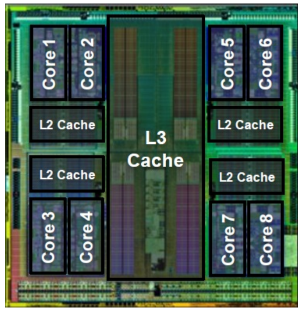
Lecture 6

High B/W Caches

Fall 2016

Pejman Lotfi-Kamran

AMD FX



Adapted from slides originally developed by Profs. Falsafi, Hill, Hoe, Lipasti, Shen, Smith, Sohi, and Vijaykumar of Carnegie Mellon University, EPFL, Purdue University, and University of Wisconsin.

Where Are We?

Fr	Sa	Su	Mo	Tu
	27-Shahrivar		29-Shahrivar	
	3-Mehr		5-Mehr	
	10-Mehr		12-Mehr	
	17-Mehr		19-Mehr	
	24-Mehr		26-Mehr	
	1-Aban		3-Aban	
	8-Aban		10-Aban	
	15-Aban		17-Aban	
	22-Aban		24-Aban	
	29-Aban		1-Azar	
	6-Azar		8-Azar	
	13-Azar		15-Azar	
	20-Azar		22-Azar	
	27-Azar		29-Azar	
	4-Dey		6-Dey	

This Lecture
- High-B/W Caches

Next Lecture:
- Prefetching

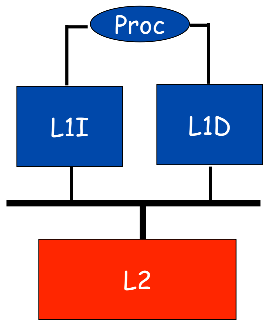
Multi-Level Caches

Processors getting faster w.r.t. main memory
- larger caches to reduce frequency of more costly misses
- but larger caches are too slow for processor

=> gradually reduce cost of misses with a multiple cache levels

$$t_{avg} = t_{hit} + \text{miss ratio} \times t_{miss}$$

Multi-Level Cache Design



different requirements
⇒ different choice of
capacity
block size
associativity

$$t_{avg\text{-}L1} = t_{hit\text{-}L1} + \text{miss-ratio}_{L1} \times t_{avg\text{-}L2}$$

$$t_{avg\text{-}L2} = t_{hit\text{-}L2} + \text{miss-ratio}_{L2} \times t_{memory}$$

The Inclusion Property

Multi-level inclusion holds if L2 is always a superset of L1, and so on down the hierarchy. Why?

- if an addr is in L1, then it must be frequently used
- L2 can handle external coherence checks without L1

Causes interesting interactions between L1I, L1D, and L2

Inclusion takes effort to maintain

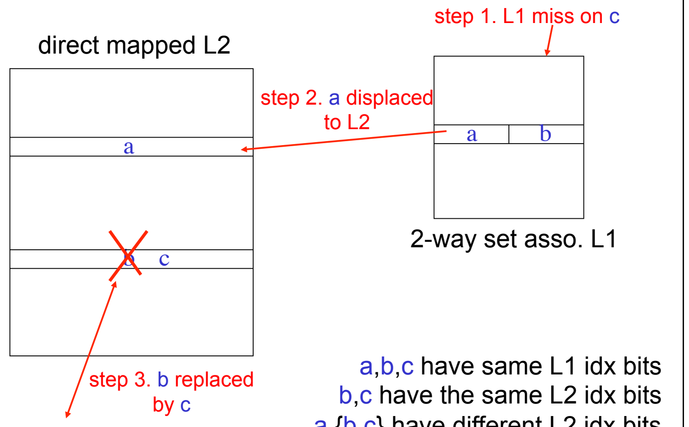
- each L2 block frame needs status bits marking subblocks in cached by L1 (# of bit = L2 blocksize/L1 blocksize)
- if a L2 block is to be displaced, must first flush out all of its L1-cached subblocks

Why would an L2 block be displaced

if it is still has subblocks in L1?

