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 processors allow only 1 outstanding miss ("hit under miss")
- some processors allow 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

Non-blocking Caches

Non-blocking cache (**lockup-free cache**)

- can be used with both in-order and out-of-order processors
  - **in-order processors** stall when an instruction that uses the load data is the next instruction to be executed
  - **out-of-order processors** can execute instructions after the load consumer

How do non-blocking caches improve memory system performance?

Victim Cache

Victim cache

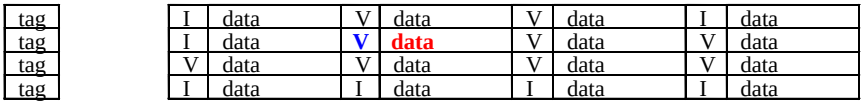
- small fully-associative cache
  - contains the most recently replaced blocks of a direct-mapped L1 cache
  - L1 cache miss & victim cache hit, swap the direct-mapped block and victim cache block
  - both miss, L1 block goes to victim cache
- alternative to 2-way set-associative cache

How do victim caches improve memory system performance?

Why do victim caches work?

Sub-block Placement

Divide a block into sub-blocks



- **sub-block** = unit of transfer on a cache miss
- **valid bit**/sub-block
- 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

\- less implicit prefetching

How does sub-block placement improve memory system performance?

Pipelined Cache Access

Pipelined cache access

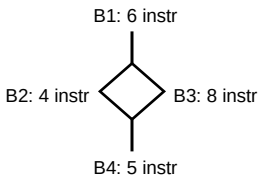
- simple 2-stage pipeline
  - access the cache
  - data transfer back to CPU
  - tag check & hit/miss logic with the shorter of the two stages

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

- Indexed with PC & prediction bit
- traces can contain decoded instructions
  - particularly useful for CISC architectures

Trace Cache

Advantages/disadvantages of a trace cache

+

-

-

Effect on memory system performance?

Cache-friendly Compiler Optimizations

Exploit spatial locality

- **schedule for array misses**
  - hoist first load to each cache block

Improve spatial locality

- **group & transpose**
  - makes data from different vectors that are accessed together contiguous in memory
- **loop interchange**
  - so inner loop follows memory layout

Improve temporal locality

- **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

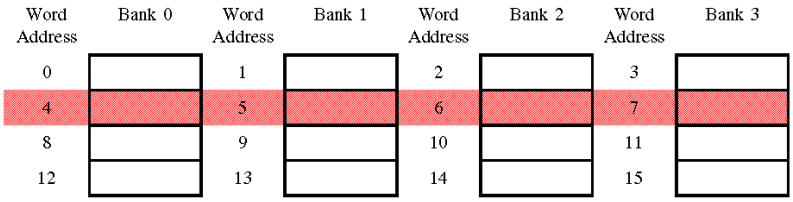
```
/* before */
for (i=0; i<n; i=i+1)
      for (j=0; j<n; j=j+1){
            r = 0;
            for (k=0; k<n; k=k+1) {
                  r = r + y[i,k] * z[k,j]; }
            x[i,j] = r;
            };
/* after */
for (jj=0; jj<n; jj=jj+T)
for (kk=0; kk<n; kk=kk+T)

  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;
            };
```


Memory Banks

Interleaved memory:

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



Memory Banks

Interleaved memory:

+ get more data for one transfer
  - data is probably used *(why?)*
- larger DRAM chip capacity means fewer banks
- power issue

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 bandwidth

Wrap-up

Victim cache (reduce miss penalty)

TLB (reduce page fault time (penalty))

Hardware or compiler-based prefetching (reduce misses)

Cache-conscious compiler optimizations (reduce misses or hide miss penalty)

Coupling a write-through memory update policy with a write buffer (eliminate store ops/hide store latencies)

Handling the read miss before replacing a block with a write-back memory update policy (reduce miss penalty)

Sub-block placement (reduce miss penalty)

Non-blocking caches (hide miss penalty)

Merging requests to the same cache block in a non-blocking cache (hide miss penalty)

Requested word first or early restart (reduce miss penalty)

Cache hierarchies (reduce misses/reduce miss penalty)

Virtual caches (reduce miss penalty)

Pipelined cache accesses (increase cache throughput)

Banked or interleaved memories (increase bandwidth)

Independent memory banks (hide latency)

Wider bus (increase bandwidth)

Trace cache (reduce accesses, reduce misses)