Cache performance and memory access patterns

Gotta cache 'em all
Today’s lecture

- Trade-offs between cache-size, block-size, associativity and performance (miss rate)
- Predicting Cache Performance from Code
  - Memory access patterns
  - Cache hit analysis for a given address stream
Load 0x1001, Load 0x1501, Load 0x160A, Load 0x1000

<table>
<thead>
<tr>
<th>Block offset</th>
<th>Set</th>
<th>Block offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>tag</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>00</td>
<td>01</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>11</td>
</tr>
</tbody>
</table>

A) Hit  
B) Miss, invalid  
C) Miss, tag mismatch
Most real architectures use multiple levels of caches

- Trade-off between access time & hit rate
  - L1 cache focuses on fast access time (with okay hit rate)
  - L2 cache focuses on good hit rate (with okay access time)
A miss on one level becomes an access at the next level.
Example cache statistics from Opteron

- **L1 Caches**: Instruction & Data
  - 64 kB
  - 64 byte blocks
  - 2-way set associative
  - 2 cycle access time

- **L2 Cache**:
  - 1 MB
  - 64 byte blocks
  - 4-way set associative
  - 16 cycle access time (total, not just miss penalty)

- **Memory**
  - 200+ cycle access time
A program has 2400 memory accesses. The program has a 1/8 miss rate in the L1 cache, resulting in 300 misses and 2100 hits in the L1 cache. How many times will the program access the L2 cache?

a) 1/8 times  
b) 300 times  
c) 2100 times  
d) 2400 times  
e) Cannot be determined
Caches have three dimensions. Use dimensional analysis to characterize the cache

\[
\text{NUM\_BLOCKS} = \text{NUM\_SETS (index)} \times \text{NUM\_BLOCKS/SET (set assoc)}
\]

\[
\text{CACHE\_SIZE} = \text{NUM\_BLOCKS} \times \text{BLOCK\_SIZE (offset)}
\]
Draw a picture of this cache

2-way set associative, 512 sets, 32B blocks
Cache Size Relationships

NUM_BLOCKS = NUM_SETS * NUM_BLOCKS/SET
CACHE_SIZE = NUM_BLOCKS * BLOCK_SIZE

- Example:
  - 2-way set associative, 512 sets, 32B blocks

- NUM_BLOCKS = a) 32  b) 64  c) 512  d) 1024

- CACHE_SIZE = a) 32B  b) 512B  c) 1KB  d) 16KB  e) 32KB
Draw a picture of this cache

64B direct-mapped with 16B blocks
Draw a picture of this cache

64B direct-mapped with 16B blocks
Code -> Address stream (Example)

```c
int A[SIZE], total = 0;
for (int i = 0 ; i < SIZE ; i ++) {
    total += A[i];
}
```

- How many loads/stores are there in this piece of code?
- What is the stream of addresses generated?

\[ \text{&A(0)}, \text{A+4, A+8, A+12, A+16} \]
Memory address stream determines where accesses map to cache

- Assume that $A = 0x00010038$

<table>
<thead>
<tr>
<th>Address</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>0x00010038 = 0001 0000 0000 0011 1000</td>
</tr>
<tr>
<td>$A+4$</td>
<td>0x0001003c = 0001 0000 0000 0011 1100</td>
</tr>
<tr>
<td>$A+8$</td>
<td>0x00010040 = 0001 0000 0000 0100 0000</td>
</tr>
<tr>
<td>$A+12$</td>
<td>0x00010044 = 0001 0000 0000 0100 0100</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>$A+40$</td>
<td>0x0001005c = 0001 0000 0000 0101 1100</td>
</tr>
<tr>
<td>$A+44$</td>
<td>0x00010060 = 0001 0000 0000 0110 0000</td>
</tr>
</tbody>
</table>

4 x 16B Blocks -> 2-bit set index, 4-bit block offset
We can estimate the miss rate based on the address stream

- Address stream: A, A+4, A+8, A+12, A+16, A+20, A+24, A+28, A+32, A+36, A+40, A+44, A+48, A+52, A+56, A+60, A+64, A+68, ...

