Outline

• Announcements
  – Assignment #4 will be available tomorrow morning
  – Final exam on December 11th: Room 402, WWH

• This lecture
  – Improving cache performance (cont’d)
  – Main memory organizations

[Hennessy/Patterson CA:QA (3rd Edition): Chapter 5]
(Review)
C. Using Parallelism to Reduce Miss Penalty/Rate

*Idea*: Permit multiple “outstanding” memory operations
  – Can overlap memory access latencies
  – Can benefit from activity done on behalf of other operations

Three commonly-employed schemes
• Non-blocking caches
• Hardware prefetching
• Software prefetching

(Review)
C.1. Non-blocking Caches to Reduce Stalls on Misses

• Decoupled instruction and data caches allow CPU to continue fetching instructions while waiting on a data cache miss
  – L1 cache misses can be tolerated by superscalar out-of-order machines

• Non-blocking or lockup-free caches allow data cache to continue to supply cache hits during a miss
  – requires out-of-order execution CPU
• “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
  – Typically also requires multiple memory banks
  – Pentium Pro allows 4 outstanding memory misses
Value of Hit-Under-Miss for SPEC92
8KB direct-mapped cache, 32B blocks, 16-cycle penalty

Floating-point

C.2. Reducing Misses by Hardware Prefetching of Instructions & Data

- Instruction Prefetching
  - Alpha 21064 fetches 2 blocks (requested and subsequent) on a miss
  - Extra block in “stream buffer”
  - On miss, check stream buffer
- Works with data blocks too
  - Hardware identifies stream of accesses and then prefetches them
  - Can compute stride by comparing current and previous access
  - UltraSPARC III supports up to 8 simultaneous prefetches
- Prefetching relies on having extra memory bandwidth that can be used without penalty

How well does this work?
- Jouppi [1990]
  - (for instructions w.r.t. to a 4KB direct-mapped cache)
    1-block buffer catches 15-25% of misses, 4-block: 50%, 16-block: 72%
  - (for data w.r.t. to a 4KB direct-mapped cache)
    1-block buffer: 25%, 4 streams: 43%
- Palacharla & Kessler [1994]
  - for scientific programs, 8 streams got 50% to 70% of misses from 2 64KB, 4-way set associative caches
C.3. Reducing Misses by Software Prefetching of Data

- Two variants
  - Load data into register (HP PA-RISC loads)
  - Load data into cache (MIPS IV, PowerPC, SPARC v. 9)

Issues
- Special prefetching instructions typically cannot cause faults (a form of speculative execution)
- Processor must be able to proceed while prefetched data is being fetched
  - i.e., non-blocking data caches
- Issuing the prefetch instructions takes time
  - Is cost of prefetch issues < savings in reduced misses?
  - Higher superscalar reduces difficulty of issue bandwidth

Example of Compiler-Controlled Prefetching

8KB direct-mapped D-cache, 16B blocks, write-back with write-allocate
- a: 3 rows, 100 columns, 8B/element
- b: 101 rows, 3 columns, 8B/element

```c
for (i=0; i<3; i++)
  for (j=0; j<100; j++)
    a[i][j] = b[j][0] * b[j+1][0]

for (j=0; j<100; j++) {
  prefetch( b[j+7][0] );
  prefetch( a[0][j+7] );
  a[0][j] = b[j][0] + b[j+1][0];
}

for (i=1; i<3; i++)
  for (j=0; j<100; j++) {
    prefetch( a[i][j+7] );
    a[i][j] = b[j][0] * b[j+1][0]
  }
```

Misses for “a”: 3x(100/2) = 150
Misses for “b”: 1 + 100 = 101
(b[j+1][0] reused for next j)

Misses for “a”: 3x(7/2) = 12
(a[i][0] ... a[i][6])
Misses for “b”: 7
(b[0][0] ... b[6][0])
i.e., saves 232 cache misses

Number of prefetches: 200 + 2x100 = 400
only consumes issue slot
D. Reducing Cache Hit Time

- Obvious approach: Smaller and simpler (low associativity) caches
  - Notable that L1 cache sizes have not increased
    - Alpha 21264/21364; UltraSPARC II/III; AMD K6/Athlon

Other techniques

- Avoiding address translation during cache lookup
  - Alternative 1: Index caches using “virtual addresses”
    - Needs to cope with several problems
      - Protection
      - Reuse of virtual addresses across processes
      - Aliasing/synonyms: Two processes refer to the same physical address
      - I/O (typically uses physical addresses)
  - Alternative 2: Use part of the page offset to index the cache
    does not change between virtual and physical addresses

---

D.1. Virtually Indexed, Physically Tagged Caches

- Overlap indexing of cache with translation of virtual addresses
  - Tag comparison done with physical addresses

Implications

- Direct-mapped caches can be no bigger than page size
  - Index determines block
- Set-associative caches
  - Page offset can be viewed as ( Index + block offset ) above
  - Cache size = 2^{page offset} x Set associativity
  - So, increased associativity allows larger cache sizes
    - Pentium III (8KB pages): 2-way set-associative 16 KB cache
    - IBM 3033 (4KB pages): 16-way set-associative 64 KB cache
D.2. Trace Caches

