Multilevel Caches, Cache Question

Instructor: Nick Riasanovsky
Great Idea #3: Principle of Locality/Memory Hierarchy

- Processor
  - Processor Register
  - CPU Cache
    - Level 1 (L1) Cache
    - Level 2 (L2) Cache
    - Level 3 (L3) Cache
  - Physical Memory
    - Random Access Memory (RAM)
  - Solid State Memory
    - Non-Volatile Flash-Based Memory
  - Virtual Memory
    - File-Based Memory
  - EDO, SD-RAM, DDR-SDRAM, RD-RAM and More...
  - SSD, Flash Drive
  - Mechanical Hard Drives

- Super Fast
  - Super Expensive
  - Tiny Capacity
- Faster
  - Expensive
  - Small Capacity
- Fast
  - Priced Reasonably
  - Average Capacity
- Average Speed
  - Priced Reasonably
  - Average Capacity
- Slow
  - Cheap
  - Large Capacity
Review of Last Lecture

• Direct-Mapped Caches:
  – Use hash function to determine location for block
    • Each block maps into a single row
    • \((\text{block address}) \mod (\# \text{ of slots in the cache})\)

• N-way Set Associative Caches:
  – Split slots into sets of size \(N\), map into set
    • \((\text{block address}) \mod (\# \text{ of sets in the cache})\)

• TIO breakdown of memory address
  – Index field is result of hash function (which set)
  – Tag field is identifier (which block is currently in slot)
  – Offset field indexes into block
Worst Case for Set Associative

- Worst case for DM was repeating pattern of 2 into same cache slot (HR = 0/n)
  - Set associative for N > 1: HR = (n-2)/n
- Worst case for N-way SA with LRU?
  - Repeating pattern of at least N+1 that maps into same set
  - Back to HR = 0:

```
0, 8, 16, 0, 8, M M M M M
000 M[0-3]
001 M[8-11]
010 M[16-19]
```
**Question:** How many total bits are stored in the following cache?

- 4-way SA cache, random replacement
- Cache size 1 KiB, Block size 16 B
- Write-back
- 16-bit address space

(A) \(2^6 \times (2^7 + 2^3 + 2^1) = 8.625 \text{ Kib}\)

(B) \(2^4 \times (2^7 + 2^3 + 2^0) = 2.140625 \text{ Kib}\)

(C) \(2^4 \times (2^7 + 2^3 + 2^1) = 2.15625 \text{ Kib}\)

(D) \(2^4 \times (2^7 + 6 + 2^1) = 2.125 \text{ Kib}\)
Question: How many total bits are stored in the following cache?

- 4-way SA cache, random replacement
- Cache size 1 KiB, Block size 16 B
- Write-back dirty bit
- 16-bit address space

(A) $2^6 \times (2^7 + 2^3 + 2^1) = 8.625 \text{ Kib}$

(B) $2^6 \times (2^7 + 2^3 + 2^0) = 2.140625 \text{ Kib}$

(C) $2^4 \times (2^7 + 2^3 + 2^1) = 2.15625 \text{ Kib}$

(D) $2^4 \times (2^7 + 6 + 2^1) = 2.125 \text{ Kib}$

# slots?
- data?
- tag?
- valid?
- dirty?
Question: How many total bits are stored in the following cache?

- 4-way SA cache, random replacement
- Cache size 1 KiB, Block size 16 B
- Write-back
- 16-bit address space

\[
\text{(A)} \quad 2^6 \times (2^7 + 2^3 + 2^1) = 8.625 \text{ Kib} \\
\text{(B)} \quad 2^6 \times (2^7 + 2^3 + 2^0) = 2.140625 \text{ Kib} \\
\text{(C)} \quad 2^4 \times (2^7 + 2^3 + 2^1) = 2.15625 \text{ Kib} \\
\text{(D)} \quad 2^4 \times (2^7 + 6 + 2^1) = 2.125 \text{ Kib}
\]
Cache Performance

• Two things hurt the performance of a cache:
  – Miss rate and miss penalty
  – Average Memory Access Time (AMAT): average time to access memory considering both hits and misses

  \[
  \text{AMAT} = \text{Hit time} + \text{Miss rate} \times \text{Miss penalty}
  \]
  (abbreviated AMAT = HT + MR \times MP)

• **Goal 1:** Examine how changing the different cache parameters affects our AMAT

• **Goal 2:** Examine how to optimize your code for better cache performance (Project 4)
AMAT Example

• **Processor specs:** 200 ps clock, MP of 50 clock cycles, MR of 0.02 misses/instruction, and HT of 1 clock cycle

  \[ \text{AMAT} = 1 + 0.02 \times 50 = 2 \text{ clock cycles} = 400 \text{ ps} \]

• Which improvement would be best?
  
