Cache Memory: Instruction Cache, HW/SW Interaction

CPS 104
Lecture 18

Admin

- Project Due Nov 22
- Homework #5 Due Nov 14

What’s Ahead
- Finish Caches
- Virtual Memory
- Input/Output (1 homework)
- Advanced Topics
Review: Cache Memory

- **Cost effective memory system**
  - Big cheap slow + small fast expensive
- **For a 1024 \(2^{10}\) byte cache with 32-byte blocks:**
  - The uppermost 22 \((32 - 10)\) address bits are the **Cache Tag**
  - The lowest 5 address bits are the **Block Offset** (Byte Select) (Block Size = \(2^5\))
  - The next 5 address bits (bit 5 - bit 9) are the **Cache Index**

<table>
<thead>
<tr>
<th>31</th>
<th>9</th>
<th>4</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cache Tag</td>
<td>Cache Index</td>
<td>Block Offset</td>
</tr>
</tbody>
</table>

**1KB Direct Mapped Cache with 32B Blocks**

- Stored as part of the cache “state”
- **Valid Bit**
- **Cache Tag**: Example: 0x50
- Ex: 0x01
- Ex: 0x00
- **Cache Data**
  - Byte 31 ** Byte 1 Byte 0
  - Byte 63 ** Byte 33 Byte 32
  - Byte 1023 ** Byte 992
  - Byte Select

© Alvin R. Lebeck
Instruction Cache

- Separate Inst & Data Caches
  - Harvard Architecture
- Can access both at same time
- Combined L2
  - L2 >> L1

Four Questions for Memory Hierarchy Designers

- Q1: Where can a block be placed in the upper level? *(Block placement)*
  - Fully Associative, Set Associative, Direct Mapped
- Q2: How is a block found if it is in the upper level? *(Block identification)*
  - Tag/Block
- Q3: Which block should be replaced on a miss? *(Block replacement)*
  - Random, LRU
- Q4: What happens on a write? *(Write strategy)*
  - Write Back or Write Through (with Write Buffer)
Write Through versus Write Back

- Cache read is much easier to handle than cache write:
  - Instruction cache is much easier to design than data cache
- Cache write:
  - How do we keep data in the cache and memory consistent?
- Two options (decision time again :-)
  - **Write Back**: write to cache only. Write the cache block to memory when that cache block is being replaced on a cache miss.
    - Need a “dirty bit” for each cache block
    - Greatly reduce the memory bandwidth requirement
    - Control can be complex
  - **Write Through**: write to cache and memory at the same time.
    - Add write buffer to absorb bursts of writes

Cache Performance

CPU time = (CPU execution clock cycles + Memory stall clock cycles) x clock cycle time

Memory stall clock cycles = Memory accesses x Miss rate x Miss penalty

Example
- Assume every instruction takes 1 cycle
- Miss penalty = 20 cycles
- Miss rate = 10%
- 1000 total instructions, 300 memory accesses
- Memory stall cycles? CPU clocks?
Cache Performance

- Memory Stall cycles = 300 * 0.10 * 20 = 600
- CPUclocks = 1000 + 600 = 1600

- 60% slower because of cache misses!

- Change miss penalty to 100 cycles
- CPUclocks = 1000 + 3000 = 4000 cycles

Improving Cache Performance

1. Reduce the miss rate,
2. Reduce the miss penalty, or
3. Reduce the time to hit in the cache.
Reducing Misses (The 3 Cs)

- **Compulsory**—The first access to a block is not in the cache, so the block must be brought into the cache. These are also called *cold start misses* or *first reference misses*. *(Misses in Infinite Cache)*

- **Capacity**—If the cache cannot contain all the blocks needed during execution of a program, capacity misses will occur due to blocks being discarded and later retrieved. *(Misses in Size X Cache)*

- **Conflict**—If the block-placement strategy is set associative or direct mapped, conflict misses (in addition to compulsory and capacity misses) will occur because a block can be discarded and later retrieved if too many blocks map to its set. These are also called *collision misses* or *interference misses*. *(Misses in N-way Associative, Size X Cache)*

Cache Performance

- Your program and caches
- Can you affect performance?
- Think about 3Cs
Mapping Arrays to Memory

Row-major

0 1 2 3 4
5 6 7 8 9
10 11 12 13 14
15 16 17 18 19
20 21 22 23 24

Column major

0 5 10 15 20
1 6 11 16 21
2 7 12 17 22
3 8 13 18 23
4 9 14 19 24

Part of the Row maps into cache

Array Mapping and Cache Behavior

Elements spread out in memory because of column-major mapping
• Fixed mapping into cache
• Self-interference in cache