- A challenge in multiple-issue processors is to supply enough instructions every cycle without dependencies
  - Challenge: fetching across branches

- Option 1: Combine branch prediction with instruction prefetching
  - Instructions stored according to memory addresses

- Option 2: A separate cache that stores and provides a dynamic sequence of instructions including taken branches (Trace Cache)
  - **Pros**
    - Effective use of cache block: no wasted words, no conflicts, …
  - **Cons**
    - Address mapping mechanisms
    - Same instruction may be stored multiple times
  - Used in the Intel NetBurst microarchitecture (Pentium 4)

## Cache Optimization Summary

<table>
<thead>
<tr>
<th>Technique</th>
<th>MP</th>
<th>MR</th>
<th>HT</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multilevel caches</td>
<td>+</td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Early Restart &amp; Critical Word 1st</td>
<td>+</td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Priority to Read Misses</td>
<td>+</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Merging write buffer</td>
<td>+</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Victim Caches</td>
<td>+</td>
<td>+</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Larger Block Size</td>
<td>−</td>
<td>+</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Higher Associativity</td>
<td>+</td>
<td>−</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Pseudo-Associative Caches</td>
<td>+</td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Compiler Reduce Misses</td>
<td>+</td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Non-Blocking Caches</td>
<td>+</td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>HW Prefetching of Instr/Data</td>
<td>+</td>
<td>+</td>
<td></td>
<td>2/3</td>
</tr>
<tr>
<td>Compiler Controlled Prefetching</td>
<td>+</td>
<td>+</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>Avoiding Address Translation</td>
<td>+</td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Trace Cache</td>
<td>+</td>
<td></td>
<td></td>
<td>3</td>
</tr>
</tbody>
</table>
Main Memory Background

- Performance of Main Memory:
  - **Latency**: Cache miss penalty
    - Access Time: time between request and word arrives
    - Cycle Time: time between requests
  - **Bandwidth**: I/O and large block miss penalty (L2)

- Main Memory is DRAM: **Dynamic Random Access Memory**
  - Dynamic since needs to be refreshed periodically (~8 ms, <5% time)
  - Addresses divided into 2 halves (Memory as a 2D matrix):
    - RAS or Row Access Strobe, CAS or Column Access Strobe

- Cache uses SRAM: **Static Random Access Memory**
  - No refresh (6 transistors/bit vs. 1 transistor/bit, area is 10X)
  - Address not divided: Full address

- DRAM/SRAM size ratio of 4 – 8,
  SRAM/DRAM cost/cycle time ratio 8 – 16
Internal Organization of a 64M bit DRAM

- Internally, might use banks of memory arrays
  - E.g., 256 1024x1024 arrays, or 16 2048x2048 arrays
- Normally packaged as dual inline memory modules (DIMMs)
  - Typically 4-16 DRAM chips, 8 byte wide

4 Key DRAM Timing Parameters

- $t_{\text{RAC}}$: minimum time from RAS line falling to the valid data output
  - Typically quoted as the speed of a DRAM
  - For a 64 Mbit DRAM, typical $t_{\text{RAC}} = 60$ ns
- $t_{\text{RC}}$: minimum time from start of one row access to the start of the next
  - $t_{\text{RC}} = 110$ ns for a 64 Mbit DRAM with a $t_{\text{RAC}}$ of 60 ns
- $t_{\text{CAC}}$: minimum time from CAS line falling to valid data output
  - 15 ns for a 64 Mbit DRAM with a $t_{\text{RAC}}$ of 60 ns
- $t_{\text{PC}}$: minimum time from start of one column access to start of the next
  - 35 ns for a 64 Mbit DRAM with a $t_{\text{RAC}}$ of 60 ns
- A 60 ns ($t_{\text{RAC}}$) DRAM can
  - perform a row access only every 110 ns ($t_{\text{RC}}$)
  - perform column access ($t_{\text{CAC}}$) in 15 ns, but time between column accesses is at least 35 ns ($t_{\text{PC}}$).
    - External address delays and turning around buses make it 40 to 50 ns
DRAM History

- DRAMs: capacity +60%/yr, cost –30%/yr
  - 2.5X cells/area, 1.5X die size in ~3 years
- Rely on increasing numbers of computers and memory per computer
  - SIMM or DIMM is replaceable unit
  - computers can use any generation DRAM
  - Growth slowing because demand is coming down
- Commodity industry
  - High volume, low profit, conservative
  - Little organization innovation in 20 years
- Order of importance: (primary) Cost/bit, (secondary) Capacity
  - First RAMBUS: 10X BW, +30% cost, but little impact

Improving Main Memory Performance

- Organizations differ in bus width, memory width, and interleaving

  (a) One-word-wide memory organization
  (b) Wide memory organization
  (c) Interleaved memory organization

Simple timing model
1 to send address
6 access time
1 to send data

Word = 32 bits
Cache block = 4 words

1 + 6 + (4 x 1) = 11
4 x (1 + 6 + 1) = 32
1 + 6 + 1 = 8
Generalization: Independent Memory Banks