  – 190 ps clock
  – MP of 40 clock cycles
  – MR of 0.015 misses/instruction
Cache Parameter Example

• What is the potential impact of much larger cache on AMAT? (same block size)
  1) Increase HR
  2) Longer HT: smaller is faster
     – At some point, increase in hit time for a larger cache may overcome the improvement in hit rate, yielding a decrease in performance
Sources of Cache Misses: The 3Cs

• **Compulsory:** (Many names: cold start, process migration (switching processes), 1\(^{st}\) reference)
  – First access to block impossible to avoid;
  Effect is small for long running programs

• **Capacity:**
  – Cache cannot contain all blocks accessed by the program, so full associativity won’t hold all blocks

• **Conflict:** (collision)
  – Multiple memory locations mapped to the same cache location, so there’s a lack of associativity
The 3Cs: Design Solutions

• Compulsory:
  – Increase block size (increases MP; too large blocks could increase MR)

• Capacity:
  – Increase cache size (may increase HT)

• Conflict:
  – Increase associativity (to fully associative) (may increase HT)
Review

• Cache performance measured using AMAT
  – Parameters that matter:
    ● Hit Time (HT)
    ● Miss Rate (MR)
    ● Miss Penalty (MP)
  – AMAT = Hit Time + Miss Rate x Miss Penalty

• The 3 Cs of cache misses and their fixes
  – Compulsory: Increase block size
  – Capacity: Increase cache size
  – Conflict: Make the cache fully associative
Agenda

• Multilevel Caches
• Administrivia
• Improving Cache Performance
• Anatomy of a Cache Question
• Example Cache Questions
• Bonus: Contemporary Cache Specs
Multiple Cache Levels

• With advancing technology, have more room on chip for bigger L1 caches and for L2 (and in some cases even L3) cache
  – Normally lower-level caches are unified (i.e. holds both instructions and data)
  – Higher numbered caches are lower-level (closer to physical mem)
• Multilevel caching is a way to reduce miss penalty
• So what does this look like?
Multilevel Cache Diagram

Legend:
- Purple: Request for data
- Orange: Return of data

CPU → L1$ (Memory Access)

L1$ → L2$ (Write Miss) → Main Memory (Write Miss)

If Write Allocate:

Path of data back to CPU:
- Hit: L1$ → L2$ → Main Memory
- Miss: L1$ → L2$ → Main Memory

Legend:
- Request for data
- Return of data

7/17/2018 CS61C Su18 - Lecture 16
Multilevel Cache AMAT

- AMAT = L1 HT + L1 MR × L1 MP
  - Now L1 MP depends on other cache levels
- L1 MP = L2 HT + L2 MR × L2 MP
  - If more levels, then continue this chain (i.e. $MP_i = HT_{i+1} + MR_{i+1} × MP_{i+1}$)
  - Final MP is main memory access time

- For two levels:
  \[
  AMAT = L1 HT + L1 MR × (L2 HT + L2 MR × L2 MP)
  \]
Multilevel Cache AMAT Example

• **Processor specs:** 1 cycle L1 HT, 2% L1 MR, 5 cycle L2 HT, 5% L2 MR, 100 cycle main memory HT
  – Here assuming unified L1$'

• Without L2$:
  \[
  \text{AMAT}_1 = 1 + 0.02 \times 100 = 3
  \]

• With L2$:
  \[
  \text{AMAT}_2 = 1 + 0.02 \times (5 + 0.05 \times 100) = 1.2
  \]
Local vs. Global Miss Rates

- **Local miss rate**: Fraction of references to one level of a cache that miss
  - e.g. \( L_2 \) local MR = \( L_2 \) misses/\( L_1 \) misses
  - Specific to level of caching (as used in AMAT)

