6.175 Final Project
Part 0: Understanding Non-Blocking Caches and Cache Coherency

Answers

Notation

- Addresses are ordered triples:
  - (tag, index, offset)
- Cache lines are addressed with ordered pairs:
  - (tag, index)
- Cache slots are addressed by index
- Reading a cache line from memory:
  - M[(tag, index)]

Non-Blocking Cache

- Given: Processor requests and memory responses
- Assignment: Complete the following tables (not all cells should be filled)
  - We will focus on Loads first and Stores second
  - In later tables we integrate Loads and stores together

Multiple Requests in Flight – Part 1

Processor		Memory		Slot 0		Slot 1		Slot 2		Slot 3	
Req	Resp	Req	Resp	V	W	V	W	V	W	V	W
				0	0	0	0	0	0	0	0
1: Ld (0,0,0)		Ld(0,0)		0	1	0	0	0	0	0	0
				0	1	0	0	0	0	0	0
2: Ld (0,1,0)		Ld (0,1)		0	1	0	1	0	0	0	0
				0	1	0	1	0	0	0	0
3: Ld (0,2,0)		Ld (0,2)		0	1	0	1	0	1	0	0
				0	1	0	1	0	1	0	0
4: Ld (0,3,0)		Ld (0,3)		0	1	0	1	0	1	0	1
				0	1	0	1	0	1	0	1
				0	1	0	1	0	1	0	1

Multiple Requests in Flight – Part 2

Processor		Memory		Slot 0		Slot 1		Slot 2		Slot 3	
Req	Resp	Req	Resp	V	W	V	W	V	W	V	W
				0	1	0	1	0	1	0	1
			M[(0,0)]	1	0	0	1	0	1	0	1
	Ld 1 – data			1	0	0	1	0	1	0	1
			M[(0,1)]	1	0	1	0	0	1	0	1
	Ld 2 – data			1	0	1	0	0	1	0	1
			M[(0,3)]	1	0	1	0	0	1	1	0
	Ld 4 – data			1	0	1	0	0	1	1	0
			M[(0,2)]	1	0	1	0	1	0	1	0
	Ld 3 – data			1	0	1	0	1	0	1	0
				1	0	1	0	1	0	1	0

Out of order resp

Same Cache Line, Different Offset

Processor		Memory		Slot 0		other
Req	Resp	Req	Resp	V	W	# elem in LdQ
				0	0	0
1: Ld (0,0,0)		Ld (0,0)		0	1	1
				0	1	1
2: Ld (0,0,1)				0	1	2
				0	1	2
			M[(0,0)]	1	0	2
	Ld 1 – data			1	0	1
	Ld 2 – data			1	0	0
3: Ld (0,0,2)	Ld 3 – data	Ld Hit		1	0	0
				1	0	0
4: Ld (0,0,3)	Ld 4 – data	Ld Hit		1	0	0
				1	0	0

Same Index, Different Tag

Processor		Memory		Slot 0		
Req	Resp	Req	Resp	V	W	Tag
				0	0	?
1: Ld (0,0,0)		Ld (0,0)		0	1	0
				0	1	0
2: Ld (1,0,0)				0	1	0
				0	1	0
			M[(0,0)]	1	0	0
	Ld 1 – data			1	0	0
Search LdQ for next Req		Ld (1,0)		0	1	1
				0	1	1
			M[(1,0)]	1	0	1
	Ld 2 – data			1	0	1
				1	0	1

Stores

Processor		Memory		Slot 0			# elements in
Req	Resp	Req	Resp	V	W	D	StQ
				0	0	0	0
1: St x (0,0,0)		Ld (0,0)		0	1	0	1
				0	1	0	1
2: St y (0,0,1)				0	1	0	2
				0	1	0	2
			M[(0,0)]	1	0	?	2
	St 1 – ACK			1	0	x	1
3: St z (0,0,2)	St 2 – ACK			1	0	x	0
3: St z (0,0,2)				1	0	x	0
3: St z (0,0,2)	St 3 - ACK			1	0	x	0
				1	0	x	0
				1	0	x	0

Cache can’t accept Req while handling memory Resp

Store Bypassing

Processor		Memory		Slot 0				# elements in	
Req	Resp	Req	Resp	V	W	D	Data(0)	StQ	LdQ
				0	0	0	-	0	0
1: Ld (0,0,0)		Ld (0,0)		0	1	0	-	0	1
				0	1	0	-	0	1
2: St y (0,0,0)				0	1	0	-	1	1
3: St z (0,0,0)				0	1	0	-	2	1
4: Ld (0,0,0)	Ld 4 – z	Hit from StQ		0	1	0	-	2	1
				0	1	0	-	2	1
			M[(0,0)]	1	0	0	?	2	1
	Ld 1 – ?			1	0	0	?	2	0
	St 2 – ACK			1	0	1	y	1	0
	St 3 – ACK			1	0	1	z	0	0
				1	0	1	z	0	0

