CS 152 Computer Architecture and Engineering
CS252 Graduate Computer Architecture

Lecture 7 – Memory III

Krste Asanovic
Electrical Engineering and Computer Sciences
University of California at Berkeley

http://www.eecs.berkeley.edu/~krste
http://inst.eecs.berkeley.edu/~cs152
Last time in Lecture 6

- 3 C’s of cache misses
  - Compulsory, Capacity, Conflict

- Write policies
  - Write back, write-through, write-allocate, no write allocate

- Pipelining write hits

- Multi-level cache hierarchies reduce miss penalty
  - 3 levels common in modern systems (some have 4!)
  - Can change design tradeoffs of L1 cache if known to have L2
  - Inclusive versus exclusive cache hierarchies
Recap: Multilevel Caches

**Problem:** A memory cannot be large and fast

**Solution:** Increasing sizes of cache at each level

Local miss rate = misses in cache / accesses to cache

Global miss rate = misses in cache / CPU memory accesses

Misses per instruction = misses in cache / number of instructions
Itanium-2 On-Chip Caches  
(Intel/HP, 2002)

**Level 1:** 16KB, 4-way s.a., 64B line, quad-port (2 load+2 store), single cycle latency

**Level 2:** 256KB, 4-way s.a, 128B line, quad-port (4 load or 4 store), five cycle latency

**Level 3:** 3MB, 12-way s.a., 128B line, single 32B port, twelve cycle latency
Power 7 On-Chip Caches [IBM 2009]

- 32KB L1 I$/core
- 32KB L1 D$/core
  - 3-cycle latency
- 256KB Unified L2$/core
  - 8-cycle latency
- 32MB Unified Shared L3$
- Embedded DRAM (eDRAM)
  - 25-cycle latency to local slice
IBM z196 Mainframe Caches 2010

- 96 cores (4 cores/chip, 24 chips/system)
  - Out-of-order, 3-way superscalar @ 5.2GHz
- L1: 64KB I-$/core + 128KB D-$/core
- L2: 1.5MB private/core (144MB total)
- L3: 24MB shared/chip (eDRAM) (576MB total)
- L4: 768MB shared/system (eDRAM)
Exponential X704 PowerPC Processor (1997)

- 2KB L1 Direct-Mapped Instruction Cache
- 2KB L1 Direct-Mapped Write-Through Data Cache
- 32KB L2 8-way Set-Associative Write-Back Unified Cache
Victim Caches (HP 7200)

 Victim cache is a small associative backup cache, added to a direct-mapped cache, which holds recently evicted lines
  • First look up in direct-mapped cache
  • If miss, look in victim cache
  • If hit in victim cache, swap hit line with line now evicted from L1
  • If miss in victim cache, L1 victim -> VC, VC victim- ->?

Fast hit time of direct mapped but with reduced conflict misses
MIPS R10000 Off-Chip L2 Cache (Yeager, IEEE Micro 1996)

Figure 1. System configuration. The cluster bus directly connects as many as four chips.
Way-Predicting Caches (MIPS R10000 L2 cache)

- Use processor address to index into way-prediction table
- Look in predicted way at given index, then:
  - **HIT**: Return copy of data from cache
  - **MISS**: Look in other way

- **MISS**: Read block of data from next level of cache
- **SLOW HIT (change entry in prediction table)**
Figure 12. Refill from the set-associative secondary cache. In this example, the secondary clock equals the processor’s internal pipeline clock. It may be slower.
Way-Predicting Instruction Cache (Alpha 21264-like)

Store last-used way for sequential path and predicted branch taken path. Can be fetching multiple instructions per cycle.

Jump control -> Jump target

0x4 Add

PC addr

Primary Instruction Cache

way

addr

inst

Sequential Way

Branch Target Way

Can be fetching multiple instructions per cycle.
Prefetching

- Speculate on future instruction and data accesses and fetch them into cache(s)
  - Instruction accesses easier to predict than data accesses

- Varieties of prefetching
  - Hardware prefetching
  - Software prefetching
  - Mixed schemes

- What types of misses does prefetching affect?
Issues in Prefetching

- Usefulness – should produce hits
- Timeliness – not late and not too early
- Cache and bandwidth pollution

Diagram:
- CPU
- L1 Instruction
- L1 Data
- Unified L2 Cache
- Prefetched data
**Hardware Instruction Prefetching**