- **Global miss rate**: Fraction of all references that miss in all levels of a multilevel cache
  - Property of the overall memory hierarchy
  - Global MR is the **product of all local MRs**
    - Start at Global MR = \( L_n \) misses/\( L_{n-1} \) accesses all multiplied together
    - So by definition, **global MR ≤ any local MR**
Global Miss Rates

• We may also refer to the global miss rate of a particular level of cache
  – For example Global MR L2
  – This means the fraction of total accesses that miss at L1 and L2

• As a result we can sometimes talk about global miss rates without necessarily involving every level of cache
For every 1000 CPU-to-memory references

- 40 will miss in L1$; what is the local MR? 0.04
- 20 will miss in L2$; what is the local MR? 0.5
- Overall global miss rate? 0.02
Rewriting Performance

- For a two level cache, we know:
  \[ MR_{global} = L1\ MR \times L2\ MR \]

- AMAT:
  - \[ AMAT = L1\ HT + L1\ MR \times (L2\ HT + L2\ MR \times L2\ MP) \]
  - \[ = L1\ HT + L1\ MR \times L2\ HT + MR_{global} \times L2\ MP \]

- Aside: Sometimes might have to convert between global and local MR
  - \[ L2\ Global\ MR = L2\ Local\ MR \times L1\ MR \]
  - \[ L2\ Local\ MR = L2\ Global\ MR ÷ L1\ MR \]
Design Considerations

• L1$ focuses on *low hit time* (fast access)
  – minimize HT to achieve shorter clock cycle
  – L1 MP significantly reduced by presence of L2$, so can be smaller/faster even with higher MR
  – e.g. smaller $ (fewer rows)

• L2$, L3$ focus on *low miss rate*
  – As much as possible avoid reaching to main memory (heavy penalty)
  – e.g. larger $ with larger block sizes (same # rows)
Multilevel Cache Practice (1/3)

• Processor specs:
  – L1$ and L2$
  – 5 cycle L1$ hit time and 4% L1$ miss rate
  – 100 cycle penalty to go to main memory
  – 0.5% L2$ global miss rate
  – 25 cycle penalty to go to L2$

• What is AMAT?
Multilevel Cache Practice (2/3)

- L2 Local MR = L2 Global MR ÷ L1 MR

- Notice given global L2$ MR. Convert to local MR!
  
  - \( \frac{5}{1000} \) of ALL accesses are misses in L2$
    - Want numerator of this fraction as numerator
    - Denominator represents total accesses
  
  - \( \frac{4}{100} \) of ALL access go to L2$
    - Want numerator of this fraction as denominator
    - Denominator represents total accesses

  \[
  \frac{5}{1000} ÷ \frac{4}{100} = \frac{5}{1000} \times \frac{100}{4} = .125
  \]
**Multilevel Cache Practice (3/3)**

- **Without L2$:**
  \[
  AMAT = (5 + 0.04 \times 100) \\
  = 9 \text{ cycles}
  \]

- **With L2$:**
  \[
  AMAT = HT_{L1} + MR_{L1} \times (HT_{L2} + MR_{L2} \times MP_{L2}) \\
  = 5 + 0.04 \times (25 + 0.125 \times 100) \\
  = 6.5 \text{ cycles}
  \]
Agenda

• Multilevel Caches

• Administrivia

• Improving Cache Performance

• Anatomy of a Cache Question

• Example Cache Questions

• Bonus: Contemporary Cache Specs
Administrivia

• Proj1 and HW3/4 scores now live!
• HW5 due 7/23, Proj-3 due 7/20
• Proj 3 party on Fri (7/20), 4-6PM @Woz
• Guerilla Session on Wed. 4-6pm @Soda 405
• Midterm 2 is coming up! Next Wed. in lecture
  – Covering up to Performance
  – Review Session Sunday 2-4pm @GPB 100
  – There will be discussion after MT2 :(  

7/17/2018
CS61C Su18 - Lecture 16
Agenda

- Multilevel Caches
- Administrivia
  - Improving Cache Performance
  - Anatomy of a Cache Question
  - Example Cache Questions
- Bonus: Contemporary Cache Specs
Improving Cache Performance (1/2)

1) Reduce the Hit Time of the cache
   – Smaller cache (less to search/check)
   – Smaller blocks (faster to return selected data)

