Parallel Architectures
Exercise 3

s0840449

1 Simulation Structure & Operation

As the structure of the simulator is largely unchanged from the previous coursework, this section will concentrate on the measuring of latency. The simulator supports the MSI coherence protocol for a *write-through* cache, and can use either a sequential or total store order model of consistency. The choice of consistency model has an impact on how latency is measured, described below.

Sequential Consistency

Measuring latency for a sequential consistency (SC) model is straightforward. For read accesses, the latency is determined by where the cache line that is being read is found. First, the L1 cache is checked (cost: 2 cycles). If the cache does not contain the appropriate line (a cache miss), the bus is informed of the read miss (cost: 20 cycles). At this point, if another cache has the requested line, the remote cache returns the line and the read finishes. Otherwise, the line is fetched from memory (cost: 200 cycles). Therefore, a read access can take 2, 22, or 222 cycles. Write accesses are even simpler. As the cache is *write-through*, all writes must access main memory (to perform the write-through). Therefore, the latency cost for any write is 222 cycles (cache access + bus transition + main memory access).

Total Store Order Consistency

Measuring latency for a total store order (TSO) consistency model is more complex. The existence of the write buffer means that writes may or may not be part of the *critical path* of the program, depending on whether they are processed from the buffer (where the latency is masked by read accesses), or if the write buffer is being drained (where latency is not masked.)

Read accesses in TSO are similar to SC, with two important differences. Before any access is made to the L1 cache, a read first *snoops* the write buffer to see if the address that is being read is also present in the write buffer (cost: 1 cycle). This means that a read can take 1, 3, 23, or 223

cycles. The second difference is that after each read completes, the write buffer must be checked to determine if a processing write has completed during the read. If a write has finished, another may also be started if there are at least N writes in the buffer (for a retire-at-N policy). Neither the finishing of a write or the processing of another incur any latency here. The state diagrams for a read access and for checking the write buffer are shown in figure 1.

Write accesses in TSO are completely unlike SC, due to the existence of the write buffer. When a write is issued, a check is first made to determine if the write buffer is full. If the buffer is full, a write buffer *drain* is initiated. Draining the write buffer is a critical-path operation, so effects the program latency. First, any currently-processing write must be finished (i.e. the current latency must be brought up to the latency that the write would end at). Then, each write in the write buffer is processed and its latency added to the overall latency. Once the drain is over, or if it is not necessary, the newly issued write is added to the end of the write buffer. The write buffer is then checked to determine if a write should be retired from the buffer - i.e. if there are at least N writes for a retire-at-N policy, and no write is currently processing. The state diagrams for a write access and for draining the write buffer are shown in figure 2.

A final difference in TSO is that the write buffer must be drained at the end of the program, to finish processing any final writes.

Testing

The tracking of latency was tested via a set of unit-tests, as well as the required test traces. Testing aimed to cover the individual components in both SC and TSO (read/write latencies, checking the write buffer, draining the write buffer, etc), as well as larger full-trace integration testing.

2 Experimentation

As defined in the coursework, experiments were run on traces *trace1.out* and *trace2.out*, using a cache size of 128 slots and a line size of 4 words. The default setting for the write buffer was a 32-write buffer with a retire-at-1 policy. For each trace the effects of (independently) altering both the size of the buffer ($size = 2^n,\ 2 \leq n \leq 6$) and altering the value of N for retire-at-N ($N = 2^n,\ 0 \leq n \leq 4$) were evaluated. In each evaluation, the reported latency is from the cache with the highest latency. In order to aid

