Advanced Caching Techniques

Approaches to improving memory system performance

- eliminate memory accesses/operations
- decrease the number of misses
- decrease the miss penalty
- decrease the cache/memory access times
- hide memory latencies
- increase cache throughput
- increase memory bandwidth

New techniques address particular components of memory system performance

Handling a Cache Miss the Old Way

1. Send the address & read operation to the next level of the hierarchy
2. Wait for the data to arrive
3. Update the cache entry with data*, rewrite the tag, turn the valid bit on, clear the dirty bit (if data cache)
4. Resend the memory address; this time there will be a hit.

* There are variations:
  - get data before replace the block
  - send the requested word to the CPU as soon as it arrives at the cache (early restart)
  - requested word is sent from memory first; then the rest of the block follows (requested word first)

How do the variations improve memory system performance?
Non-blocking Caches

Non-blocking cache (lockup-free cache)

• allows the CPU to continue executing instructions while a miss is handled
• some caches allow only 1 outstanding miss ("hit under miss")
• some caches have multiple misses outstanding ("miss under miss")
• miss status holding registers (MSHR)
  • hardware structure for tracking outstanding misses
    • physical address of the block
    • which word in the block
    • destination register number (if data)
    • mechanism to merge requests to the same block
    • mechanism to insure accesses to the same location execute in program order

How do non-blocking caches improve memory system performance?
**Non-blocking Caches**

**in-order processors**

- `lw $3, 100($4)` in execution, cache miss
- `add $2, $3, $4` consumer waits until the miss is satisfied
- `sub $5, $6, $7` independent instruction waits for the add

**out-of-order processors**

- `lw $3, 100($4)` in execution, cache miss
- `sub $5, $6, $7` independent instruction can execute during the cache miss
- `add $2, $3, $4` consumer waits until the miss is satisfied

---

**Victim Cache**

**Victim cache**

- small fully-associative cache
  - contains the most recently replaced blocks of a direct-mapped L1 cache
  - if L1 cache miss & victim cache hit, swap the direct-mapped block and victim cache block
  - if both miss, L1 block goes to victim cache
  - alternative to 2-way set-associative cache

**Why do victim caches work?**

**How do victim caches improve memory system performance?**
**Sub-block Placement**

Divide a block into sub-blocks

<table>
<thead>
<tr>
<th>tag</th>
<th>I data</th>
<th>V data</th>
<th>V data</th>
<th>I data</th>
</tr>
</thead>
<tbody>
<tr>
<td>tag</td>
<td>I data</td>
<td>V data</td>
<td>V data</td>
<td>V data</td>
</tr>
<tr>
<td>tag</td>
<td>V data</td>
<td>V data</td>
<td>V data</td>
<td>V data</td>
</tr>
<tr>
<td>tag</td>
<td>I data</td>
<td>I data</td>
<td>I data</td>
<td>I data</td>
</tr>
</tbody>
</table>

- **sub-block** = unit of transfer on a cache miss
- **valid bit**/sub-block
- 2 kinds of misses:
  - block-level miss: tags didn’t match
  - sub-block-level miss: tags matched, valid bit was clear
  + the transfer time of a sub-block
  + fewer tags than if each block was the size of a subblock
- can’t exploit spatial locality

How does sub-block placement improve memory system performance?

---

**Pipelined Cache Access**

**Pipelined cache access**

- simple 2-stage pipeline
  - access the cache
  - data transfer back to CPU
  - tag check & hit/miss logic with the shorter of the two stages

How do pipelined caches improve memory system performance?
Trace Cache

Contains instructions from the *dynamic* instruction stream

- fetch statically noncontiguous instructions in a single cycle, called a trace
- limit on # basic blocks & # instructions in a trace

- instructions may appear more than once
  - accessed with PC & prediction bit
  - traces can contain decoded instructions
  - particularly useful for CISC architectures

Spring 2015 CSE 471: Advanced Caching Techniques
Cache-friendly Compiler Optimizations

Improve spatial locality for data
- **loop interchange**
  - so inner loop follows memory layout of data
- **group & transpose**
  - collects data from different vectors or structures that are accessed together & places them contiguously in memory

Improve temporal locality for data
- **loop fusion**
  - put computations on the same portion of an array from separate loops into one loop
- **tiling (also called blocking)**
  - do all computation on a small block of an array that will fit in the cache

Tiling Example

```c
/*@ 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;
  }

/*@ after */
for (jj=0; jj<n; jj=jj+T)
for (kk=0; kk<n; kk=kk+T)
  for (i=0; i<n; i=i+1)
    for (j=jj; j<min(jj+T-1,n); j=j+1) {
      r = 0;
      for (k=kk; k<min(kk+T-1,n); k=k+1) {
        r = r + y[i,k] * z[k,j];
      }
      x[i,j] = x[i,j] + r;
    }
```

Spring 2015 CSE 471: Advanced Caching Techniques
Memory Banks

Interleaved memory:
- multiple memory banks
- word locations are assigned across banks
- send a single address to all banks at once
- interleaving factor: number of banks

<table>
<thead>
<tr>
<th>Word Address</th>
<th>Bank 0</th>
<th>Word Address</th>
<th>Bank 1</th>
<th>Word Address</th>
<th>Bank 2</th>
<th>Word Address</th>
<th>Bank 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>1</td>
<td></td>
<td>2</td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>5</td>
<td></td>
<td>6</td>
<td></td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>9</td>
<td></td>
<td>10</td>
<td></td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td>13</td>
<td></td>
<td>14</td>
<td></td>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>

Effect on memory system performance?
**Memory Banks**

Independent memory banks
- different banks can be accessed at once, with different addresses
- allows parallel access, possibly parallel data transfer
- multiple memory controllers & separate address lines, one for each access
  - different controllers cannot access the same bank
- less area than dual porting

Effect on memory system performance?

---

**Advanced Caching Techniques**

Approaches to improving memory system performance
- eliminate memory accesses
- decrease the number of misses
- decrease the miss penalty
- hide memory latencies
- increase cache throughput
- increase memory/cache bandwidth
Other Techniques

Hardware or compiler-based prefetching (decreases misses)
Coupling a write-through memory update policy with a write buffer (eliminates store ops/hides store latencies)
TLB (reduce page fault time (penalty))
Cache hierarchies (reduce miss penalty)
Virtual caches (reduce L1 cache access time)
Wider bus (increase bandwidth)