Parallel Architectures
Exercise 2

s0840449

1 Simulation Structure & Operation

An overview of my simulator is shown in Figure 1. The simulator consists of four reusable modules: *Cache*, which models an individual processor cache; *Bus*, which models the system bus that connects the processors; *Statistics Tracker*, which calculates statistics for the simulation; and *Simulation Environment*, which controls the overall simulation. The simulator fully implements the MSI protocol for cache coherency.

The high level operation of the simulation proceeds as follows:

1. The *Simulation Environment* parses a line of the trace file, extracting the issuing processor, instruction type (read or write), and address. It then informs the relevant *Cache* of the instruction.

2. The cache splits the address into the tag, line id, and offset, and looks up the appropriate cache line using the id. Based on the current tag and state of the cache line, the cache determines if the access was a hit or miss.

3. If the instruction caused a cache miss, the cache informs the *Bus* of the miss. The bus then passes this information onto each other cache. This is semantically equivalent to the real-system behaviour of caches snooping the bus.

4. When a cache is informed of a remote read or write miss to an address, it splits the address and looks up its own cache line. If the cache line has a matching tag (i.e. contains the same memory line), its state is updated according to the MSI protocol for remote cache misses.

5. Once all remote caches have been informed (or immediately in the case of an instruction hit), the original issuing cache updates its own cache line state and tag. Assuming the parsed line was not the last in the file, control then returns to step 1.

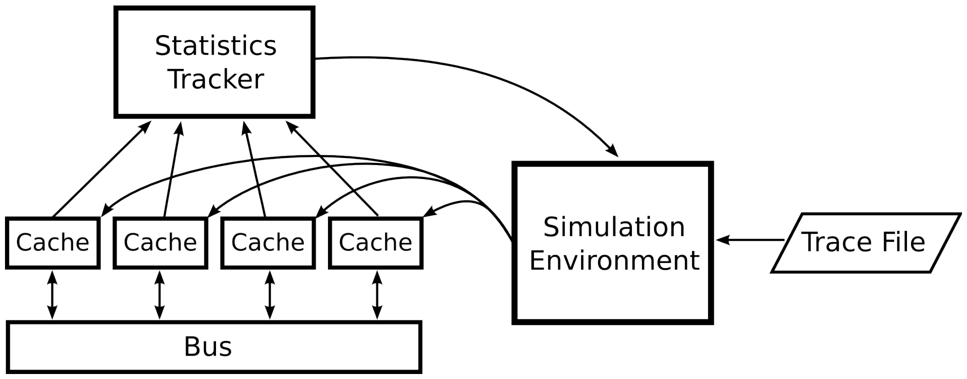


Figure 1: An overview of the simulator. Arrow indicate directional communication. Note that while only 4 caches are shown, the simulator is able to support an arbitrary number of caches.

To ensure simulator correctness, both unit tests and test traces have been defined. These include a verification of all local and remote state transitions in the MSI protocol, along with tests for the statistics gathering. All test traces are run with the simulator in *debug mode*. While running in this mode the simulator performs a full-system consistency check after every line of the trace file is parsed. This check, which has also been run on *trace1.out* and *trace2.out*, ensures that no memory lines are ever in inconsistent states across multiple processors during simulation (e.g. a memory line in `MODIFIED` state in more than one cache.)

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. Due to the lack of clarity in the coursework document regarding whether line status (private, shared read-only, shared read-write) are computed dynamically as the trace is running or statically after the simulation has completed, both dynamic and static cache line statistics were tracked. I currently believe the dynamic case to be the one the coursework requires, so only the dynamic cache line results are shown in the presented graphs (figures 2 and 3). All statistic values are present in the tables in appendix A.