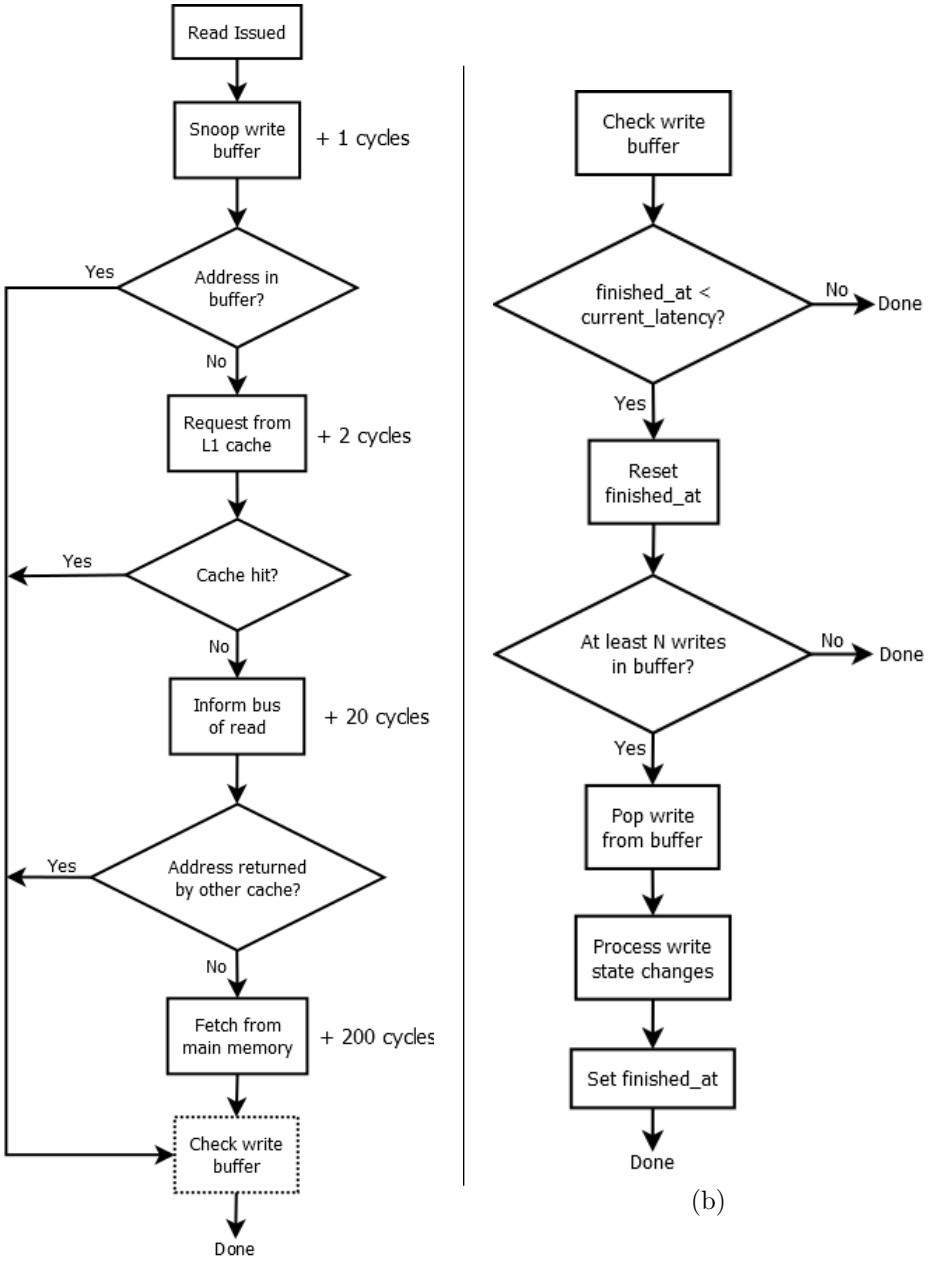


Figure 1: State transition diagrams for (a) issuing a read and (b) checking the write buffer. Dashed boxes indicate sub-diagrams.