2) Reduce the Miss Rate
   – Bigger cache (capacity)
   – Larger blocks (compulsory & spatial locality)
   – Increased associativity (conflict)
Improving Cache Performance (2/2)

3) Reduce the Miss Penalty
   – Smaller blocks (less to move)
   – Use multiple cache levels
   – Use a write buffer
The Cache Design Space

Several interacting dimensions

- **Cache parameters:**
  - Cache size, Block size, Associativity
- **Policy choices:**
  - Write-through vs. write-back
  - Replacement policy
- **Optimal choice is a compromise**
  - Depends on access characteristics
    - Workload and use (I$, D$)
  - Depends on technology / cost
- **Simplicity often wins**
Effect of Block and Cache Sizes on Miss Rate

- Miss rate goes up if the block size becomes a significant fraction of the cache size because the number of blocks that can be held in the same size cache is smaller (increasing capacity misses)
Benefits of Set-Associative Caches

- Consider cost of a miss vs. cost of implementation
- Largest gains are in going from direct mapped to 2-way (20%+ reduction in miss rate)
Agenda

• Multilevel Caches
• Administrivia
• Improving Cache Performance
• Anatomy of a Cache Question
• Example Cache Questions
• Bonus: Contemporary Cache Specs
Anatomy of a Cache Question

• Cache questions come in a few flavors:
  1) TIO Breakdown
  2) For fixed cache parameters, analyze the performance of the given code/sequence
  3) For fixed cache parameters, find best/worst case scenarios
  4) For given code/sequence, how does changing your cache parameters affect performance?
  5) AMAT
The Cache

• What are the important cache parameters?
  – Must figure these out from problem description
  – Address size, cache size, block size, associativity, replacement policy
  – Solve for TIO breakdown, # of sets, set size

• Are there multiple levels?
  – Mostly applies to AMAT questions

• What starts in the cache?
  – Not always specified (best/worst case)
Code: Arrays

• Elements stored sequentially in memory
  – Ideal for spatial locality
  – Different arrays not necessarily next to each other

• Remember to account for data size!
  – char is 1 byte, int is 4 bytes

• Pay attention to access pattern
  – Touch all elements (e.g. shift, sum)
  – Touch some elements (e.g. histogram, stride)
  – How many times do we touch each element?
Code: Linked Lists/Structs

• Nodes stored separately in memory
  – Addresses of nodes may be very different
  – Type and ordering of linking is important
• Remember to account for size/ordering of struct elements
• Pay attention to access pattern
  – Generally must start from “head”
  – How many struct elements are touched?
Access Patterns

- How many hits within a single block once it is loaded into cache?
- Will block still be in cache when you revisit its elements?
- Are there special/edge cases to consider?
  - Usually edge of block boundary or edge of cache size boundary
### Meet the Staff

<table>
<thead>
<tr>
<th></th>
<th>Emaan</th>
<th>Sruthi</th>
<th>Sean</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Favorite Villain</strong></td>
<td>Syndrome</td>
<td>X11 Errors</td>
<td>Students who don't read specs</td>
</tr>
<tr>
<td><strong>What you’d protest</strong></td>
<td>Lack of educational funding</td>
<td>Harvard's CRISPR patent</td>
<td>Inaccessibility of Soda Hall</td>
</tr>
<tr>
<td><strong>Your passion</strong></td>
<td>61C Staff</td>
<td>Logisim Evolution</td>
<td>Politics</td>
</tr>
<tr>
<td><strong>What you'd be famous for?</strong></td>
<td>POTUS or SCOTUS</td>
<td>Special Investigator for Emaan</td>
<td>Emaan's Puppetmaster</td>
</tr>
</tbody>
</table>
Agenda

• Multilevel Caches
• Administrivia
• Improving Cache Performance
• Anatomy of a Cache Question
• Example Cache Questions
• Bonus: Contemporary Cache Specs
Example 1 (Sp07 Final)

a) 1 GiB address space, 100 cycles to go to memory. Fill in the following table:

