Assignment 2: Cache Coherence Protocols

CS4/MSc Parallel Architectures

Issued:	Tuesday, February 13, 2018
Due:	Tuesday, March 6, 2018 at 4:00 PM

1. Introduction

This assignment is the second practical of the Parallel Architectures module for CS4/MSc students. This practical will contribute 20% of the final mark for this course. It consists of a programming exercise culminating in a brief written report. Assessment of this practical will be based on the correctness and the clarity of the solution. This practical is to be solved individually to assess your competence on the subject. Please bear in mind the School of Informatics guidelines on plagiarism. Please refer to Section 4 for submission instructions.

2. Overview of the Practical

The goal of this practical is to develop a cache coherence protocol simulator and evaluate it on 2 memory access traces. The trace files will be given to you and their format is described in §3.2. To accomplish this goal, you will need to undertake the following subtasks, which will be marked separately; however, bear in mind, that 1-3 can only be awarded fully if these subtasks are documented sufficiently in the report.

1. **Implementation of the simulator *(40 marks).*** In order to simulate coherent caches, first you will have to write a simple cache simulator. For simplicity, assume a direct mapped cache with write back policy for your experiments. The size of the lines and the number of lines in the cache, however, should be parameterised and your final experiments should be performed for different values of such parameters. Please refer to Hennessy and Patterson [1] or the CS3 Computer Architecture notes for details on how addresses (word addresses in the case of this practical) are broken into index and tag. Note that the cache simulator code should be modular as you will need multiple instances of it, one for each processor in the system.

   On top of this very simple cache simulator you will have to build a simple coherence protocol simulator. For simplicity, you can assume a system with only 4 processors and with a single level of cache per processor (all given trace files will have only 4

processors). You will have to implement invalidation based MSI *(10 marks)* and MESI *(10 marks)* snooping coherence protocols, and update based MES *(20 marks)* snooping coherence protocol described in Appendix A. Again, for simplicity you can assume that all transactions on the bus are instantaneous and you *do not* have to model transient states nor split transactions.

Additionally, you will have to augment your simulator with mechanisms to collect key statistics, such as miss rate[1], number of invalidations[2], number of updates[3], and anything else that will help you perform a quantitative analysis of the results you obtain from your experiments (see below).

2. **Validation of the simulator *(10 marks).*** In order to have high confidence in the validity of your results, you should provide a clearly written description of the structure and operation of your simulator and a set of small, artificial examples which use the various output options to demonstrate that your simulator is working correctly. The examples you choose are entirely up to you, but you should make sure you cover important cases, such as reads and writes to a cache-line/block that is in states Modified or Shared in another cache.

3. **Design and execution of experiments *(30 marks).*** The aim of the experiments with the given traces is to investigate the behaviour of both the caches (miss rate, etc.) and programs. Answer these questions:

   a) Compare the behaviour of the three protocols in terms of miss-rate and messages sent on the bus. Messages can be classified into invalidation messages (present only in invalidation based protocols), and data messages (write-backs[4] and write-updates[5]).

   b) Vary the cache-line/block size to iterate over the above experiments. How do differing cache-line/block sizes affect miss/hit rates and invalidations/updates?

   c) In terms of memory accesses, what is the distribution of accesses to private data (hits in state Modified, and also Exclusive for MESI/MES), shared data (hits in Shared).

4. **Report and discussion *(20 marks).*** You must report and discuss the results of your experiments. The written report should be clear and concise, and you should present your results in the form of plots. You should carefully plot the results to highlight the impact of various parameters. At the end of the report you should provide a summary of the results along with your inferences and conclusions.

[1] Assume that a tag match on a line cached in shared state upon a write also counts as a "miss".

[2] You should separately count both the number of invalidation broadcasts and the number of lines invalidated at each broadcast.

[3] You should separately count both the number of update broadcasts and the number of lines updated at each broadcast. This stat is only relevant for update based MES protocol.

[4] Cache-line write-back message is sent on the bus on observing a Remote Read Miss or Remote Write Miss for a cache-line in M state.

[5] Cache-line write-update message is sent on the bus on a CPU Write (only for update based coherence protocols).

You are strongly advised to work in stages and iteratively. For instance, try to validate each new feature of the simulator and describe it in the report as you develop it. Also, try to perform the main experiments in stages with a small set of parameters at a time and alternating with the writing of the report.

3. Baseline specification

Any assumptions you make must be stated clearly in your report. Below are the given parameters of the system you must implement.