- Direct-mapped cache, 4x16B blocks (total size: 32B. MUCH SMALLER THAN REAL CACHES)

- What is the miss rate if the stream is smaller than the cache (i.e., SIZE is smaller than 4 x 16)? 

\[ \frac{1}{4} \]
i>clicker question

- Address stream: A, A+4, A+8, A+12, A+16, A+20, A+24, A+28, A+32, A+36, A+40, A+44, A+48, A+52, A+56, A+60, A+64, A+68, ...
- Direct-mapped cache, 1024 x 32B blocks (total size: 32,768B)
- What is the miss rate if the stream is greater than the cache?
  a) >1/4  b) 1/4  c) <1/4
Estimate the miss rate for a two-way set-associative cache with the same number of blocks

- Address stream: A, A+4, A+8, A+12, A+16, A+20, A+24, A+28, A+32, A+36, A+40, A+44, A+48, A+52, A+56, A+60, A+64, A+68, ...
- Cache: Two-way set associative, 2 sets, 16B blocks (same size)

```
<table>
<thead>
<tr>
<th>Block offset</th>
<th>0x0</th>
<th>0x4</th>
<th>0x8</th>
<th>0xc</th>
</tr>
</thead>
<tbody>
<tr>
<td>LRU</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AC0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Set Index</th>
<th>0x0</th>
<th>0x1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Block offset</th>
<th>0x0</th>
<th>0x4</th>
<th>0x8</th>
<th>0xc</th>
<th>LRU</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>

a) >1/4  b) 1/4  c) <1/4
Stride length is the amount we increment over an array

```c
int A[SIZE], total = 0;
for (int i = 0 ; i < SIZE ; i += STRIDE) {
    // STRIDE != 1
    total += A[i];
}
```

- If STRIDE = 2, what is the stream of addresses generated?

\[ A, A+8, A+16, A+24 \]
What is the miss rate for a stride length of 2?

```
int A[SIZE], total = 0;
for (int i = 0 ; i < SIZE ; i += 2) {
    total += A[i];
}
```

Cache: 2-way SA, 2 sets, 16B blocks

a) 1/16  b) 1/8  c) 1/4  d) 1/2  e) Not enough info
Strided access generally increases the miss rate because it reduces spatial locality

```c
int A[SIZE], total = 0;
for (int i = 0 ; i < SIZE ; i += STRIDE) {
    // STRIDE != 1
    total += A[i];
}
```

In general, two scenarios:

**STRIDE < CACHE BLOCK SIZE**

```
+---------------------+
| 3                   |
| Block Offset        |
| 0x0     0x4   0x8   0xc |
```

```
+---------------------+
| 0x0     0x4   0x8   0xc |
```

**STRIDE >= CACHE BLOCK SIZE**

```
+---------------------+
| 5                   |
| Block Offset        |
|                    |
```

```
+---------------------+
| 0x0     0x4   0x8   0xc |
```

```
+---------------------+
| 0x0     0x4   0x8   0xc |
```

The miss rate of strided access can be expressed mathematically

- Miss Rate = MIN(1, DATA_SIZE*STRIDE / CACHE_BLOCK_SIZE)
Nested loops create temporal locality in data access

```c
int A[SIZE], total = 0;
for (int j = 0 ; j < N ; j ++) {
    for (int i = 0 ; i < SIZE ; i ++) {
        total += A[i];
    }
}
```

What does stream of addresses look like?

```
1 A, A+4, A+8, A+C ...
2 A, A+4, A+8, A+C ...
3 A, A+4, A+8, A+C ...
```

If sizeof(int) * SIZE < CACHE_SIZE and N is large, what is the miss rate?

a) 1/8  b) 1/4  c) 1  d) 0  e) not enough info

(pick best answer)
If the data structure is too big, we lose temporal locality. If sizeof(int)*SIZE >= 2*CACHE_SIZE and N is large, what is the miss rate?

```c
int A[SIZE], total = 0;
for (int j = 0 ; j < N ; j++) {
    for (int i = 0 ; i < SIZE ; i++) {
        total += A[i];
    }
} // &A[0] = 0x10392400
```

What does stream of addresses look like?

```
0x0 A(0), 1, 2, 3, 4...
0x4 A(0), 1, 2, 3, 4...
0x8 A(0), 1, 2, 3, 4...
0xc A(0), 1, 2, 3, 4...
```

If the data structure is too big, we lose temporal locality. If sizeof(int)*SIZE >= 2*CACHE_SIZE and N is large, what is the miss rate?
Performance depends on both the cache and data structure properties

