Homework 3 Solution

CSE 661 - Parallel and Vector Architectures

5.3 (? pts) Given: 50% of instructions are loads and stores
Private reads = 70%, private writes = 20%, shared reads = 8%, shared writes = 2%
Cache block size = 16 bytes
Hit rates: private data = 97%, shared data = 95%, instructions = 98.5%
Bus has 64 data lines and 32 address lines
Processor clock twice as fast as that of a bus and CPI = 2.0 without memory penalties
Memory latency = 2 bus cycles
Probability of replacing a modified block = 0.3

(a) Write-through caches with write-allocate

Bus cycles for each type of hit and miss (ignoring cycles for cache consistency and bus contention):

Read miss: 1-cycle address + 2-cycle memory latency (given) + 2-cycle data transfer
16 bytes are transferred with 8 bytes per cycle (64-bit data bus) = 2 bus-cycle transfer
Bus read = 5 bus cycles = 10 CPU cycles

Write miss (write-through + write allocate): 1-cycle address + 2-cycle memory latency + 2-cycle read + 1-cycle write through = 6 bus cycles = 12 CPU cycles
Write miss is equivalent to a read miss + write through

Read hit: no bus transaction

Write hit (write-through): 1-cycle address overlapped with 1-cycle data transfer
Only word being written is transferred, not entire block
Write hit = 1 bus cycle = 2 CPU cycles.

Average bus cycles to execute 100 instructions

Instruction fetch cycles = 100 * (1 – 0.985) * 5 bus cycles = 7.50 bus cycles
Private data read miss cycles: 50 * 0.70 * (1 – 0.97) * 5 bus cycles = 5.25 bus cycles
Private data write miss cycles: 50 * 0.20 * (1 – 0.97) * 6 bus cycles = 1.80 bus cycles
Private data write hit cycles: 50 * 0.20 * 0.97 * 1 bus cycle = 9.70 bus cycles
Shared data read miss cycles: 50 * 0.08 * (1 – 0.95) * 5 bus cycles = 1.00 bus cycles
Shared data write miss cycles: 50 * 0.02 * (1 – 0.95) * 6 bus cycles = 0.30 bus cycles
Shared data write hit cycles: 50 * 0.02 * 0.95 * 1 bus cycle = 0.95 bus cycles

Total = 26.50 bus cycles = 53 CPU cycles per 100 instructions

Therefore, to execute 100 instructions, we have:
100 * 2 CPI = 200 CPU cycles without considering memory penalties +
53 CPU stall cycles to access memory = 253 cycles
So a single processor uses 53/253 = 20.9% of its time using the bus, or
At most 253/53 = 4.77 truncated to **4 processors** can be supported

(b) Write-back caches:

The write hits take no bus cycles. Each cache miss though has the possibility of a writeback (the whole cache line must be written back)

Read miss: 1-cycle address + 2-cycle memory latency + 2-cycle transfer = 5 bus cycles
Write miss: 1-cycle address + 2-cycle memory latency + 2-cycle transfer = 5 bus cycles
Writeback: 1-cycle address overlapped with 2-cycle data transfer = 2 bus cycles

Adjusted Read miss with Writeback: 5 + 0.3 * 2 = 5.6 bus cycles
Adjusted Write miss with Writeback: 5 + 0.3 * 2 = 5.6 bus cycles

Average bus cycles to execute 100 instructions:
Instruction fetch cycles = 100 * (1 – 0.985) * 5 bus cycles = 7.50 bus cycles
Private data read miss cycles: 50 * 0.70 * (1 – 0.97) * 5.6 bus cycles = 5.88 bus cycles
Private data write miss cycles: 50 * 0.20 * (1 – 0.97) * 5.6 bus cycles = 1.68 bus cycles
Shared data read miss cycles: 50 * 0.08 * (1 – 0.95) * 5.6 bus cycles = 1.12 bus cycles
Shared data write miss cycles: 50 * 0.02 * (1 – 0.95) * 5.6 bus cycles = 0.28 bus cycles

Total = 16.46 Bus cycles = 32.92 CPU cycles per 100 instructions

Therefore, to execute 100 instructions, we have:
100 * 2 CPI = 200 CPU cycles without considering memory penalties +
32.92 CPU stall cycles to access memory = 232.92 CPU cycles
So a single processor uses 32.92/232.92 = 14.1% of its time using the bus, or
At most 232.92/32.92 = 7.08 truncated to **7 processors** can be supported

5.4 (? pts) Given the following cost model:
read/write cache hit = 1 cycle
misses requiring simple transaction on bus (BusUpgr, BusUpd) = 60 cycles
misses requiring whole cache block transfer = 90 cycles

(a) Illinois MESI protocol

Stream 1:

Operation	P1	P2	P3	Bus Action	Cycles
Read1	E			BusRd(~S)	90
Write1	M				1
Read1	M				1
Write1	M				1
Read2	S	S		BusRd(S), Flush	90
Write2	I	M		BusUpgr	60
Read2	I	M			1
Write2	I	M			1
Read3	I	S	S	BusRd(S), Flush	90
Write3	I	I	M	BusUpgr	60
Read3	I	I	M		1
Write3	I	I	M		1
				Total	397