3.1. Architecture

Our imaginary architecture has 4 processors and a very simple memory and bus model. Memory is addressed at a word granularity and data addresses start from 0. There is no virtual memory. Thus, address 0 indexes the first word in memory, address 1 the second word, and so on. The cache-line/block size will correspond to some multiple of 2 number of words, which is a parameter that should be parametrised in your simulator in a suitable manner (e.g., command line argument). The caches are direct mapped with a write back policy[6].

In the default configuration, private cache of each processor has 512 cache-lines and the cache-line/block size is 4 words. In the report, results for all other configurations should be analysed relative to the default configuration. In all, cache-line/block size of 2, 4, 8 and 16 words should be evaluated. While varying the number of words per cache-line/block, the total cache size should be kept constant as shown in Table 1.

Cache-line/Block size	Number of cache-lines
2	1024
4	512
8	256
16	128

Table 1

The main point of this simple model is that your simulator does not have to concern itself with byte addressing within words, word alignment and so on. The bus is non-split-transaction and you can assume that transactions are instantaneous. You can also assume that all state changes in the caches are instantaneous.

3.2. Trace File Format

Your simulator *must* accept its input from trace files which have the format described here. Every line in the trace describes either a memory operation or a command for your simulator to produce some output.

Memory operations: A memory access line in the trace consists of the processor identifier (e.g., `P1`), followed by a space, followed by the type of operation (`R` or `W`, for read or

[6]Note that for S state in MES protocol, the behaviour will be similar to write through cache as described in Appendix A.

write, respectively), followed by another space, followed by the word address, and ending with a newline. Thus for instance, the line:

`P2 R 12`

indicates a read by processor 2 to word 12. Memory operations must appear in the bus in the order specified in the trace. The actual timing of the operations is not relevant and is not taken into account.

Note that neither the actual data value being written nor the register being used are specified. Thus, your simulator cannot keep track of real data. All that matters in calculating hit rate and invalidations is the sequence of addresses used.

Output commands: An output command line in the trace consists of one of the following:

- `v`: indicates that full line-by-line explanation should be toggled, i.e. switched on (if it is currently off) or off (if it is currently on). The default is that it is off.
- `p`: indicates that the complete content of the cache should be output in some suitable format. A sample format is shown below, but you are free to use your own format.

Format	Example (Processors: 2 | Cache Size: 2 lines)

`<Processor>`
`<cache-line/block number> <tag> <state>`
	`P1`
	`0 13 S`
	`1 8 M`
	`P2`
	`0 4 S`
	`1 32 E`

Table 2

- `h`: indicates that the hit-rate achieved so far should be output.
- `i`: indicates the that total number of invalidations seen so far should be output.

You are free to choose the actual wording of the explanations and other output. As a suggestion you can use: "A read by processor P2 to word 17 looked for tag 0 in cache-line/block 2, was found in state Invalid (cache miss) in this cache and found in state Modified in the cache of P1."

For example, the meaning of the following trace file is shown in parenthesis, to the right of the commands (these comments are not actually present in the trace):

```
v          (switch on line by line explanation)
P0 R 17    (a read by processor 0 to word 17)
P1 R 18    (a read by processor 1 to word 18)
P2 W 17    (a write by processor 2 to word 17)
v          (switch off line by line explanation)
P2 R 25    (a read by processor 2 to word 25)
P0 R 35    (a read by processor 0 to word 35)
p          (print out the cache contents)
h          (print out the hit rate)
i          (print out the number of invalidations)
```

4. Format of your submission

Your submission should clearly indicate which parts of the coherence protocol you have simulated completely and which you have only partially completed. Submit an archive of your simulator *and* a soft copy of your report as follows, using the DICE environment:

```
$ submit pa cw2 <cache-simulator.tar.gz> <report.pdf>
```

Ensure that your source code is well documented. Your report should be self-sufficient, but also ensure your simulator source code is well documented. The report should contain the following:

1. Written description of the internal structure and workings of your simulator (*1-2 pages, single spaced,* depending upon complexity), explaining clearly which parts are complete and which incomplete.

2. Extracts of simple trace files used for validation and a description of expected and actual results on these, in order to provide evidence that your simulations are accurate (a few pages including examples).

3. A report detailing the experiments you ran and giving a critical summary of the results (*2-4 pages, single spaced,* including results). You should provide enough information to make the experiments repeatable. Present your data clearly and concisely. Also present any inferences and conclusions you have obtained from the results of these experiments.