Resending Requests

Processor		Memory		Slot 0				# elements in	
Req	Resp	Req	Resp	V	W	D	Tag	StQ	LdQ
				0	0	0	?	0	0
1: St y (0,1,0)		Ld (0,1)		0	0	0	?	1	0
2: St z (0,0,0)		Ld (0,0)		0	1	0	0	2	0
			M[(0,0)]	1	0	0	0	2	0
3: Ld (1,0,0)		Ld (1,0)		0	1	0	1	2	1
				0	1	0	1	2	1
			M[(0,1)]	0	1	0	1	2	1
	St 1 – ACK			0	1	0	1	1	1
			M[(1,0)]	1	0	0	1	1	1
	Ld 3 – data			1	0	0	1	1	0
Req sent from head of StQ		Ld (0,0)		0	1	0	0	1	0
			M[(0,0)]	1	0	0	0	1	0
	St 2 – ACK			1	0	1	0	0	0

Cache Coherency

- Given: Initial cache states for a single address and a cache request for that address
- Assignment: Write the rules each module needs to execute to perform the cache request
  - You may have to keep track of what messages are still in the message network. Unfortunately there is not enough space to include it in the table.

No Contest: Cache 0 - Ld

Cache 0			Cache 1			Parent				
						Cache 0		Cache 1		rule
state	waitp	rule	state	waitp	rule	state	waitc	state	waitc	
I			I			I		I		
I	S	1	I			I		I		
I	S		I			S		I		2
S		3	I			S		I		

Other Cache is Writing: Cache 0 – Ld

Cache 0			Cache 1			Parent				
						Cache 0		Cache 1		rule
state	waitp	rule	state	waitp	rule	state	waitc	state	waitc	
I			M			I		M		
I	S	1	M			I		M		
I	S		M			I		M	S	4
I	S		S		5	I		M	S	
I	S		S			I		S		6
I	S		S			S		S		2
S		3	S			S		S		

Lots of Downgrading: Cache 0 – St

M state for different tag. Need to first evict this line, and then upgrade to M for the desired tag

Cache 0			Cache 1			Parent				
						Cache 0		Cache 1		rule
state	waitp	rule	state	waitp	rule	state	waitc	state	waitc	
M			M			M		M		
I	M	8, 1	M			M		M		
I	M		M			I		M		6
I	M		M			I		M	I	4
I	M		I		5	I		M	I	
I	M		I			I		I		6
I	M		I			M		I		2
M		3	I			M		I		

Bonus: Both want to write

Cache 0			Cache 1			Parent				
						Cache 0		Cache 1		rule
state	waitp	rule	state	waitp	rule	state	waitc	state	waitc	
S			S			S		S		
S	M	1	S	M	1	S		S		
S	M		S	M		S		S	I	4 (from 0)
S	M		I	M	5	S		S	I	
S	M		I	M		S		I		6
S	M		I	M		M		I		2
M		3	I	M		M	I	I		4 (from 1)
I		5	I	M		M	I	I		
I			I	M		I		I		6
I			I	M		I		M		2
I			M		4	I		M		

The Rest of the Project – Part 1

- Building a non-blocking cache hierarchy and testing with simulated use cases
  - This includes designing modules for Message FIFOs, the Message Network, the Cache Parent Processor, and the Non-blocking Caches
  - Some of the included tests are identical to the executions shown in Part 0
    - It is important to know what the modules are supposed to do when debugging!

Part 1: Non-Blocking Coherent Cache Summary

- Request from processor:
  - If Ld request:
    - If in StQ – return data
    - If in cache – return data
    - Otherwise:
      - Enqueue into LdQ
      - Send downgrade response* and upgrade request if possible
  - If St request:
    - If cache hit and StQ empty – write to cache
    - Otherwise:
      - Enqueue into StQ
      - Send downgrade response* and upgrade request if possible

*Downgrade responses are not always necessary

Part 1: Non-Blocking Coherent Cache Summary

- Message from Parent:
  - If upgrade response:
    - Update cache line
    - Search LdQ and return responses until no more hits (multiple cycles)
    - Write to cache for head of StQ until cache miss (multiple cycle)
    - Send upgrade to M request for head of StQ (if possible)
    - Send upgrade to S request for LdQ entry with index matching the response (if possible)
  - If downgrade request:
    - Update cache line (if necessary)
    - Send response (if necessary)

The Rest of the Project – Part 2

- Integrating the Non-Blocking Cache with an out-of-order processor core to create a multicore SMIPS processor
  - Requires adding support for LL (load-link) and SC (store-conditional) instruction and memory fences.
  - This will also include

The Rest of the Project – Timeline

- Part 1:
  - Distributed in waves this weekend
  - Finish before checkpoint meetings
- Checkpoint Meetings:
  - Wednesday, December 3rd and Friday, December 5th during class time
  - You will sign up for slots soon
- Part 2:
  - Distributed around the time of the checkpoint meetings
  - Due December 10th at 3 PM – Strict deadline!
- Presentations:
  - December 10th from 3 PM to 6 PM in 32-D463 (Star)
  - Includes Pizza!