- Memory banks for independent accesses vs. faster sequential accesses
  - Multiprocessors
  - I/O
  - CPU with Hit under n Misses, Non-blocking Caches
  - Each bank needs separate address and possibly data lines

New terminology
- **Superbank**: all memory active on one block transfer (or **Bank**)
- **Bank**: portion within a superbank that is word-interleaved (or **Subbank**)

- How many banks?
  - Ideally, more than the number of cycles to access word in a bank
    - Optimizes sequential access streams
  - Unfortunately, larger memory chips implies fewer banks

---

DRAMs per PC over Time

<table>
<thead>
<tr>
<th>DRAM Generation</th>
</tr>
</thead>
<tbody>
<tr>
<td>'86</td>
</tr>
<tr>
<td>1 Mb</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>4 MB</td>
</tr>
<tr>
<td>8 MB</td>
</tr>
<tr>
<td>16 MB</td>
</tr>
<tr>
<td>32 MB</td>
</tr>
<tr>
<td>64 MB</td>
</tr>
<tr>
<td>128 MB</td>
</tr>
<tr>
<td>256 MB</td>
</tr>
</tbody>
</table>
Avoiding Bank Conflicts

- Even if we assume that there are lots of banks, run into conflicts

```c
int x[256][512];
for (j = 0; j < 512; j = j+1)
    for (i = 0; i < 256; i = i+1)
        x[i][j] = 2 * x[i][j];
```

- With 128 banks, conflict on word accesses (512 is a multiple of 128)
- Software fixes: loop interchange, or padding array so that it is not $2^k$
- Hardware fix: Prime number of banks, $b$, each with $n$ words
  - Property: No conflicts for any sequence of consecutive addresses, as long as stride is not a multiple of $b$
  - Problem: Resolving the address to a bank number, address within bank
    - bank number = address mod $b$
    - address within bank = address / $b$
    - modulo and divide per memory access are easy if number of banks is $2^k$
    - for prime number of banks, harder (particularly /)

Address Computation w/ Prime Number of Banks

- Fast computation is possible by storing words in banks using modulo interleaving ($b$ banks, $n = 2^c$ words per bank) …
  - bank number = address mod $b$ (same as before)
  - address within bank = address mod $2^c$
- Above result stems from the Chinese Remainder Theorem

<table>
<thead>
<tr>
<th>Bank Number:</th>
<th>Seq. Interleaved</th>
<th>Modulo Interleaved</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0    1    2</td>
<td>0    1    2</td>
</tr>
<tr>
<td>Address within Bank:</td>
<td>0    1    2</td>
<td>0    16   8</td>
</tr>
<tr>
<td>1</td>
<td>3    4    5</td>
<td>9    1    17</td>
</tr>
<tr>
<td>2</td>
<td>6    7    8</td>
<td>18   10   2</td>
</tr>
<tr>
<td>3</td>
<td>9    10   11</td>
<td>3    19   11</td>
</tr>
<tr>
<td>4</td>
<td>12   13   14</td>
<td>12   4    20</td>
</tr>
<tr>
<td>5</td>
<td>15   16   17</td>
<td>21   13   5</td>
</tr>
<tr>
<td>6</td>
<td>18   19   20</td>
<td>6    22   14</td>
</tr>
<tr>
<td>7</td>
<td>21   22   23</td>
<td>15   7    23</td>
</tr>
</tbody>
</table>
Improving Memory Performance in a DRAM

- Increasingly important because fewer chips/system

Evolutionary
- Fast page mode
  - Allow multiple CAS accesses without need for intervening RAS
    - Optimizes sequential access, exploiting the row buffer (1024-2048 bits)
  - Extended Data Out (EDO): 30% faster in page mode
- Synchronous DRAM (SDRAM)
  - Avoid need for handshaking between chip and memory controller
  - Chip also has a register with number of requested bytes: these are transmitted without explicit requests from controller
- Double Data Rate (DDR) DRAM
  - Transmit data from chip on both the falling and rising edge of clock signal

A New DRAM Interface: Rambus
RAMBUS Details

- First-generation interface
  - Dropped RAS/CAS, replacing it with a packet-switched bus
    - Can multiplex address/data on this bus
  - Each chip acts as a memory bank (internally, 4 banks of memory cells)
    - Can transfer variable amount of data from a single request
    - Can perform its own refresh
    - Transfers data on both edges of its clock

- Second-generation interface
  - Separate row- and column-command buses, plus 18-bit data bus
    - Can perform 3 transactions simultaneously
  - 16 internal banks/chip

- RAMBUS improves bandwidth (at cost of latency, price premium)
  - Nintendo game machines (where number of chips is a restriction)

Internal Organization of a RAMBUS DRAM Chip

- Large number of internal banks
  - Shared row-buffers, so not pull parallelism
- Multiplexed data buses
- Packet-switched

- 400 MHz clock signal
  - Data on both rising and falling edge
    - 2 bytes/cycle x 2
    - Data A and B
  - 1.6 GB/s (1 chip)
- Compared to 800-1200 MB/s (DIMM)
SDRAM versus RAMBUS