Stream 2:

Operation	P1	P2	P3	Bus Action	Cycles
Read1	E			BusRd(~S)	90
Read2	S	S		BusRd(S)	90
Read3	S	S	S	BusRd(S)	90
Write1	M	I	I	BusUpgr	60
Write2	I	M	I	BusRdX, Flush	90
Write3	I	I	M	BusRdX, Flush	90
Read1	S	I	S	BusRd(S), Flush	90
Read2	S	S	S	BusRd(S), Flush	90
Read3	S	S	S		1
Write3	I	I	M	BusUpgr	60
Write1	M	I	I	BusRdX, Flush	90
				Total	841

Stream 3:

Operation	P1	P2	P3	Bus Action	Cycles
Read1	E			BusRd(~S)	90
Read2	S	S		BusRd(S)	90
Read3	S	S	S	BusRd(S)	90
Read3	S	S	S		1
Write1	M	I	I	BusUpgr	60
Write1	M	I	I		1
Write1	M	I	I		1
Write1	M	I	I		1
Write2	I	M	I	BusRdX, Flush	90
Write3	I	I	M	BusRdX, Flush	90
				Total	514

(b) Dragon protocol:

Stream 1:

Operation	P1	P2	P3	Bus Action	Cycles
Read1	E			BusRd(~S)	90
Write1	M				1
Read1	M				1
Write1	M				1
Read2	Sm	Sc		BusRd(S), C2C	90
Write2	Sc	Sm		BusUpd(S)	60
Read2	Sc	Sm			1
Write2	Sc	Sm		BusUpd(S)	60
Read3	Sc	Sm	Sc	BusRd(S), C2C	90
Write3	Sc	Sc	Sm	BusUpd(S)	60
Read3	Sc	Sc	Sm		1
Write3	Sc	Sc	Sm	BusUpd(S)	60
				Total	515

Stream 2:

Operation	P1	P2	P3	Bus Action	Cycles
Read1	E			BusRd(~S)	90
Read2	Sc	Sc		BusRd(S)	90
Read3	Sc	Sc	Sc	BusRd(S)	90
Write1	Sm	Sc	Sc	BusUpd(S)	60
Write2	Sc	Sm	Sc	BusUpd(S)	60
Write3	Sc	Sc	Sm	BusUpd(S)	60
Read1	Sc	Sc	Sm		1
Read2	Sc	Sc	Sm		1
Read3	Sc	Sc	Sm		1
Write3	Sc	Sc	Sm	BusUpd(S)	60
Write1	Sm	Sc	Sc	BusUpd(S)	60
				Total	573

Stream 3:

Operation	P1	P2	P3	Bus Action	Cycles
Read1	E			BusRd(~S)	90
Read2	Sc	Sc		BusRd(S)	90
Read3	Sc	Sc	Sc	BusRd(S)	90
Read3	Sc	Sc	Sc		1
Write1	Sm	Sc	Sc	BusUpd(S)	60
Write1	Sm	Sc	Sc	BusUpd(S)	60
Write1	Sm	Sc	Sc	BusUpd(S)	60
Write1	Sm	Sc	Sc	BusUpd(S)	60
Write2	Sc	Sm	Sc	BusUpd(S)	60
Write3	Sc	Sc	Sm	BusUpd(S)	60
				Total	631

5.10 (? pts) Four-processor bus-based multiprocessor

Each processor executes test&set lock to gain access to a null critical section
Test&set always goes on the bus and it takes the same time as a normal read transaction.
Initial condition: processor1 has the lock and processors 2, 3, 4 are spinning on their caches waiting for the lock to be released.

Test-and-Test&Set algorithm is used

(a) Best-case number of bus transactions = 7

Trans	Action	P1	P2	P3	P4	Comment
		S	S	S	S	Initial State
1: BusUpgr	P1: st loc, #0	M	I	I	I	P1 releases lock
2: BusRd, Flush	P2: ld reg, loc	S	S	I	I	P2 reads lock and finds it 0
3: BusUpgr	P2: t&s reg, loc	I	M	I	I	P2 acquires lock
	P2: st loc, #0	I	M	I	I	P2 releases – No transaction
4: BusRd, Flush	P3: ld reg, loc	I	S	S	I	P3 reads lock and finds it 0
5: BusUpgr	P3: t&s reg, loc	I	I	M	I	P3 acquires lock
	P3: st loc, #0	I	I	M	I	P3 releases – No transaction

6: BusRd, Flush	P4: ld reg, loc	I	I	S	S	P4 reads lock and finds it 0
7: BusUpgr	P4: t&s reg, loc	I	I	I	M	P4 acquires lock
	P4: st loc, #0	I	I	I	M	P4 releases – No transaction