```c
int A[SIZE], total = 0;
for (int j = 0 ; j < N ; j ++) {
    for (int i = 0 ; i < SIZE ; i ++) {
        total += A[i];
    }
    1st iter 5m, 20A = \frac{1}{4} miss rate
    2nd iter 2m, 20A = \frac{1}{16}
    3rd iter 2m, 20A = \frac{1}{64}
    CACHE_SIZE <= sizeof(int)*SIZE <= 2*CACHE_SIZE
}
```

Block Offset

<table>
<thead>
<tr>
<th>Set Index</th>
<th>0x0</th>
<th>0x1</th>
<th>0x2</th>
<th>0x3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0</td>
<td>n.m.</td>
<td>13</td>
<td>18</td>
<td>19</td>
</tr>
<tr>
<td>0x1</td>
<td>M.H.</td>
<td>M.H.</td>
<td>M.H.</td>
<td>M.H.</td>
</tr>
<tr>
<td>0x2</td>
<td>M.H.</td>
<td>M.H.</td>
<td>M.H.</td>
<td>M.H.</td>
</tr>
<tr>
<td>0x3</td>
<td>M.H.</td>
<td>M.H.</td>
<td>M.H.</td>
<td>M.H.</td>
</tr>
</tbody>
</table>

Array < Cache

Array >= 2*Cache
Non-linear access of arrays sacrifices temporal and spatial locality

```c
int A[SIZE], total = 0;
for (int j = 0; j < 4; j++) {
    for (int i = 0; i < SIZE; i += 4) {
        total += A[i+j];
    }
}
```

What does stream of addresses look like?

- A(0), A(4), A(8), A(12), ...
- A(1), A(5), A(9), A(13), ...
- A(2), A(6), A(10), A(14), ...
- A(3), A(7), A(11), A(15), ...

What is the miss rate?

- a) 1/8
- b) 1/4
- c) 1/2
- d) 0
- e) 1

SIZE = 1024
General Reuse Questions

- Is the data still in the cache when reuse occurs?
  Temporal Locality

- Special case: Does all of the data fit in the cache?
i>clicker, how do linked lists perform on a cache?

struct list_t { int data, struct list_t *next };
for (struct list_t *l = list_head; l != NULL; l = l->next) {
    total += l->data;
} //list_head = 0x101c

Assume a 2-way SA cache with 8 sets, 32B blocks. If the list is 1000 elements long, how many misses per iteration of the loop?

a) 1  b) ¼  c) 1/8  d) 0  e) not enough info
int A[SIZE], B[SIZE];
for (int i = 0 ; i < SIZE ; i ++) {
    B[i] = A[i];
}

What is the stream of addresses generated?

Assume a direct-mapped cache with 8, 32B blocks.
Assume sizeof(int)*SIZE >> CACHE_SIZE.
How many cache misses per iteration of the loop?
   a) 2   b) 1   c) 1/4   d) 1/8   e) not enough info
What addresses map to the same set?

- Multiples of the set size.

**SET_SIZE = NUM_SETS * BLOCK_SIZE**
  - Also SET_SIZE = CACHE_SIZE / NUM_BLOCKS/SET

Examples:
- Direct-mapped, 1024 32B blocks
- 2-way SA, 512 sets, 32B blocks
Cache Analysis Questions

- How big is each data item? (how many fit in a cache block?)
- Is there data reuse or use of data in the same cache block?
- Is the data/block still in cache when reuse occurs?
  - How big is the data relative to the cache?
  - Is there aliasing (cache conflicts) that is problematic?

Some useful relationships:
- Strides: miss rate = \( \text{MIN}(1, \text{sizeof(data)} \times \text{stride}/\text{BLOCK\_SIZE}) \)
- Adjacent blocks in memory -> adjacent sets
- Addresses A, A+\(\text{BLOCK\_SIZE} \times \text{NUM\_SETS}\) are in the same set
Tradeoffs between cache parameters are evaluated experimentally

- General goal is to minimize miss rate
- Will look at examples on the following slides
- We will do some cache simulations on the MP’s.

Remember: miss rate is the number of misses per memory access
Predict associativity vs. miss rate

Increasing associativity will
a) Decrease the miss rate
b) Not affect the miss rate
c) Increase the miss rate
d) Have mixed results
Increasing associativity generally improves miss rates at the cost of hardware complexity

- Each set has more blocks, so there’s less chance of a conflict between two addresses which both belong in the same set.
Predict cache size vs. miss rate

Increasing cache size will
a) Decrease the miss rate
b) Not affect the miss rate
c) Increase the miss rate
d) Have mixed results
Increasing cache size also generally improves miss rates

- The larger a cache is, the lower chance of conflict.
Predict block size vs. miss rate

Increasing block size will

a) Decrease the miss rate
b) Not affect the miss rate
c) Increase the miss rate
d) Have mixed results
Small blocks do not take advantage of spatial locality

- But if cache blocks get too big relative to the cache size, they increase conflicts