Instruction prefetch in Alpha AXP 21064

- Fetch two lines on a miss; the requested line (i) and the next consecutive line (i+1)
- Requested line placed in cache, and next line in instruction stream buffer
- If miss in cache but hit in stream buffer, move stream buffer line into cache and prefetch next line (i+2)
Hardware Data Prefetching

- **Prefetch-on-miss:**
  - Prefetch $b + 1$ upon miss on $b$

- **One-Block Lookahead (OBL) scheme**
  - Initiate prefetch for block $b + 1$ when block $b$ is accessed
  - Why is this different from doubling block size?
  - Can extend to N-block lookahead

- **Strided prefetch**
  - If observe sequence of accesses to line $b$, $b+N$, $b+2N$, then prefetch $b+3N$ etc.

- **Example:** IBM Power 5 [2003] supports eight independent streams of strided prefetch per processor, prefetching 12 lines ahead of current access
CS152 Administrivia

- Lab 1 due at start of class on Monday Feb 12
- PS 2 out on Wednesday, Thursday, due Wednesday Feb 21
- Friday’s sections will review PS 1 and solutions
CS252 Administrivia

- Start thinking of class projects and forming teams of two
- Proposal due Monday March 5th
Software Prefetching

for (i=0; i < N; i++) {
    prefetch( &a[i + 1] );
    prefetch( &b[i + 1] );
    SUM = SUM + a[i] * b[i];
}
Software Prefetching Issues

- Timing is the biggest issue, not predictability
  - If you prefetch very close to when the data is required, you might be too late
  - Prefetch too early, cause pollution
  - Estimate how long it will take for the data to come into L1, so we can set P appropriately
  - *Why is this hard to do?*

```c
for(i=0; i < N; i++) {
    prefetch( &a[i + P] );
    prefetch( &b[i + P] );
    SUM = SUM + a[i] * b[i];
}
```

*Must consider cost of prefetch instructions*
Software Prefetching Example

["Data prefetching on the HP PA8000", Santhanam et al., 1997]

subroutine example(X,Y,Z,B,n)
  real *4 X(1), Y(1), Z(1), B
  do i = 1, n
    X(i) = Y(i) + B*Z(i)
  enddo

Table 1: Speedup with/without Prefetch Minimization

<table>
<thead>
<tr>
<th></th>
<th>With cache capacity misses</th>
<th>With cache hits</th>
</tr>
</thead>
<tbody>
<tr>
<td>With prefetch instruction minimization</td>
<td>70%</td>
<td>-13%</td>
</tr>
<tr>
<td>Without prefetch instruction minimization</td>
<td>70%</td>
<td>-55%</td>
</tr>
</tbody>
</table>

Figure 2: Speedup Ratio for Different Prefetch Distances
Compiler Optimizations

- Restructuring code affects the data access sequence
  - Group data accesses together to improve spatial locality
  - Re-order data accesses to improve temporal locality

- Prevent data from entering the cache
  - Useful for variables that will only be accessed once before being replaced
  - Needs mechanism for software to tell hardware not to cache data (“no-allocate” instruction hints or page table bits)

- Kill data that will never be used again
  - Streaming data exploits spatial locality but not temporal locality
  - Replace into dead cache locations
Loop Interchange

```
for(j=0; j < N; j++) {
    for(i=0; i < M; i++) {
        x[i][j] = 2 * x[i][j];
    }
}
```

```
for(i=0; i < M; i++) {
    for(j=0; j < N; j++) {
        x[i][j] = 2 * x[i][j];
    }
}
```

What type of locality does this improve?
Loop Fusion

for(i=0; i < N; i++)
    a[i] = b[i] * c[i];

for(i=0; i < N; i++)
    d[i] = a[i] * c[i];

for(i=0; i < N; i++)
{
    a[i] = b[i] * c[i];
    d[i] = a[i] * c[i];
}

What type of locality does this improve?
Matrix Multiply, Naïve Code

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

Not touched  Old access  New access
Matrix Multiply with Cache Tiling

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

What type of locality does this improve?
Acknowledgements

- This course is partly inspired by previous MIT 6.823 and Berkeley CS252 computer architecture courses created by my collaborators and colleagues:
  - Arvind (MIT)
  - Joel Emer (Intel/MIT)
  - James Hoe (CMU)
  - John Kubiatowicz (UCB)
  - David Patterson (UCB)