15-740/18-740 Computer Architecture Lecture 20: Caching III

> Prof. Onur Mutlu Carnegie Mellon University Fall 2011, 11/2/2011

Announcements

- CALCM Seminar Today at 4pm
 - Hamerschlag Hall, D-210
 - Edward Suh, Cornell
 - Hardware-Assisted Run-Time Monitoring for Trustworthy Computing Systems
 - http://www.ece.cmu.edu/~calcm/doku.php?id=seminars:semi nar_11_11_02

Milestone II

- Due November 4, Friday
- Please talk with us if you are not making good progress

Review Set 12

- Due Next Monday (Nov 7)
 - Joseph and Grunwald, "Prefetching using Markov Predictors," ISCA 1997.
 - Srinath et al., "Feedback Directed Prefetching: Improving the Performance and Bandwidth-Efficiency of Hardware Prefetchers", HPCA 2007.

Last Lecture

- More caching
 - Write handling, sectored caches
 - Inclusion vs. exclusion
 - Multi-level caching in a pipelined design
 - Miss types: Compulsory, conflict, capacity, coherence/communication
 - Techniques to improve cache performance
 - Victim caches, hashing, pseudo associativity
 - Better data structure layout
 - Blocking



More caching

Review: Improving Hit Rate via Software (II)

Blocking

- Divide loops operating on arrays into computation chunks so that each chunk can hold its data in the cache
- Avoids cache conflicts between different chunks of computation
- Essentially: Divide the working set so that each piece fits in the cache

- But, there are still self-conflicts in a block
 - 1. there can be conflicts among different arrays
 - 2. array sizes may be unknown at compile/programming time

Improving Basic Cache Performance

- Reducing miss rate
 - More associativity
 - Alternatives to associativity
 - Victim caches, hashing, pseudo-associativity, skewed associativity
 - Software approaches
- Reducing miss latency/cost
 - Multi-level caches
 - Critical word first
 - Subblocking/sectoring
 - Multiple outstanding accesses (Non-blocking caches)
 - Multiple accesses per cycle
 - Software approaches

Handling Multiple Outstanding Accesses

- Non-blocking or lockup-free caches
 - Kroft, "Lockup-Free Instruction Fetch/Prefetch Cache Organization," ISCA 1981.
- Question: If the processor can generate multiple cache accesses, can the later accesses be handled while a previous miss is outstanding?
- Idea: Keep track of the status/data of misses that are being handled in Miss Status Handling Registers (MSHRs)
 - A cache access checks MSHRs to see if a miss to the same block is already *pending*.
 - If pending, a new request is not generated
 - If pending and the needed data available, data forwarded to later load
 - Requires buffering of outstanding miss requests

Non-Blocking Caches (and MLP)

- Enable cache access when there is a pending miss
- Enable multiple misses in parallel
 - Memory-level parallelism (MLP)
 - generating and servicing multiple memory accesses in parallel
 - Why generate multiple misses?



- Enables latency tolerance: overlaps latency of different misses
- How to generate multiple misses?
 - Out-of-order execution, multithreading, runahead, prefetching

Miss Status Handling Register

- Also called "miss buffer"
- Keeps track of
 - Outstanding cache misses
 - Pending load/store accesses that refer to the missing cache block
- Fields of a single MSHR
 - Valid bit
 - Cache block address (to match incoming accesses)
 - Control/status bits (prefetch, issued to memory, which subblocks have arrived, etc)
 - Data for each subblock
 - For each pending load/store
 - Valid, type, data size, byte in block, destination register or store buffer entry address

Miss Status Handling Register

1	27	1	1	3	5	5	
Valid	Block Address	Issued	Valid	Туре	Block Offset	Destination	Load/store 0
			Valid	Туре	Block Offset	Destination	Load/store 1
			Valid	Туре	Block Offset	Destination	Load/store 2
			Valid	Туре	Block Offset	Destination	Load/store 3

MSHR Operation

- On a cache miss:
 - Search MSHR for a pending access to the same block
 - Found: Allocate a load/store entry in the same MSHR entry
 - Not found: Allocate a new MSHR
 - No free entry: stall

When a subblock returns from the next level in memory

- Check which loads/stores waiting for it
 - Forward data to the load/store unit
 - Deallocate load/store entry in the MSHR entry
- Write subblock in cache or MSHR
- If last subblock, dellaocate MSHR (after writing the block in cache)

Non-Blocking Cache Implementation

- When to access the MSHRs?
 - □ In parallel with the cache?
 - After cache access is complete?
- MSHRs need not be on the critical path of hit requests
 - Which one below is the common case?
 - Cache miss, MSHR hit
 - Cache hit