Dynamic Cache Line Statistics

For the dynamic cache line statistics, the three stages - private, shared read-only, and shared read-write - were determined as follows:

- Private: When an access was made to a cache line, if no other processor had the line in their cache (i.e. the slot was `INVALID` or the tag did not match), the access was counted as a private line access.
- Shared Read-Only: When an access was made to a cache line, if some other processor had the line in their cache, but no processor had written to it, it was counted as a shared read-only line access. The written-to status for a cache line was set by a cache on any write to the cache line, and cleared when the cache line was either invalidated by a remote cache, or when a read for the same cache line with a different tag was issued (i.e. flushing the cache line).
- Shared Read-Write: When an access was made to a cache line, if some other processor had the line in their cache and some processor had it marked as written-to, it was counted as a read-write line. While the coursework specified that these were lines that had been read *or* written to, that information can be found by summing read-only and read-write and so I preferred to report the more detailed statistic.

In each case, statistics were determined before the system status was changed; for example, if a processor issued a write to a line that was currently shared read-only, the access was counted as a shared read-only access. (Any access after that, until the line was flushed, would of course be read-write.)

Static Cache Line Statistics

For the static cache line statistics, the three stages - private, shared read-only, and shared read-write - were determined as follows:

- Private: A line was considered private if it was only ever accessed by a single processor during the entire trace. Every access to such lines (for each cache) were then counted as private line accesses.
- Shared Read-Only: A line was considered shared read-only if it was not private and was only every read from during the entire trace. Every

read (access) to such lines (for each cache) were then counted as shared read-only line accesses.

- Shared Read-Write: A line was considered shared read-write if it was not private and not read-only. Every access to such lines (for each cache) were then counted as shared read-write line accesses.

Other Statistics

The tracking of other statistics was fairly straight-forward: hits and misses were tracked dynamically as the trace is simulated, then summed at the end to determine overall miss rates. Coherence misses were defined as the case where an access occurs to a cache line that is `INVALID` but has the same tag (as this means that a remote cache issued a write that caused the local cache to miss), or where a write occurs to a cache line that was `MODIFIED` until a remote cache read from it (detected by keeping track of the previous state: only remote read misses can move a line from `MODIFIED` to `SHARED`.) Finally, address statistics were tracked similarly to miss rates: accessed addresses for each cahe were tracked dynamically then summed at the end of the simulation (that is, the number of addresses was counted, not *accesses* to the addresses.)

3 Results and Discussion

Graphs of the statistics for *trace1.out* and *trace2.out* can be found in figures 2, and 3. Full tables of results can also be found in appendix A.

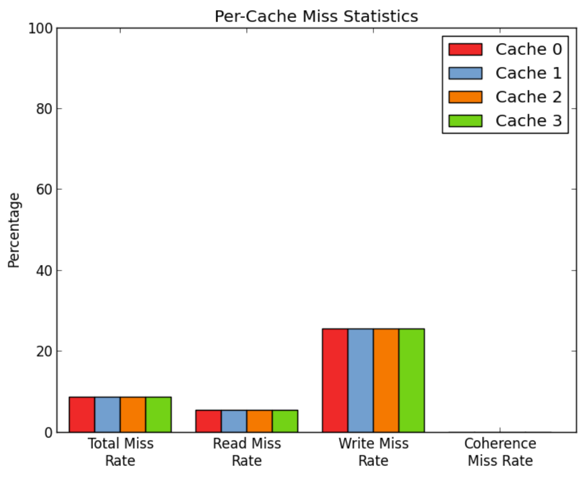
The program represented by *trace1.out* appears to consist of four processes operating on mostly distinct data. The identical miss rates for each of the four processors imply that they are each carrying out the same code, while the high levels of dynamic private line access (99.80% on average) and addresses accessed by only one processor (95.45%) show a lack of communication between processors. The static private line accesses show that some cross-line accesses occur (with a 4.69% access rate to shared read-write lines), but these happen at different times and do not cause coherence problems. I would expect that this program is carrying out SIMD-style processing, where the same code is run over distinct data sets (or parts of a data set). Interestingly the data accessed in *trace1.out* appears to be quite local; increasing the size of the cache lines to 32 words reduces the miss rate to near zero, implying that the accessed region for each processor is an area of around 1kb (assuming a 4-byte word size).