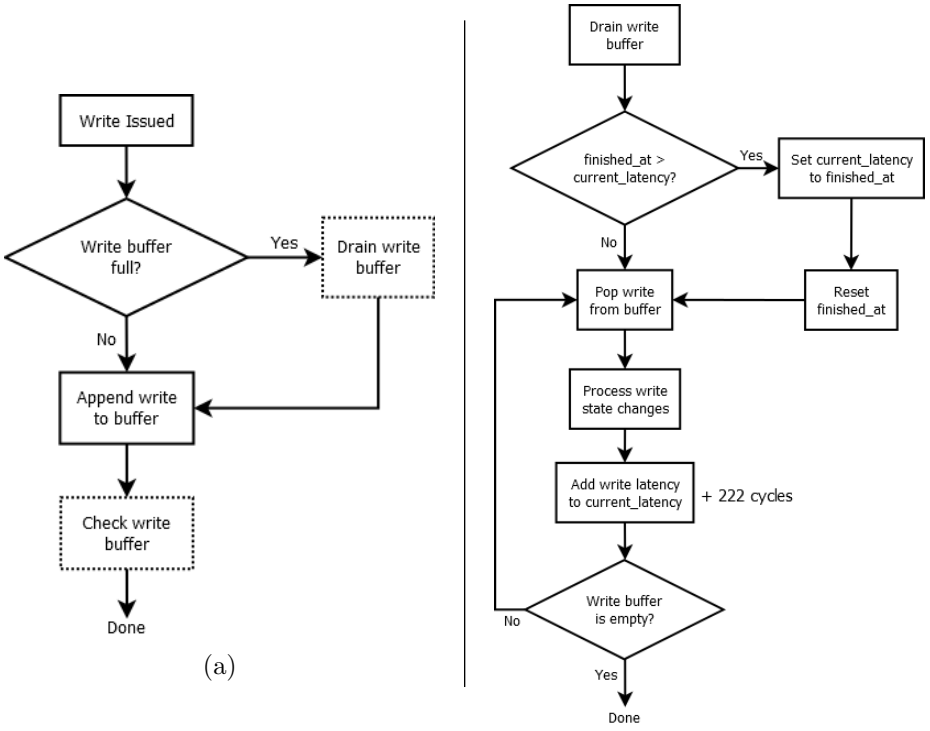


Figure 2: State transition diagrams for (a) issuing a write and (b) draining the write buffer. Dashed boxes indicate sub-diagrams.

understanding of the results, the number of write-buffer drains, the average number of writes drained from the buffer, and the mean and median latency between writes were also tracked and reported.

2.1 Write-Buffer Size Evaluation

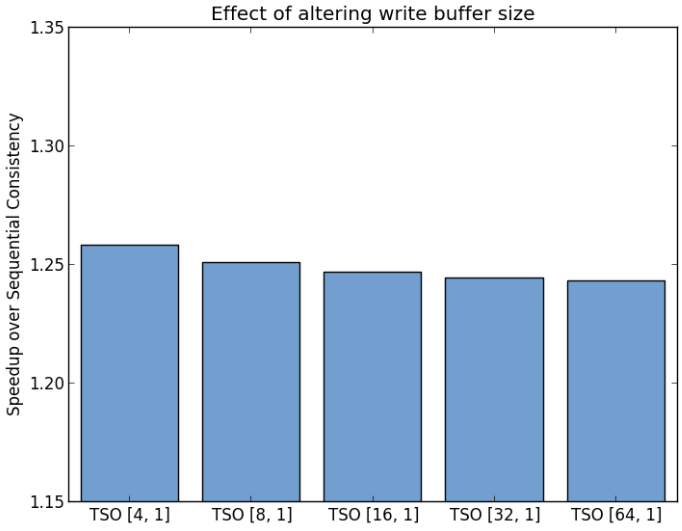
The write buffer size experiment described above was carried out on both *trace1.out* and *trace2.out*. The results can be seen in figure 3.

Starting with *trace1.out*, it is clear that TSO outperforms SC for every combination evaluated, obtaining speedups between 1.24 and 1.26. This is expected; as TSO buffers at least some writes regardless of its parameters, it is impossible for it to have a higher latency than SC. The performance of TSO appears to fluctuate slightly with the size of the write buffer. The smallest write buffer evaluated (size 4) has a speedup of approximately 1.01 over the largest buffer (size 64).

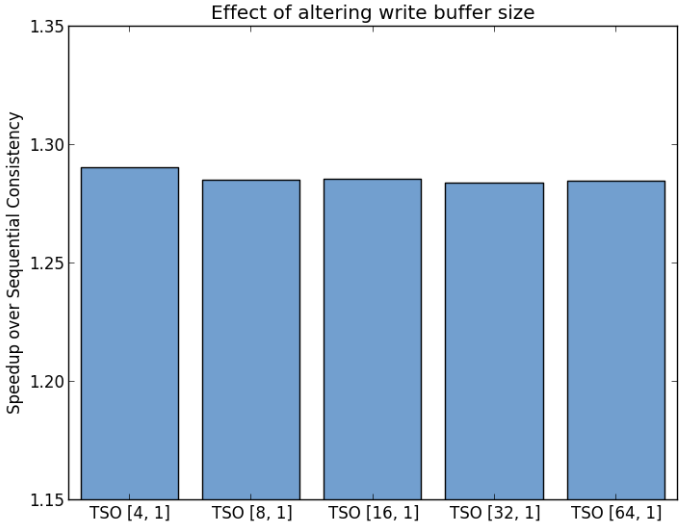
The degradation in performance as the write buffer size increases can be explained by examining the number of write buffer drains and how many writes are processed in each drain. Write buffer drains are singled out as they are the sole reason that writes can affect the program latency. Although using a larger sized write buffers does result in fewer drains, the overall number of writes drained increases from $1201 * 4 = 4804$ with a size-4 buffer to $93 * 63.73 = 5926.89$ with a size-64 buffer. The question now becomes: why the increase in writes drained?