© Alvin R. Lebeck
Data Cache Performance

- **Instruction Sequencing**
  - *Loop Interchange*: change nesting of loops to access data in order stored in memory
  - *Loop Fusion*: Combine 2 independent loops that have same looping and some variables overlap
  - *Blocking*: Improve temporal locality by accessing “blocks” of data repeatedly vs. going down entire columns or rows

- **Data Layout**
  - *Merging Arrays*: Improve spatial locality by single array of compound elements vs. 2 separate arrays
  - *Nonlinear Array Layout*: Mapping 2 dimensional arrays to the linear address space
  - *Pointer-based Data Structures*: node-allocation

---

Loop Interchange Example

/* Before */
for (k = 0; k < 100; k = k+1)
  for (j = 0; j < 100; j = j+1)
    for (i = 0; i < 5000; i = i+1)
      x[i][j] = 2 * x[i][j];

/* After */
for (k = 0; k < 100; k = k+1)
  for (i = 0; i < 5000; i = i+1)
    for (j = 0; j < 100; j = j+1)
      x[i][j] = 2 * x[i][j];

Sequential accesses instead of striding through memory every 100 words
Loop Fusion Example

/* Before */
for (i = 0; i < N; i = i+1)
    for (j = 0; j < N; j = j+1)
        a[i][j] = 1/b[i][j] * c[i][j];
for (i = 0; i < N; i = i+1)
    for (j = 0; j < N; j = j+1)
        d[i][j] = a[i][j] + c[i][j];
/* After */
for (i = 0; i < N; i = i+1)
    for (j = 0; j < N; j = j+1)
        { a[i][j] = 1/b[i][j] * c[i][j];
          d[i][j] = a[i][j] + c[i][j];}

2 misses per access to a & c vs. one miss per access

Blocking Example

/* Before */
for (i = 0; i < N; i = i+1)
    for (j = 0; j < N; j = j+1)
        { r = 0;
          for (k = 0; k < N; k = k+1)
            r = r + y[i][k]*z[k][j];
          x[i][j] = r;}

• Two Inner Loops:
  ✍ Read all NxN elements of z[]
  ✍ Read N elements of 1 row of y[] repeatedly
  ✍ Write N elements of 1 row of x[]

• Capacity Misses a function of N & Cache Size:
  ✍ 3 NxN ⇒ no capacity misses; otherwise ...

• Idea: compute on BxB submatrix that fits
### Blocking Example

/* After */
for (jj = 0; jj < N; jj = jj+B)
for (kk = 0; kk < N; kk = kk+B)
for (i = 0; i < N; i = i+1)
  for (j = jj; j < min(jj+B-1,N); j = j+1)
    {r = 0;
     for (k = kk; k < min(kk+B-1,N); k = k+1)
      {r = r + y[i][k]*z[k][j];};
    x[i][j] = x[i][j] + r;};

• Capacity Misses from $2N^3 + N^2$ to $2N^3/B + N^2$
• B called *Blocking Factor*
• Conflict Misses Too?

### Reducing Conflict Misses by Blocking

- **Conflict misses in caches not FA vs. Blocking size**
  - Lam et al [1991] a blocking factor of 24 had a fifth the misses vs. 48 despite both fit in cache
Data Layout Optimizations

- So far program control
- Changes the order in which memory is accessed
- We can also change the way our data structures map to memory
- 2-dimensional array
- Pointer-based data structures

Merging Arrays Example

/* Before */
int val[SIZE];
int key[SIZE];

/* After */
struct merge {
    int val;
    int key;
};
struct merge merged_array[SIZE];

Reducing conflicts between val & key
Layout and Cache Behavior

- Tile elements spread out in memory because of column-major mapping
- Fixed mapping into cache
- Self-interference in cache

Making Tiles Contiguous

- Elements of a quadrant are contiguous
- Recursive layout
- Elements of a tile are contiguous
- No self-interference in cache
Pointer-based Data Structures

- Linked List, Binary Tree
- Basic idea is to group linked elements close together in memory
- Need relatively static traversal pattern
- Or could do it during garbage collection/compaction

Summary of Compiler Optimizations to Reduce Cache Misses

![Graph showing performance improvement for different benchmarks](image_url)

© Alvin R. Lebeck
Reducing I-Cache Misses by Compiler Optimizations

- **Instructions**
  - Reorder procedures in memory to reduce misses
  - Profiling to look at conflicts
  - McFarling [1989] reduced caches misses by 75% on 8KB direct mapped cache with 4 byte blocks

Summary

- Cost Effective Memory Hierarchy
- Split Instruction and Data Cache
- 4 Questions
- CPU cycles/time, Memory Stall Cycles
- Your programs and cache performance

Next Time

- Exceptions and Interrupts
- Reading Chapter 5.6