In contrast, the program represented by *trace2.out* appears to be communication heavy. On average only 15.15% of accesses are to private lines, and 27-29% of misses are caused by coherence issues. Combined with the high write miss (69-76%), this implies a program that reads from large amounts of shared memory, and writes back to a significant portion of it. This is backed by the shared read-only and shared read-write access rates: 52.93% and 31.92% on average respectively.

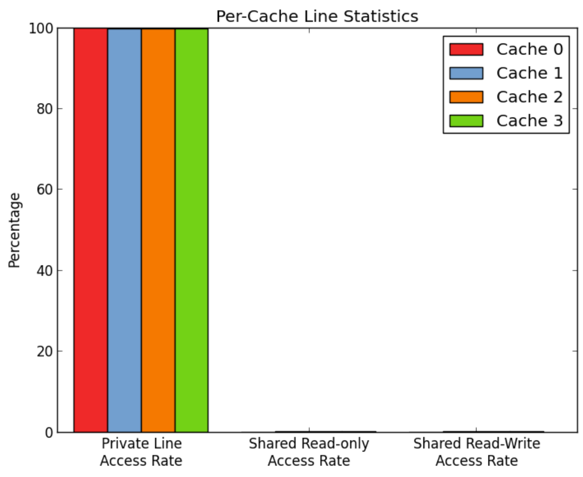
However, it is possible that the observed coherence in *trace2.out* is not real; the processors may simply be *false sharing*. False sharing occurs when processors access addresses that are distinct but on the same memory line[1]. The presence of false sharing causes coherence misses and a high ratio of shared line use even in a program that is not actually communication-heavy. In the case of *trace2.out*, a large percentage of addresses (81.32%) are still accessed by only one processor. This could be an indicator of false sharing. To detect whether the observed coherence is real or not, the line size should be lowered to a single word and the experiment re-run. This prohibits false sharing (although it may increase non-coherency misses significantly), as each address has its own line. Doing so for *trace2.out* results in a reduce coherence miss rate of 6.05% and an increase in the average private line access rate to 27.39% . This implies that there is some false sharing in *trace2.out*, but also a large chunk of true sharing that causes coherency problems.

As a final observation, it is not possible in *trace2.out* to estimate how close together the accessed memory is (from the tracked statistics). Due to the large amount of shared memory, increasing the line size actually increases the miss rate, as processors increasingly write to shared lines and cause coherence misses. Using the same 32-word line size as was tried for *trace1.out* increases the coherence miss rate to 84-89% (with the overall miss rate holding steady at around 20% - this value increases as the line size becomes increasingly bigger).

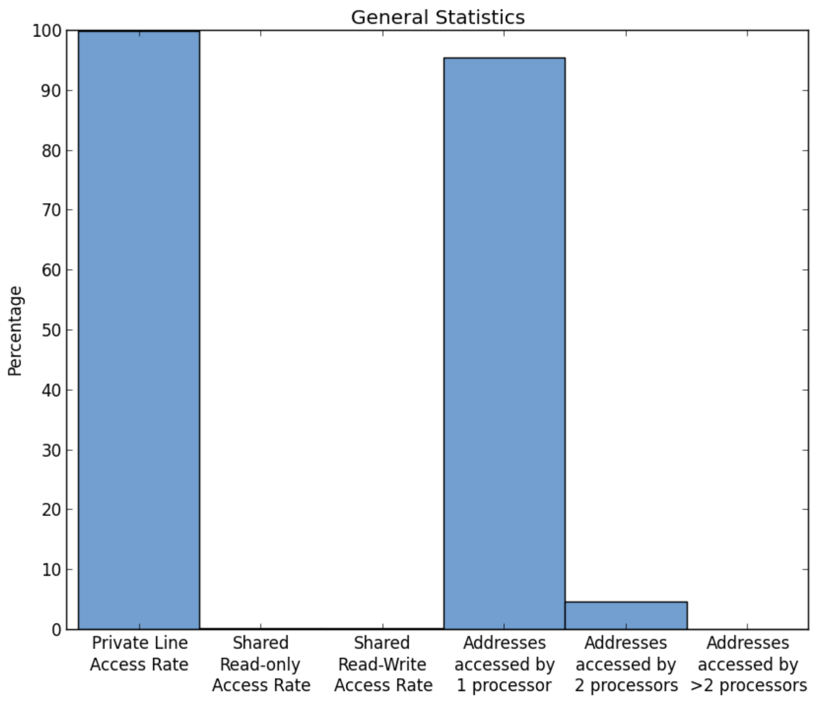
[1]False sharing can be observed in *trace1.out* by increasing the size of the lines to a higher number such as 64 words per line. This then causes coherence misses that did not originally occur.