The most likely cause is that writes are being issued too fast during execution (i.e. with too few reads between them) and thus filling up the write buffer and forcing drains. Even in the best case over half of all writes (4804 of 8192) are issued during write-buffer drains. Another signal is the number of cycles that occur between writes being issued. At first this looks quite high, with a mean of around 230 cycles for all tested write buffer sizes. However, examining the median number of cycles between writes shows that the mean is being influenced by outliers. The median is 13, meaning that usually only 13 cycles occur between writes being issued. This means that any write buffer, regardless of size, will quickly fill up.

So given this rapid rate of accumulating writes, why do smaller buffers perform better? This is likely due to the smaller size forcing more frequent drains. When a drain occurs, the new write will then be processed immediately (as the cache is retire-at-1). This then hides a handful of cycles (i.e. approximately 13 * 4 = 52 for the size-4 buffer), until the write buffer fills up again and forces a drain. Larger buffers have more time to finish the ongoing write, but then suffer a much larger drain penalty. Some evidence



(a)



(b)

Figure 3: The effects of altering the write buffer size for (a) *trace1.out* and (b) *trace2.out*. Results are given as speedups over the latency caused by a sequential consistency model.

for this theory can be seen by increasing the cost of a read (and thus the latency between writes). Setting the L1 cache access cost to 20 cycles for reads results in a speedup increase as the write-buffer size gets *bigger*, from 1.47 for a size-4 buffer to 1.51 for a size-64 buffer.

The results for *trace2.out* are similar to those for *trace1.out*. Unlike *trace1.out* the speedup does not decrease monotonically as the size increases: instead it contains two minima points at size-8 and size-32 buffers. This is likely due to a more complex interaction between the writes and reads, such that the different size buffers toe the line between allowing more writes to finish and accepting a more damaging penalty when buffers are flushed. Increasing the cost of reads results in the expected behaviour: a monotonic increase in speedup from 1.54 for a size-4 buffer to 1.65 for a size-64.

The full set of results for this evaluation (for both *trace1.out* and *trace2.out*) can be found in table 1.

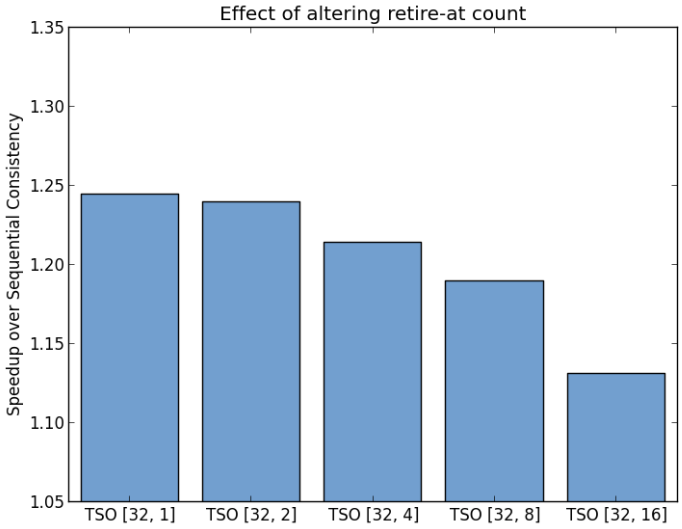
2.2 Retire-At-N Evaluations

The evaluation of the effect of altering the value of N for the retire-at-N policy was carried out on both *trace1.out* and *trace2.out*. The results can be seen in figure 4.

