Last time in Lecture 7

• 3 C’s of cache misses: compulsory, capacity, conflict
• Average memory access time = hit time + miss rate * miss penalty
• To improve performance, reduce:
  – hit time
  – miss rate
  – and/or miss penalty
• Primary cache parameters:
  – Total cache capacity
  – Cache line size
  – Associativity
**Multilevel Caches**

- A memory cannot be large and fast
- 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

**A Typical Memory Hierarchy c.2008**

Split instruction & data primary caches (on-chip SRAM)
Multiple interleaved memory banks (off-chip DRAM)

Multiported register file (part of CPU)
Large unified secondary cache (on-chip SRAM)
Presence of L2 influences L1 design

- Use smaller L1 if there is also L2
  - Trade increased L1 miss rate for reduced L1 hit time and reduced L1 miss penalty
  - Reduces average access energy
- Use simpler write-through L1 cache with on-chip L2
  - Write-back L2 cache absorbs write traffic, doesn’t go off-chip
  - At most one L1 miss request per L1 access (no dirty victim write back) simplifies pipeline control
  - Simplifies coherence issues
  - Simplifies error recovery in L1 (can use just parity bits in L1 and reload from L2 when parity error detected on L1 read)

Inclusion Policy

- Inclusive multilevel cache:
  - Inner cache holds copies of data in outer cache
  - External access need only check outer cache
  - Most common case
- Exclusive multilevel caches:
  - Inner cache may hold data not in outer cache
  - Swap lines between inner/outer caches on miss
  - Used in AMD Athlon with 64KB primary and 256KB secondary cache

Why choose one type or the other?
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

Reducing penalty of associativity

- Associativity reduces conflict misses, but requires expensive (area, energy, delay) multi-way tag search
- Two optimizations to reduce cost of associativity
  - Victim caches
  - Way prediction
**Victim Caches (Jouppi 1990)**

Victim cache is a small associative back up 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.

**Way Predicting Caches (MIPS R10000 off-chip 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)
Way Predicting Instruction Cache
(Alpha 21264-like)

Reduce Miss Penalty of Long Blocks:
Early Restart and Critical Word First

- Don’t wait for full block before restarting CPU
- Early restart—As soon as the requested word of the block arrives, send it to the CPU and let the CPU continue execution
  - Spatial locality ⇒ tend to want next sequential word, so not clear size of benefit of just early restart
- Critical Word First—Request the missed word first from memory and send it to the CPU as soon as it arrives; let the CPU continue execution while filling the rest of the words in the block
  - Long blocks more popular today ⇒ Critical Word 1st Widely used

block
Increasing Cache Bandwidth with Non-Blocking Caches

- **Non-blocking cache** or **lockup-free cache** allow data cache to continue to supply cache hits during a miss
  - requires Full/Empty bits on registers or out-of-order execution
- “hit under miss” reduces the effective miss penalty by working during miss vs. ignoring CPU requests
- “hit under multiple miss” or “miss under miss” may further lower the effective miss penalty by overlapping multiple misses
  - Significantly increases the complexity of the cache controller as there can be multiple outstanding memory accesses, and can get miss to line with outstanding miss (secondary miss)
  - Requires pipelined or banked memory system (otherwise cannot support multiple misses)
  - Pentium Pro allows 4 outstanding memory misses
  - (Cray X1E vector supercomputer allows 2,048 outstanding memory misses)

Value of Hit Under Miss for SPEC (old data)

**Hit Under Misses**

**Integer**
- FP programs on average: AMAT= 0.68 → 0.52 → 0.34 → 0.26
- Int programs on average: AMAT= 0.24 → 0.20 → 0.19 → 0.19
- 8 KB Data Cache, Direct Mapped, 32B block, 16 cycle miss, SPEC 92
CS152 Administrivia

- Textbooks should appear in Cal book store soon (already there? Or next couple of days)
- Quiz 1 handed back next Tuesday in class.
- First set of open-ended labs seemed fine. Opinions?
  - Encourage you to try to find your own ideas, instead of following suggestions.

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

Hardware Instruction Prefetching

Instruction prefetch in Alpha AXP 21064
- Fetch two blocks on a miss; the requested block (i) and the next consecutive block (i+1)
- Requested block placed in cache, and next block in instruction stream buffer
- If miss in cache but hit in stream buffer, move stream buffer block into cache and prefetch next block (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 block \( b, b+N, b+2N, \) 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

Software Prefetching

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

What property do we require of the cache for prefetching to work?
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

Compiler Optimizations

• Restructuring code affects the data block 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 (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?*
Blocking

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

What type of locality does this improve?
Workstation Memory System
(Apple PowerMac G5, 2003)

Dual 2GHz processors, each has:
• 64KB I-cache, direct mapped
• 32KB D-cache, 2-way
• 512KB L2 unified cache, 8-way
• All 128B lines

AGP Graphics Card, 533MHz, 32-bit bus, 2.1GB/s

Up to 8GB DDR SDRAM, 400MHz, 128-bit bus, 6.4GB/s

1GHz, 2x32-bit bus, 16GB/s

PCI-X Expansion, 133MHz, 64-bit bus, 1 GB/s

Acknowledgements

• These slides contain material developed and copyright by:
  – Arvind (MIT)
  – Krste Asanovic (MIT/UCB)
  – Joel Emer (Intel/MIT)
  – James Hoe (CMU)
  – John Kubiatowicz (UCB)
  – David Patterson (UCB)

• MIT material derived from course 6.823
• UCB material derived from course CS252