(a)

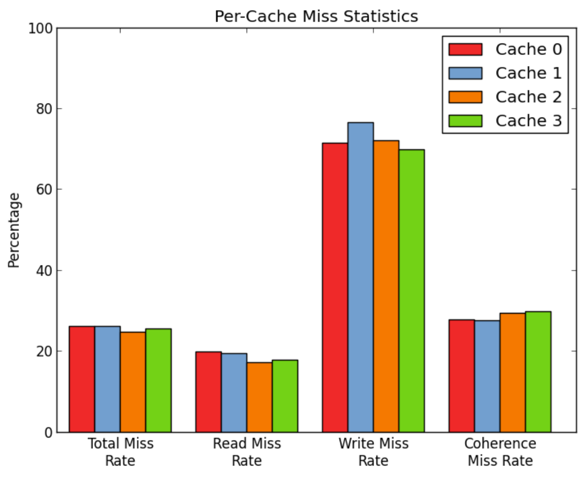


(b)

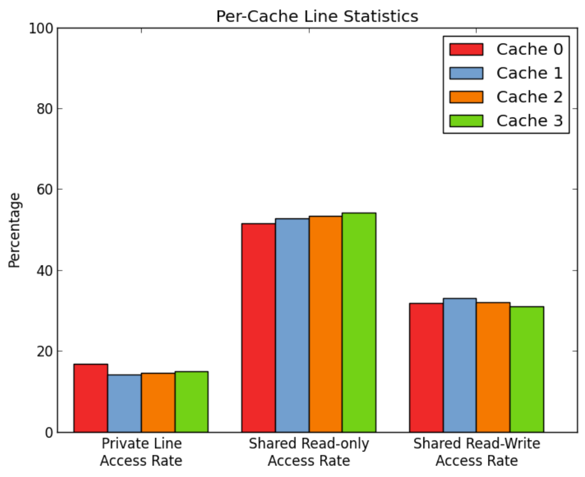


(c)

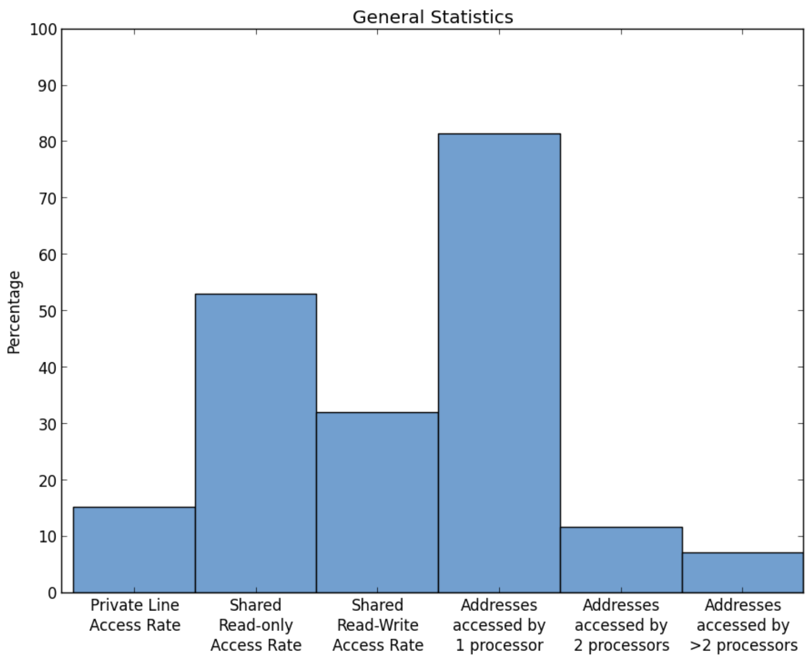
Figure 2: Statistics for *trace1.out*. These include: (a) per-cache miss rate statistics, (b) per-cache line access statistics, and (c) general statistics for the whole trace.