<table>
<thead>
<tr>
<th></th>
<th>L1</th>
<th>L2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache Size</td>
<td>32 KiB</td>
<td>512 KiB</td>
</tr>
<tr>
<td>Block Size</td>
<td>8 B</td>
<td>32 B</td>
</tr>
<tr>
<td>Associativity</td>
<td>4-way</td>
<td>Direct-mapped</td>
</tr>
<tr>
<td>Hit Time</td>
<td>1 cycle</td>
<td>33 cycles</td>
</tr>
<tr>
<td>Miss Rate</td>
<td>10%</td>
<td>2%</td>
</tr>
<tr>
<td>Write Policy</td>
<td>Write-through</td>
<td>Write-through</td>
</tr>
<tr>
<td>Replacement Policy</td>
<td>LRU</td>
<td>n/a</td>
</tr>
<tr>
<td>Tag</td>
<td>17</td>
<td>11</td>
</tr>
<tr>
<td>Index</td>
<td>10</td>
<td>14</td>
</tr>
<tr>
<td>Offset</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>AMAT</td>
<td>AMAT L1 = 1 + 0.1 * 35 = 4.5</td>
<td>AMAT L2 = 33 + 0.02 * 100 = 35</td>
</tr>
</tbody>
</table>
Example 1 (Sp07 Final)

Only use L1$: $C = 32$ KiB, $K = 8$ B, $N = 4$, LRU, write-through
char A[] is block aligned and $\text{SIZE} = 32$ MiB

```c
char *A = (char *) malloc (SIZE*sizeof(char));
/* number of STRETCHes */
for (i = 0; i < (SIZE/STRETCH); i++) {
    /* go up to STRETCH */
    for(j=0; j<STRETCH; j++) sum += A[i*STRETCH+j];
    /* down from STRETCH */
    for(j=STRETCH-1; j>=0; j--) prod *= A[i*STRETCH+j];
}
```

- 2\textsuperscript{nd} inner for loop hits same indices as 1\textsuperscript{st} inner for loop, but in reverse order
- Always traverse full SIZE, regardless of STRETCH
Example 1 (Sp07 Final)

Only use L1$: $C = 32$ KiB, $K = 8$ B, $N = 4$, LRU, write-through
char $A[]$ is block aligned and $SIZE = 32$ MiB

```c
char *A = (char *) malloc (SIZE*sizeof(char));
for (i = 0; i < (SIZE/STRETCH); i++) {
    for(j=0;j<STRETCH;j++) sum += A[i*STRETCH+j];
    for(j=STRETCH-1;j>=0;j++) prod *= A[i*STRETCH+j];
}
```

b) As we double our STRETCH from 1 to 2 to 4 (...etc), we notice the number of cache misses doesn’t change! What is the largest value of STRETCH before cache misses changes? (Use IEC)

$32$ KiB, when STRETCH exactly equals $C$
Example 1 (Sp07 Final)

Only use L1$: \textbf{C} = 32 \text{ KiB}, \textbf{K} = 8 \text{ B}, \textbf{N} = 4, \text{ LRU, write-through}

char A[] is block aligned and SIZE = 32 \text{ MiB}

```c
char *A = (char *) malloc (SIZE*sizeof(char));
for (i = 0; i < (SIZE/STRETCH); i++) {
    for (j=0; j<STRETCH; j++)    sum  += A[i*STRETCH+j];
    for (j=STRETCH-1; j>=0; j++) prod += A[i*STRETCH+j];
}
```

c) If we double our STRETCH from (b), what is the ratio of cache \textit{hits} to \textit{misses}?

Now STRETCH = 64 \text{ KiB}. Moving sequentially by byte, so each block for entire 1\textsuperscript{st} inner loop has 1 miss and 7 hits (7:1). Upper half of STRETCH lives in cache, so first half of 2\textsuperscript{nd} inner loop is 8 hits/block (8:0). Second half is as before (7:1).
Example 1 (Sp07 Final)

Only use L1$: \textbf{C} = 32 \text{ KiB}, \textbf{K} = 8 \text{ B}, \textbf{N} = 4, \text{ LRU, write-through}

char A[] is block aligned and \textbf{SIZE} = 32 \text{ MiB}

\begin{verbatim}
    char *A = (char *) malloc (SIZE*sizeof(char));
    for (i = 0; i < (SIZE/STRETCH); i++) {
       for (j=0; j<STRETCH; j++) sum += A[i*STRETCH+j];
       for (j=STRETCH-1; j>=0; j++) prod += A[i*STRETCH+j];
    }
\end{verbatim}

c) If we double our \textbf{STRETCH} from (b), what is the ratio of cache \textit{hits} to \textit{misses}?