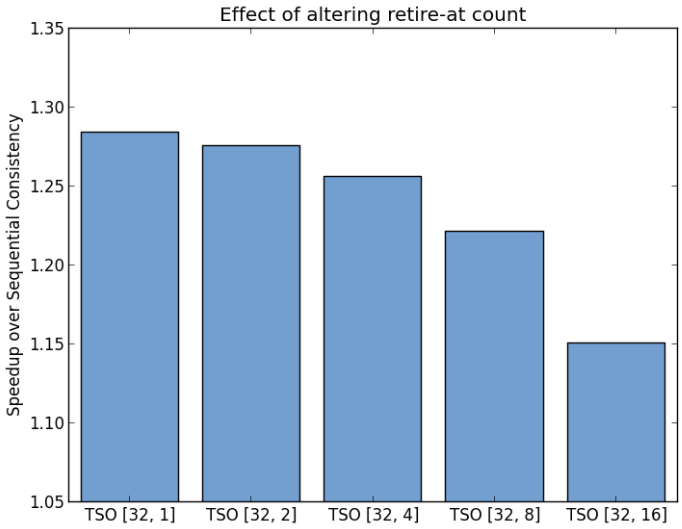
For both *trace1.out* and *trace2.out*, altering the retire-at policy gave a larger difference in performance than altering the write buffer size. For *trace1.out* the performance decreases from a 1.24 speedup with a retire-at-1 policy to a 1.13 speedup with a retire-at-16 policy. For *trace2.out* retire-at-1 gave a 1.28 speedup, decreasing to 1.15 for retire-at-16.

The decrease in performance can likely be attributed to the same problem identified above: writes being issued too rapidly. In the situation where the write buffer is being frequently drained, there is little point to delaying the reitiring of writes - this only lowers the number of cycles hidden. Indeed, the only benefit of retiring writes later is the increased chance for reads to be able to snoop the write buffer. The number of (successful) write-buffer snoops in *trace1.out* stays mostly constant regardless of the retire-at policy: 7884 for retire-at-1, and 8128 for all others. In *trace2.out* there are almost no successful write-buffer snoops, regardless of when writes are retired. Therefore in either case little gain can be had from retiring later, and any improvement is overshadowed by the increased number of drains.

In order for a later retirement policy to be performant, the difference in cost for a write-buffer snoop and a full read must be large. To show this, the cost of a read accessing the L1 cache was set at 100 cycles. For *trace1.out*, a retire-at-1 policy now achieved a 1.38 speedup, while a retire-at-16 policy



(a)



(b)

Figure 4: The effects of altering the 'retire-at' count for (a) *trace1.out* and (b) *trace2.out*. Results are given as speedups over the latency caused by a sequential consistency model.

Setup	Latency	# drains	Mean drain size	Cycles between writes (mean, median)
trace1.out				
SC	2,386,744	N/A	N/A	N/A
TSO [4,1]	1,897,493	1201	4.00	232, 13
TSO [8,1]	1,908,324	667	7.99	233, 13
TSO [16,1]	1,914,642	335	15.96	234, 13
TSO [32,1]	1,918,151	183	31.89	234, 13
TSO [64,1]	1,920,567	93	63.37	233, 13
trace2.out				
SC	6,561,390	N/A	N/A	N/A
TSO [4,1]	5,086,901	3015	4.00	243, 38
TSO [8,1]	5,107,656	1622	8.00	244, 32
TSO [16,1]	5,105,123	838	16.00	243, 29
TSO [32,1]	5,111,229	428	31.97	244, 27
TSO [64,1]	5,109,302	217	63.71	244, 26

Table 1: The results of the write-buffer size evaluation. For the Setup entries, *TSO [X, Y]* indicates a TSO consistency model with a write-buffer of size X and a retire-at-Y policy.

achieved a 1.68 speedup. The effects were less noticeable for *trace2.out*, as it still snooped the write buffer relatively rarely, and so achieved only an increase from a 1.25 speedup for retire-at-1 to 1.26 for retire-at-16.

The full set of results for this evaluation (for both *trace1.out* and *trace2.out*) can be found in table 2.

Setup	Latency	# drains	Mean drain size	Write buffer snoops
trace1.out				
SC	2,386,744	N/A	N/A	N/A
TSO [32,1]	1,918,151	183	31.89	7884
TSO [32,2]	1,926,025	184	31.90	8128
TSO [32,4]	1,966,342	190	31.84	8128
TSO [32,8]	2,006,413	194	31.94	8128
TSO [32,16]	2,110,379	208	31.98	8128
trace2.out				
SC	6,561,390	N/A	N/A	N/A
TSO [32,1]	5,111,229	428	31.97	1
TSO [32,2]	5,145,768	433	31.95	3
TSO [32,4]	5,224,987	441	31.99	2
TSO [32,8]	5,372,941	464	31.95	3
TSO [32,16]	5,704,981	509	31.98	4

Table 2: The results of the retire-at-N evaluation. For the Setup entries, *TSO [X, Y]* indicates a TSO consistency model with a write-buffer of size X and a retire-at-Y policy.