Consider 1. L1 block size < L2 block size 2. associativity in L1

Possible Inclusion Violation



a,b,c have same L1 idx bits

b,c have the same L2 idx bits

a,{b,c} have different L2 idx bits

Multi-Level Inclusion (Cont.)

Inclusion takes effort to maintain

- make L2 cache have bits or pointers giving L1 contents
- invalidate from L1 before replacing from L2
- number of pointers per L2 block
  - L2 blocksize/L1 blocksize

Interesting interaction between L1I, L1D, and L2

- Example a matrix multiply program for a large matrix
- What happens if you maintain inclusion for L1I?

Level Two Cache Design

What is miss ratio?

- global: L2 misses after L1 references
- local: L2 misses after L1/L1 misses
- solo: as only cache / references

L1 cache design similar to single-level cache

- when main memory “faster” w.r.t. CPU

Apply previous experience to L2 design?

L2 “global” miss ratios not significantly altered by L1 presence

- if L2 cache size >= 8 x L1 cache size
- Przybylski et al., ISCA 1989

But L2 caches bigger than before

Level Two Cache Example

Recall adding associativity to single-level cache helped if

$diff(t_{cache}) + diff(\text{miss}) \times t_{miss} < 0$

$diff(\text{miss}) = -1\%,\ t_{miss} = 20$

$\Rightarrow diff(t_{cache}) < 0.2$ cycle

Consider doing the same in an L2 cache where

$t_{avg} = t_{cache1} + \text{miss1} \times t_{cache2} + \text{miss2} \times t_{memory}$

Improvement only if

$\text{miss1} \times diff(t_{cache2}) + diff(\text{miss2}) \times t_{memory} < 0$

$diff(t_{cache2}) < (-diff(\text{miss2})/\text{miss1}) \times t_{memory}$

$diff(t_{cache2}) < 0.0005/0.05 \times 100 = 1$ cycle

Lock-up Free Caches

Also known as non-blocking caches

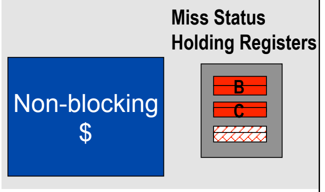
Proposed first by Kroft

Only makes sense if processor
- handle mult. pending references
- can do useful work under a miss
- has misses that can be overlapped

Key implementation problems
- handle reads to pending miss
- handle writes to pending miss
- keep multiple requests straight

Memory Access Stream

ld A	hit
ld B	miss
ld C	miss
ld D	hit
st B	miss (pend.)



Lock-up Free Caches (Cont.)

MSHRs: miss status holding registers

1. Is there already a miss to the same block?
2. Route data back to CPU

- Valid bit and tag: associatively compared on each miss
- Status & pointer to block frame
- What happens on a miss?
- Tag L1 requests to allow out-of-order service from L2
- Split-transaction/pipelined L1-L2 interface
- Associative MSHRs could become bottleneck

Issue Bandwidth

Increasing issue width => wider caches

Parallel cache access is harder than parallel FUs
- fundamental difference: caches have state, FUs don’t
- one port affects future for other ports

Several approaches used
- true multi-porting
- multiple cache copies
- virtual multi-porting
- multi-banking (interleaving)
- line buffers

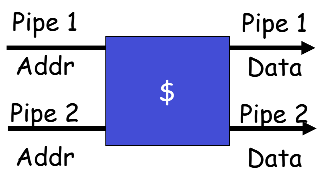
Multi-Port Caches: True Multiporting

Superscalar processors requires multiple data references per cycle

Time-multiplex a single port (double pump)
- need cache access to be faster than datapath clock
- not scalable

Truly multiported SRAMs are available, but
- more chip area
- slower access

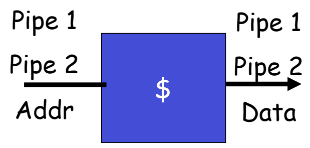
(very undesirable for L1-D)



Virtual Multi-Porting

Used in Power2 and Alpha 21164
- 21164 uses wave pipelining

Time-share a single port
- may require access to be faster than clock
- probably not scalable beyond 2