Considering the equal-sized chunks of half of each inner for loop, we have loop 1 1\textsuperscript{st} (7:1), loop 1 2\textsuperscript{nd} (7:1), loop 2 1\textsuperscript{st} (8:0), and loop 2 2\textsuperscript{nd} (7:1).

\[ 7+7+8+7:1+1+0+1 = 29:3 \]
Questions?
Example 2 (Sp13 Final)

32-bit MIPS, 4 GiB memory, single L1$ of size $C$ with block size $K$ ($C \geq K$ and a power of 2).

$A$, $B$ are arrays in different places of memory of equal size $n$ (power of 2 and a [natural #] multiple of $C$), block aligned.

```
// sizeof(uint8_t) = 1
SwapLeft(uint8_t *A, uint8_t *B, int n) {
    uint8_t tmp;
    for (int i = 0; i < n; i++) {
        tmp = A[i];
        A[i] = B[i];
        B[i] = tmp;
    }
}
```

Array data size is 1 byte

Do $n$ times:
- Read $A[i]$  
- Read $B[i]$, Write $A[i]$  
- Write $B[i]$  

Array data size is 1 byte
Example 2 (Sp13 Final)

32-bit MIPS, 4 GiB memory, single L1$ of size $C$ with block size $K$ ($C \geq K$ and a power of 2).

$A$, $B$ are arrays in different places of memory of equal size $n$ (power of 2 and a [natural #] multiple of $C$), block aligned.

a) If the cache is direct-mapped and the best hit:miss ratio is “H:1”, what is the block size in bytes?

Best case is $A[i]$ and $B[i]$ DON’T map to same slot.

Use every value of $i \in [0,n)$ only once.

Rd $A$, Rd $B$, Wr $A$, Wr $B$ $\rightarrow$ Miss, Miss, Hit, Hit ($1^{st}$ time) $\rightarrow$ Hit, Hit, Hit, Hit ($K$-1 times in block)

Per block:

$$4^*(K-1)+2:2 = 4K-2:2 = 2K-1:1 = H:1 \rightarrow K = (H+1)/2$$
Example 2 (Sp13 Final)

32-bit MIPS, 4 GiB memory, single L1$ of size $C$ with block size $K$ ($C \geq K$ and a power of 2).

$A, B$ are arrays in different places of memory of equal size $n$ (power of 2 and a [natural #] multiple of $C$), block aligned.

b) What is the worst hit:miss ratio?

Worst case is $A[i]$ and $B[i]$ map to same slot (conflict).
Rd $A$, Rd $B$, Wr $A$, Wr $B$ $\rightarrow$ Miss, Miss, Miss, Miss (all times)
because blocks keep replacing each other

0:1 (or 0:<anything>)
Example 2 (Sp13 Final)

c) Fill in code for SwapRight so that it does the same thing as SwapLeft but improves the (b) hit:miss ratio.

SwapRight(uint8_t *A, uint8_t *B, int n) {
    uint8_t tmpA, tmpB;
    for (int i = 0; i < n; i++) {
        tmpA = A[i]; ← Read A[i]
        tmpB = B[i]; ← Read B[i]
        B[i] = tmpA; ← Write B[i]
        A[i] = tmpB; ← Write A[i]
    }
}
Example 2 (Sp13 Final)

32-bit MIPS, 4 GiB memory, single L1$ of size $C$ with block size $K$ ($C \geq K$ and a power of 2).

$A, B$ are arrays in different places of memory of equal size $n$ (power of 2 and a [natural #] multiple of $C$), block aligned.

d) What is the worst hit:miss ratio for SwapRight?

Worst case is $A[i]$ and $B[i]$ map to same slot (conflict).

Rd $A$, Rd $B$, Wr $B$, Wr $A$ $\rightarrow$ Miss, Miss, Hit, Miss ($1^{st}$ time)

$\rightarrow$ Hit, Miss, Hit, Miss ($K$-1 times)

Per block:

$$(K-1)*2+1:(K-1)*2+3 = 2K-1:2K+1$$
Example 2 (Sp13 Final)

e) Change the cache to be **2-way set-associative**. Cache size $C$, block size $K$. What is the *worst* hit:miss ratio for SwapLeft with the following replacement policy?