(a)



(b)



(c)

Figure 3: Statistics for *trace2.out*. These include: (a) per-cache miss rate statistics, (b) per-cache line access statistics, and (c) general statistics for the whole trace.

A Full Result Tables

Statistic	**Value**
Cache 0	
Miss Rate	8.73% (4292 of 49152)
Read Miss Rate	5.40% (2210 of 40960)
Write Miss Rate	25.42% (2082 of 8192)
Coherence Miss Rate	0.00% (0 of 4292)
Dynamic Private Line Access Rate	100.00% (49152 of 49152)
Dynamic Shared Read-Only Line Access Rate	0.00% (0 of 49152)
Dynamic Shared Read-Write Line Access Rate	0.00% (0 of 49152)
Static Private Line Access Rate	96.88% (47616 of 49152)
Static Shared Read-Only Line Access Rate	0.00% (0 of 49152)
Static Shared Read-Write Line Access Rate	3.12% (1536 of 49152)
Cache 1	
Miss Rate	8.73% (4292 of 49152)
Read Miss Rate	5.40% (2210 of 40960)
Write Miss Rate	25.42% (2082 of 8192)
Coherence Miss Rate	0.00% (0 of 4292)
Dynamic Private Line Access Rate	99.74% (49022 of 49152)
Dynamic Shared Read-Only Line Access Rate	0.10% (48 of 49152)
Dynamic Shared Read-Write Line Access Rate	0.17% (82 of 49152)
Static Private Line Access Rate	93.75% (46080 of 49152)
Static Shared Read-Only Line Access Rate	0.00% (0 of 49152)
Static Shared Read-Write Line Access Rate	6.25% (3072 of 49152)
Cache 2	
Miss Rate	8.73% (4292 of 49152)
Read Miss Rate	5.40% (2210 of 40960)
Write Miss Rate	25.42% (2082 of 8192)
Coherence Miss Rate	0.00% (0 of 4292)
Dynamic Private Line Access Rate	99.74% (49022 of 49152)
Dynamic Shared Read-Only Line Access Rate	0.10% (48 of 49152)
Dynamic Shared Read-Write Line Access Rate	0.17% (82 of 49152)
Static Private Line Access Rate	96.88% (47616 of 49152)
Static Shared Read-Only Line Access Rate	0.00% (0 of 49152)
Static Shared Read-Write Line Access Rate	3.12% (1536 of 49152)

Table 1: The first half of the tracked statistics for *trace1.out*.

Statistic	**Value**
Cache 2	
Miss Rate	8.73% (4292 of 49152)
Read Miss Rate	5.40% (2210 of 40960)
Write Miss Rate	25.42% (2082 of 8192)
Coherence Miss Rate	0.00% (0 of 4292)
Dynamic Private Line Access Rate	99.74% (49022 of 49152)
Dynamic Shared Read-Only Line Access Rate	0.10% (48 of 49152)
Dynamic Shared Read-Write Line Access Rate	0.17% (82 of 49152)
Static Private Line Access Rate	93.75% (46080 of 49152)
Static Shared Read-Only Line Access Rate	0.00% (0 of 49152)
Static Shared Read-Write Line Access Rate	6.25% (3072 of 49152)
General Statistics	
Dynamic Private Line Access Rate	99.80% (196218 of 196608)
Dynamic Shared Read-Only Line Access Rate	0.07% (144 of 196608)
Dynamic Shared Read-Write Line Access Rate	0.13% (246 of 196608)
Static Private Line Access Rate	95.31% (187392 of 196608)
Static Shared Read-Only Line Access Rate	0.00% (0 of 196608)
Static Shared Read-Write Line Access Rate	4.69% (9216 of 196608)
Addresses Accessed By One Processor	95.45% (16128 of 16896)
Addresses Accessed By Two Processors	4.55% (768 of 16896)
Addresses Accessed By More Than Two Processors	0.00% (0 of 16896)

Table 2: The second half of the tracked statistics for *trace1.out*.