5. Reporting Problems

Send an email to arpit.joshi@ed.ac.uk for any issues regarding the assignment.

References

[1] J. L. Hennessy and D. A. Patterson. *Computer Architecture - A Quantitative Approach (5. ed.)*. Morgan Kaufmann, 2011.

Appendices

A. MES Protocol

Figure 1 shows the state transition diagram for MES, which is an update based cache coherence protocol. MES protocol has three states, namely Modified (M), Exclusive (E) and Shared (S). The state transition diagram is color coded. Events in blue colour indicate CPU side events and events in red colour indicate bus side events. The Shared part with CPU Read/Write Miss request is the shared wire on the bus. This wire is pulled high by processors on seeing a Remote Read/Write Miss request for a cache line that they have in their private cache (in any of M, E or S states).

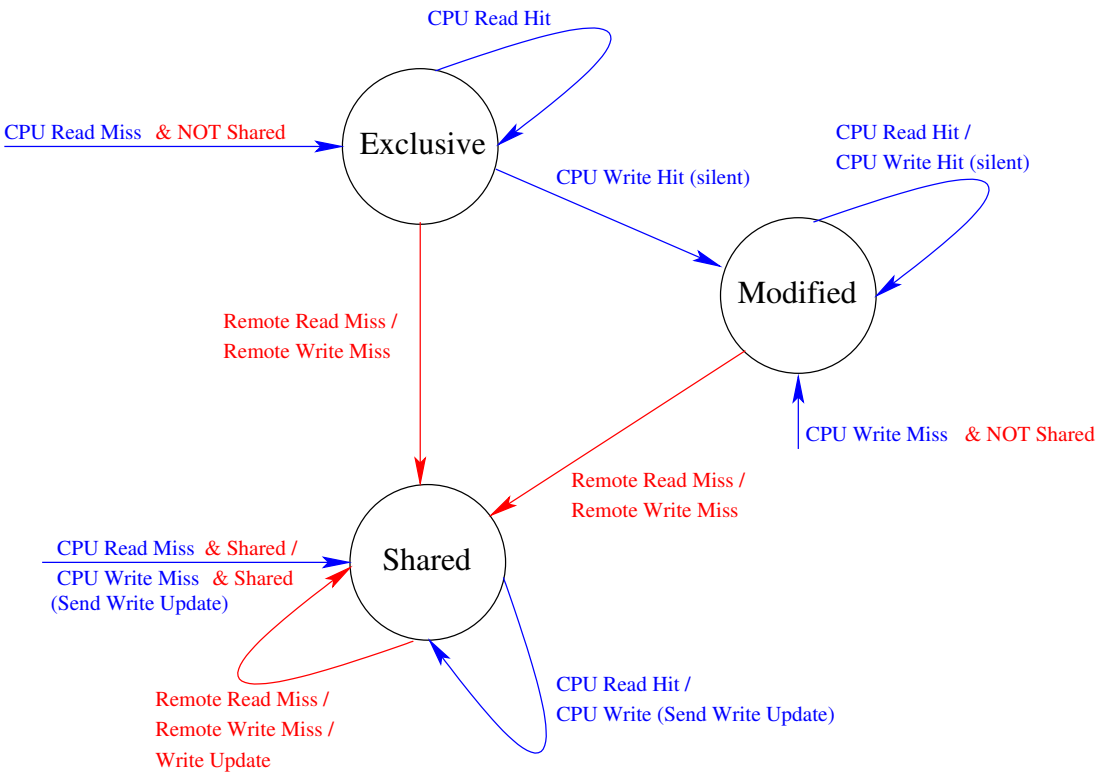

Figure 1: MES State Transition Diagram

In addition to the transitions shown in the diagram, following things need to be taken care of:

- On a cache line replacement for a cache line in M state a write back message is sent on the bus to update memory. That message should be counted as a data message.
- A data write message is also sent on the bus to update memory and requesting processor on a transition from M $\rightarrow$ S.
- If a cache controller observes a Remote Read/Write Miss for a cacheline that it has cached in any of the M, E or S states then it will pull the shared wire on the bus to high.
- Although the cache is a write-back cache, on each write to a cache-line in S state or on a write to a cache line not present in the cache (Write Miss & Shared event), the processor will send a write-update message on the bus so that all the processors (caching that particular cache-line in their private cache) and the memory will be updated with the latest value of the cache-line. Note that on a write to a cache line in E or M state, no update message is sent on the bus.