Improving Basic Cache Performance

- Reducing miss rate
 - More associativity
 - Alternatives/enhancements to associativity
 - Victim caches, hashing, pseudo-associativity, skewed associativity
 - Software approaches
- Reducing miss latency/cost
 - Multi-level caches
 - Critical word first
 - Subblocking
 - Non-blocking caches
 - Multiple accesses per cycle
 - Software approaches

Reducing Miss Cost/Latency via Software

- Enabling more memory-level parallelism
 - Restructuring code
 - E.g., Pai and Adve, "Code transformations to improve memory parallelism," MICRO 1999.
 - Taking advantage of stall-on-use policy in hardware
- Inserting prefetch instructions

Enabling High Bandwidth Caches

Multiple Instructions per Cycle

- Can generate multiple cache accesses per cycle
- How do we ensure the cache can handle multiple accesses in the same clock cycle?
- Solutions:
 - true multi-porting
 - virtual multi-porting (time sharing a port)
 - multiple cache copies
 - banking (interleaving)

Handling Multiple Accesses per Cycle (I)

- True multiporting
 - Each memory cell has multiple read or write ports
 - + Truly concurrent accesses (no conflicts regardless of address)
 - -- Expensive in terms of area, power, and delay
 - What about read and write to the same location at the same time?
 - Peripheral logic needs to handle this
- Virtual multiporting
 - Time-share a single port
 - Each access needs to be (significantly) shorter than clock cycle
 - □ Used in Alpha 21264
 - Is this scalable?

Handling Multiple Accesses per Cycle (II)

- Multiple cache copies
 - Stores update both caches
 - Loads proceed in parallel
- Used in Alpha 21164
- Scalability?
 - Store operations form a bottleneck
 - Area proportional to "ports"



Handling Multiple Accesses per Cycle (III)

- Banking (Interleaving)
 - Bits in address determines which bank an address maps to
 - Address space partitioned into separate banks
 - Which bits to use for "bank address"?
 - + No increase in data store area
 - -- Cannot always satisfy multiple accesses Bank 0: Why? Bank 0: Even addresses
 - -- Crossbar interconnect in input/output

Bank conflicts

- Two accesses are to the same bank
- □ How can these be reduced?
 - Hardware? Software?

Bank 1:

Odd

addresses

Evaluation of Design Options

- Which alternative is better?
 - true multi-porting
 - virtual multi-porting (time sharing a port)
 - multiple cache copies
 - banking (interleaving)
 - How do we answer this question?
- Simulation
 - See Juan et al.'s evaluation of above options: "Data caches for superscalar processors," ICS 1997.
 - What are the shortcomings of their evaluation?
 - Can one do better with sole simulation?

Caches in Multi-Core Systems

Multi-Core Issues in Caching

Multi-core

- More pressure on the memory/cache hierarchy → cache efficiency a lot more important
- Private versus shared caching
- Providing fairness/QoS in shared multi-core caches
- Migration of shared data in private caches
- How to organize/connect caches:
 - Non-uniform cache access and cache interconnect design
- Placement/insertion
 - Identifying what is most profitable to insert into cache
 - Minimizing dead/useless blocks
- Replacement
 - Cost-aware: which block is most profitable to keep?

Cache Coherence

Basic question: If multiple processors cache the same block, how do they ensure they all see a consistent state?











Cache Coherence: Whose Responsibility?

Software

- Can the programmer ensure coherence if caches are invisible to software?
- What if the ISA provided the following instruction?
 - FLUSH-LOCAL A: Flushes/invalidates the cache block containing address A from a processor's local cache
 - When does the programmer need to FLUSH-LOCAL an address?
- What if the ISA provided the following instruction?
 - FLUSH-GLOBAL A: Flushes/invalidates the cache block containing address A from all other processors' caches
 - When does the programmer need to FLUSH-GLOBAL an address?

Hardware

- Simplifies software's job
- One idea: Invalidate all other copies of block A when a processor writes to it

Snoopy Cache Coherence

- Caches "snoop" (observe) each other's write/read operations
- A simple protocol:



- Write-through, nowrite-allocate cache
- Actions: PrRd, PrWr, BusRd, BusWr

Multi-core Issues in Caching

- How does the cache hierarchy change in a multi-core system?
- Private cache: Cache belongs to one core
- Shared cache: Cache is shared by multiple cores





Shared Caches Between Cores

- Advantages:
 - Dynamic partitioning of available cache space
 - No fragmentation due to static partitioning
 - Easier to maintain coherence
 - Shared data and locks do not ping pong between caches
- Disadvantages
 - Cores incur conflict misses due to other cores' accesses
 - Misses due to inter-core interference
 - Some cores can destroy the hit rate of other cores
 What kind of access patterns could cause this?
 - Guaranteeing a minimum level of service (or fairness) to each core is harder (how much space, how much bandwidth?)
 - □ High bandwidth harder to obtain (N cores \rightarrow N ports?)
 - Potentially higher latency (interconnect between cores-cache)