storage**

- Fastest, Most Expensive
- Access time:
  - CPU: < 1ns
  - Cache: < 1ns ~ 20 ns
  - Main Memory: 100ns
  - Secondary Storage: 10,000,000ns

- Sizes:
  - CPU: 32*, 64-bit registers
  - L1: 16KB-64KB
  - L2: 128KB-512KB
  - L3: Several MBs
  - Secondary Storage: Several GBs, 500+ GB

- Cost:
  - CPU: $< 1ns ~ 20 ns
By how much can cache help?

- Assume your processor takes only 1 cycle to process an instruction if the DRAM is the same speed as the processor. In fact, the latency of DRAM is 100 cycles.
  - Now, if we add a cache between CPU and DRAM. If 90% of time, the data/instruction can be found in the cache — no additional cycles is wasted to fetch the data/instruction.
  - If the data is missing in the cache, it takes 100 cycles to retrieve data from the DRAM
- You may ignore all virtual memory overheads and processor optimizations
- The application contains 20% instructions that perform data memory accesses
- How many cycles do you need to process an instruction on average?

A. ~ 10  
B. ~ 20  
C. ~ 100  
D. ~ 120  
E. ~ 200

\[
\text{average} = 1 + (1\times100 + 0.2\times100)\times(1-90\%) = 13
\]
Why can a small, fast SRAM help?
Localities in your code

- Which description about locality of arrays \texttt{sum} and \texttt{A} in the following code is the most accurate?

```c
for(i = 0; i< 100000; i++)
{
    \texttt{sum}[i\%10] += \texttt{A}[i];
}
```

A. Access of \texttt{A} has temporal locality, \texttt{sum} has spatial locality
B. Both \texttt{A} and \texttt{sum} have temporal locality, and \texttt{sum} also has spatial locality
C. Access of \texttt{A} has spatial locality, \texttt{sum} has temporal locality
D. Both \texttt{A} and \texttt{sum} have spatial locality
E. Both \texttt{A} and \texttt{sum} have spatial locality, and \texttt{sum} also has temporal locality
Localities in your code

• Spatial locality: programs tend to access neighboring data/instructions
  • Data structures (e.g. arrays) demonstrate strong spatial locality
  • Especially effective for code/instructions — you usually just move to the next instruction or loop back to the small piece of code

• Temporal locality: programs tend to have frequently accessed data
  • You may update/reference the same set of memory locations many times in your code
Cache organization
Architecting caches to capture localities

- To capture spatial locality
  - We need to put not only just a “word” or small piece of data/instructions, but a “block” of data/instructions

- To capture temporal locality
  - We need to keep frequently used data
Organizing memory locations into blocks
Architecting caches to capture localities

• To capture spatial locality
  • We need to put not only just a “word” or small piece of data/instructions, but a “block” of data/instructions
  • How to distinguish each block?

• To capture temporal locality
  • We need to keep frequently used data
How do you make a cheatsheet?

- Go through your homework
- Write down the topic and content
- If running out of space: kick out the least recently used content

1. Performance equation
2. Amdahl’s law
3. MIPS
4. Power consumption
5. Performance equation 😊
6. Amdahl’s law 😊
7. MFLOPS

Tag: the address prefix of data in the cacheline/block

<table>
<thead>
<tr>
<th>Performance equation</th>
<th>ET = IC * CPI * CT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amdahl’s law</td>
<td>ET_after = ET_affected/Speedup + ET_unaffected</td>
</tr>
<tr>
<td>MFLOPS</td>
<td>MIPS = f(CET10^6)</td>
</tr>
<tr>
<td>Power consumption</td>
<td>P = aCV^2f</td>
</tr>
</tbody>
</table>

Cacheline/block: data with the same prefix in their addresses
A simple cache: now with tags associated with blocks

- Assume each block contains 16B data
- A total of 4 blocks

<table>
<thead>
<tr>
<th>tag</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>content of 0b00000000 - 0b00001111</td>
</tr>
<tr>
<td>0b0100</td>
<td>content of 0b01000000 - 0b01001111</td>
</tr>
<tr>
<td>0b1100</td>
<td>content of 0b11000000 - 0b11001111</td>
</tr>
<tr>
<td>0b1111</td>
<td>content of 0b11110000 - 0b11111111</td>
</tr>
</tbody>
</table>
Architecting caches to capture localities

• To capture spatial locality
  • We need to put not only just a “word” or small piece of data/instructions, but a “block” of data/instructions
  • A tag associated with each block

• To capture temporal locality
  • A cache replacement policy to keep most frequently used data (e.g. LRU)
  • LRU — kick out the least recently used block when we need to kick out one
A simple cache: a block can go anywhere

- Assume each block contains 16B data
- A total of 4 blocks
- LRU — kick out the least recently used whenever we need to

<table>
<thead>
<tr>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x4</td>
<td>0b00000100</td>
</tr>
<tr>
<td>0x48</td>
<td>0b01001000</td>
</tr>
<tr>
<td>0xC4</td>
<td>0b11000100</td>
</tr>
<tr>
<td>0xFC</td>
<td>0b11111100</td>
</tr>
<tr>
<td>0x12</td>
<td>0b00001100</td>
</tr>
<tr>
<td>0x44</td>
<td>0b01000100</td>
</tr>
<tr>
<td>0x68</td>
<td>0b01100100</td>
</tr>
</tbody>
</table>

- Too slow if the number of entries/blocks/cachelines is huge
Architecting caches to capture localities

- To capture spatial locality
  - We need to put not only just a “word” or small piece of data/instructions, but a “block” of data/instructions
  - A tag associated with each block

- To capture temporal locality
  - A cache replacement policy to keep most frequently used data (e.g. LRU)
  - LRU — kick out the least recently used block when we need to kick out one

- Performance needs to be better than linear search
  - Make cache a hardware hash table!
  - The hash function takes memory addresses as inputs
The structure of a cache

**Set:** cache blocks/lines sharing the same index. A cache is called N-way set associative cache if N blocks share the same set/index (this one is a 2-way set cache)

**Tag:** the high order address bits stored along with the data in a block to identify the actual address of the cache line.

**Valid:** if the data is meaningful

**Dirty:** if the block is modified

**Block / Cacheline:** The basic unit of data storage in cache. Contains all data with the same tag/prefix and index in their memory addresses
Accessing the cache

Hit: The data was found in the cache
Miss: The data was not found in the cache

Hit? miss?

Offset: The position of the requesting word in a cache block
Announcement

• Homework #3 due next Monday
  • Will drop one of the lowest homework grade
• Reading quiz due next Wednesday
  • Will drop one of the lowest quiz grade
• Check your grades online