Statistic	**Value**
Cache 0	
Miss Rate	26.09% (39869 of 152805)
Read Miss Rate	19.86% (26667 of 134303)
Write Miss Rate	71.35% (13202 of 18502)
Coherence Miss Rate	27.73% (11055 of 39869)
Dynamic Private Line Access Rate	16.80% (25668 of 152805)
Dynamic Shared Read-Only Line Access Rate	51.45% (78618 of 152805)
Dynamic Shared Read-Write Line Access Rate	31.75% (48519 of 152805)
Static Private Line Access Rate	0.00% (0 of 152805)
Static Shared Read-Only Line Access Rate	61.84% (94500 of 152805)
Static Shared Read-Write Line Access Rate	38.16% (58305 of 152805)
Cache 1	
Miss Rate	26.13% (36885 of 141135)
Read Miss Rate	19.50% (24310 of 124681)
Write Miss Rate	76.43% (12575 of 16454)
Coherence Miss Rate	27.57% (10169 of 36885)
Dynamic Private Line Access Rate	14.22% (20066 of 141135)
Dynamic Shared Read-Only Line Access Rate	52.79% (74505 of 141135)
Dynamic Shared Read-Write Line Access Rate	32.99% (46564 of 141135)
Static Private Line Access Rate	0.00% (0 of 141135)
Static Shared Read-Only Line Access Rate	69.08% (97500 of 141135)
Static Shared Read-Write Line Access Rate	30.92% (43635 of 141135)
Cache 2	
Miss Rate	24.73% (37355 of 151060)
Read Miss Rate	17.12% (22278 of 130136)
Write Miss Rate	72.06% (15077 of 20924)
Coherence Miss Rate	29.34% (10959 of 37355)
Dynamic Private Line Access Rate	14.60% (22060 of 151060)
Dynamic Shared Read-Only Line Access Rate	54.35% (80742 of 151060)
Dynamic Shared Read-Write Line Access Rate	31.95% (48258 of 151060)
Static Private Line Access Rate	0.00% (0 of 151060)
Static Shared Read-Only Line Access Rate	67.66% (102200 of 151060)
Static Shared Read-Write Line Access Rate	32.34% (48860 of 151060)

Table 3: The first half of the tracked statistics for *trace2.out*.

Statistic	**Value**
Cache 2	
Miss Rate	25.54% (35393 of 138555)
Read Miss Rate	17.83% (21030 of 117973)
Write Miss Rate	69.78% (14363 of 20582)
Coherence Miss Rate	29.77% (10537 of 35393)
Dynamic Private Line Access Rate	14.87% (20600 of 138555)
Dynamic Shared Read-Only Line Access Rate	54.15% (75034 of 138555)
Dynamic Shared Read-Write Line Access Rate	30.98% (42921 of 138555)
Static Private Line Access Rate	0.01% (20 of 138555)
Static Shared Read-Only Line Access Rate	70.37% (97500 of 138555)
Static Shared Read-Write Line Access Rate	29.62% (41035 of 138555)
General Statistics	
Dynamic Private Line Access Rate	15.15% (88394 of 583555)
Dynamic Shared Read-Only Line Access Rate	52.93% (308899 of 583555)
Dynamic Shared Read-Write Line Access Rate	31.92% (186262 of 583555)
Static Private Line Access Rate	0.00% (20 of 583555)
Static Shared Read-Only Line Access Rate	67.12% (391700 of 583555)
Static Shared Read-Write Line Access Rate	32.87% (191835 of 583555)
Addresses Accessed By One Processor	81.32% (1641 of 2018)
Addresses Accessed By Two Processors	11.60% (234 of 2018)
Addresses Accessed By More Than Two Processors	7.09% (143 of 2018)

Table 4: The second half of the tracked statistics for *trace2.out*.