Cache Memory: Instruction Cache, HW/SW Interaction

CPS 104
Lecture 18

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 (bit5 - bit9) are the Cache Index

1KB Direct Mapped Cache with 32B Blocks

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)
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?

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)
Part of the Row maps into cache

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
- 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 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

Array Mapping and Cache Behavior

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

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] = Z * 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] = Z * x[i][j];
```
Sequential accesses instead of striding through memory every 100 words

Blocking Example

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

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

- Capacity Misses a function of N & Cache Size:
  - 3 N x N => no capacity misses; otherwise ...
- Idea: compute on B x B submatrix that fits
### Blocking Example

```c
/* 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];
    };
```

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

### 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

```c
/* 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

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