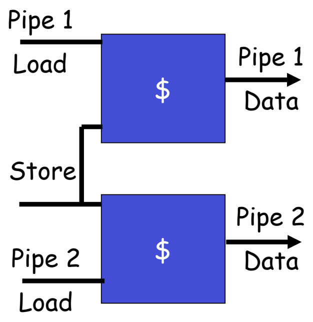
Multiple Cache Copies

Used in Alpha 21264

Independent fast load paths

Single shared store path

Not a scalable solution
- Store is a bottleneck
- Doubles area



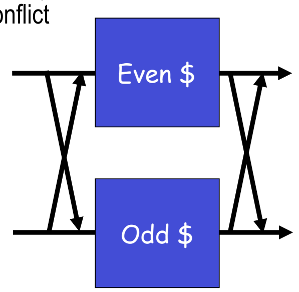
Multi-Banking (Interleaving) Caches

Address space is statically partitioned and assigned to different caches *Which addr bit to use for partitioning?*

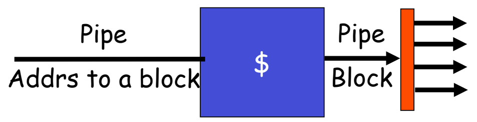
A compromise (e.g. Intel P6, MIPS R10K)
- multiple references per cyc. if no conflict
- only one reference goes through if conflicts are detected
- the rest are deferred

(bad news for scheduling logic)

Most helpful is compiler knows about the interleaving rules



Line Buffers (L0)



Allows multiple ops to/from a block

Increases bandwidth for wide issue

Juan et al.'s Simulation Study

Focus on two solutions
- true multiporting: ideal
- multi-banking: more scalable

Use dynamically scheduled processor
- 4-issue, 64-inst window to 32 issue, 512-inst window

Direct-mapped 32KB cache
- processor-cache ports
- n-banks
- cell-port (internal)

Multi-banked Caches

Configuration
- each bank is blocking but interleaving gives non-blocking
- addresses use simple interleaving
- use 4 processor-to-cache ports

Performance of Multi-banked Caches

Bank conflicts is a problem
- multi-ported vs. multi-banked non-blocking

Bank blocking hurts for small # of banks
- multi-banked non-blocking vs. multi-banked blocking

Nature of Bank Conflicts

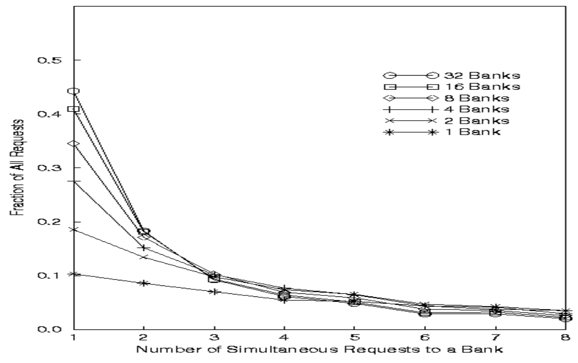
Many requests are to the same bank

But many are to the same block

- 0.75 of pending requests with 4 banks are to same line

Lecture 6
Slide 21

Bank Conflicts



Lecture 6
Slide 22

Alternative Design

Multi-banks with multi-ports per bank

- improves performance considerably
- compare 4 banks 1 port with 2 banks 2 ports

Add FIFOs to banks

- absorbs some bank busy stalls

Add coalescing to FIFOs

- combines requests to same line

Lecture 6
Slide 23

Area/Performance

Multi-bank with same-line optimization seems to work best

Standard multi-bank caches have relatively low cost

- but also lower performance

Hybrid designs perform well but at relatively high cost

Lecture 6
Slide 24

Latency vs. Bandwidth

Latency can be handled by

- hiding/tolerating techniques
  - e.g., parallelism
  - may increase bandwidth demand
- reducing techniques

Ultimately limited by physics

Latency vs. Bandwidth (Cont.)

Bandwidth can be handled by

- banking/interleaving/multiporting
- wider buses
- hierarchies (multiple levels)

What happens if average demand not supplied?

- bursts are smoothed by queues
  - if burst is much larger than average => long queue
  - eventually increases delay to unacceptable levels