(b) Worst-case number of bus transactions = 15

Transaction	Action	P1	P2	P3	P4	Comment
		S	S	S	S	Initial State
1: BusUpgr	P1: st loc, #0	M	I	I	I	P1 releases lock
2: BusRd, Flush	P2: ld reg, loc	S	S	I	I	P2 reads lock and finds it 0
3: BusRd	P3: ld reg, loc	S	S	S	I	P3 reads lock and finds it 0
4: BusRd	P4: ld reg, loc	S	S	S	S	P4 reads lock and finds it 0
5: BusUpgr	P2: t&s reg, loc	I	M	I	I	P2 acquires lock
6: BusRdT, Flush	P3: t&s reg, loc	I	S	S	I	P3 fails to acquire lock
7: BusRdT	P4: t&s reg, loc	I	S	S	S	P4 fails to acquire lock
8: BusUpgr	P2: st loc, #0	I	M	I	I	P2 releases lock
9: BusRd, Flush	P3: ld reg, loc	I	S	S	I	P3 reads lock and finds it 0
10: BusRd	P4: ld reg, loc	I	S	S	S	P4 reads lock and finds it 0
11: BusUpgr	P3: t&s reg, loc	I	I	M	I	P3 acquires lock
12:BusRdT,Flush	P4: t&s reg, loc	I	I	S	S	P4 fails to acquires lock
13: BusUpgr	P3: st loc, #0	I	I	M	I	P3 releases lock
14: BusRd, Flush	P4: ld reg, loc	I	I	S	S	P4 reads lock and finds it 0
15: BusUpgr	P4: t&s reg, loc	I	I	I	M	P4 acquires lock
	P4: st loc, #0	I	I	I	M	P4 releases – No transaction

The assumption here is that a miss on a test&set generates a single **BusRdT** (Bus Read for Test&Set) that invalidates other caches (equivalent to Bus read exclusive **BusRdX**) when it succeeds, but is equivalent to a Bus Read **BusRd** when it fails.

(c) Dragon Protocol

Best-case = 7 transactions

Trans	Action	P1	P2	P3	P4	Comment
		Sm	Sc	Sc	Sc	Initial State
1: BusUpd	P1: st loc, #0	Sm	Sc	Sc	Sc	P1 releases lock
	P2: ld reg, loc	Sm	Sc	Sc	Sc	P2 reads lock and finds it 0
2: BusUpd	P2: t&s reg, loc	Sc	Sm	Sc	Sc	P2 acquires lock
3: BusUpd	P2: st loc, #0	Sc	Sm	Sc	Sc	P2 releases lock
	P3: ld reg, loc	Sc	Sm	Sc	Sc	P3 reads lock and finds it 0
4: BusUpd	P3: t&s reg, loc	Sc	Sc	Sm	Sc	P3 acquires lock
5: BusUpd	P3: st loc, #0	Sc	Sc	Sm	Sc	P3 releases lock
	P4: ld reg, loc	Sc	Sc	Sm	Sc	P4 reads lock and finds it 0
6: BusUpd	P4: t&s reg, loc	Sc	Sc	Sc	Sm	P4 acquires lock
7: BusUpd	P4: st loc, #0	Sc	Sc	Sc	Sm	P4 releases lock

Worst-case = 7 transactions

Trans	Action	P1	P2	P3	P4	Comment
		Sm	Sc	Sc	Sc	Initial State
1: BusUpd	P1: st loc, #0	Sm	Sc	Sc	Sc	P1 releases lock
	P2: ld reg, loc	Sm	Sc	Sc	Sc	P2 reads lock and finds it 0
	P3: ld reg, loc	Sm	Sc	Sc	Sc	P3 reads lock and finds it 0
	P4: ld reg, loc	Sm	Sc	Sc	Sc	P4 reads lock and finds it 0
2: BusUpd	P2: t&s reg, loc	Sc	Sm	Sc	Sc	P2 acquires lock
	P3: t&s reg, loc	Sc	Sm	Sc	Sc	P3 fails to acquires lock
	P4: t&s reg, loc	Sc	Sm	Sc	Sc	P4 fails to acquires lock
3: BusUpd	P2: st loc, #0	Sc	Sm	Sc	Sc	P2 releases lock
	P3: ld reg, loc	Sc	Sm	Sc	Sc	P3 reads lock and finds it 0
	P4: ld reg, loc	Sc	Sm	Sc	Sc	P4 reads lock and finds it 0
4: BusUpd	P3: t&s reg, loc	Sc	Sc	Sm	Sc	P3 acquires lock
	P4: t&s reg, loc	Sc	Sc	Sm	Sc	P4 fails to acquires lock
5: BusUpd	P3: st loc, #0	Sc	Sc	Sm	Sc	P3 releases lock
	P4: ld reg, loc	Sc	Sc	Sm	Sc	P4 reads lock and finds it 0
6: BusUpd	P4: t&s reg, loc	Sc	Sc	Sc	Sm	P4 acquires lock
7: BusUpd	P4: st loc, #0	Sc	Sc	Sc	Sm	P4 releases lock

The assumption here is that test&set generates a bus update transaction when it succeeds but does nothing when it fails.