- LRU and an empty cache

  Even if $A[i]$ and $B[i]$ map to same set, they can both co-exist.

  $\text{Rd } A, \text{Rd } B, \text{Wr } A, \text{Wr } B \rightarrow \text{Miss, Miss, Hit, Hit (1}^{\text{st}} \text{ time)}$

  $\rightarrow \text{Hit, Hit, Hit, Hit (}K\text{-1 times in block)}$

  So $2K\text{-1:1}$ (from part (a))
Summary

• Multilevel caches reduce *miss penalty*
  – Local vs. global miss rate
  – Optimize first level to be fast (low HT)
  – Optimize lower levels to not miss (minimize MP)

• Cache performance depends heavily on cache design (there are many choices)
  – Effects of parameters and policies
  – Cost vs. effectiveness

• Cache problems are hard!
BONUS SLIDES

You are NOT responsible for the material contained on the following slides, and we may not have enough time to get to them in lecture. They are good to look at if you have free time. They have been prepared in a way that should be easily readable.
Agenda

• Multilevel Caches
• Administrivia
• Improving Cache Performance
• Anatomy of a Cache Question
• Example Cache Questions
• **Bonus:** Contemporary Cache Specs
<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Intel Nehalem</th>
<th>AMD Opteron X4 (Barcelona)</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 cache organization</td>
<td>Split instruction and data caches</td>
<td>Split instruction and data caches</td>
</tr>
<tr>
<td>L1 cache size</td>
<td>32 KB each for instructions/data per core</td>
<td>64 KB each for instructions/data per core</td>
</tr>
<tr>
<td>L1 cache associativity</td>
<td>4-way (I), 8-way (D) set associative</td>
<td>2-way set associative</td>
</tr>
<tr>
<td>L1 replacement</td>
<td>Approximated LRU replacement</td>
<td>LRU replacement</td>
</tr>
<tr>
<td>L1 block size</td>
<td>64 bytes</td>
<td>64 bytes</td>
</tr>
<tr>
<td>L1 write policy</td>
<td>Write-back, Write-allocate</td>
<td>Write-back, Write-allocate</td>
</tr>
<tr>
<td>L1 hit time (load-use)</td>
<td>Not Available</td>
<td>3 clock cycles</td>
</tr>
<tr>
<td>L2 cache organization</td>
<td>Unified (instruction and data) per core</td>
<td>Unified (instruction and data) per core</td>
</tr>
<tr>
<td>L2 cache size</td>
<td>256 KB (0.25 MB)</td>
<td>512 KB (0.5 MB)</td>
</tr>
<tr>
<td>L2 cache associativity</td>
<td>8-way set associative</td>
<td>16-way set associative</td>
</tr>
<tr>
<td>L2 replacement</td>
<td>Approximated LRU replacement</td>
<td>Approximated LRU replacement</td>
</tr>
<tr>
<td>L2 block size</td>
<td>64 bytes</td>
<td>64 bytes</td>
</tr>
<tr>
<td>L2 write policy</td>
<td>Write-back, Write-allocate</td>
<td>Write-back, Write-allocate</td>
</tr>
<tr>
<td>L2 hit time</td>
<td>Not Available</td>
<td>9 clock cycles</td>
</tr>
<tr>
<td>L3 cache organization</td>
<td>Unified (instruction and data)</td>
<td>Unified (instruction and data)</td>
</tr>
<tr>
<td>L3 cache size</td>
<td>8192 KB (8 MB), shared</td>
<td>2048 KB (2 MB), shared</td>
</tr>
<tr>
<td>L3 cache associativity</td>
<td>16-way set associative</td>
<td>32-way set associative</td>
</tr>
<tr>
<td>L3 replacement</td>
<td>Not Available</td>
<td>Evict block shared by fewest cores</td>
</tr>
<tr>
<td>L3 block size</td>
<td>64 bytes</td>
<td>64 bytes</td>
</tr>
<tr>
<td>L3 write policy</td>
<td>Write-back, Write-allocate</td>
<td>Write-back, Write-allocate</td>
</tr>
<tr>
<td>L3 hit time</td>
<td>Not Available</td>
<td>38 (?)clock cycles</td>
</tr>
</tbody>
</table>
Core Area Breakdown

32KiB I$ per core
32KiB D$ per core
512KiB L2$ per core
Share one 8-MiB L3$