**ECE382N.10 Parallel Computer Architecture Lab 2**

**Work division**

**Qiyang Ding: Protocol design and test design/implementation**

**Mingyu Lei: Protocol design and implementation**

1. **Part 1: Warm Up**

Since the random here is a uniform distribution and cache size is half the memory size, the hit rate should be 0.5. However, at load operations, the stats consider full hits and partial hits based on whether the copy is exclusive/modified or shared not, even in this case where we have only one node. That is, the hit rate is less than 0.5 because it doesn’t count partial hits as hits. To fix this problem, there are two ways. One is to directly store the data into the cache line once it hits because it is not necessary to consider cache coherence in this case (there is only one node). The other is to increase the cache size to give more hits.

1. **Part 2: RWITM**

The difference between storing to an invalid cache line and a shared cache line is the data requirement. That is, when storing into an invalid cache line, we need to first initiate a read request to get the data, whereas shared state means that we already have the copy. Therefore, if they are treated the same, there will be less logic to determine whether the cache line is shared or invalid, and we don’t need to distinguish between bus ops with or without data. The normal way is to split them into two operations, as seeing a shared cache line saves us one data request to the bus.

1. **Protocol Design and implementation**
   1. **Directory**

There are four states in the directory entry, which are *Invalid*, *Shared*, *Owned*, and *Shared*-*no-data*. *Invalid* means that no one shares or owns this cache line; *Shared* means someone has a copy of this cache line; *Owned* means one node owns this cache line; *Shared-no-data* means this cache line will be written back from the owner, but it is not ready.

There is one directory entry associated with each cache line. Each entry is composed of 2 bits for 4 different states, 5 bits for the owner ID and 32 bits for the sharer list.

* 1. **Network Queues**

Two new network queues (priority levels) - *Writeback* and *Forward* have been added between the existing *Reply* and *Request* queues. The *Writeback* queue is used to optimize the transaction speed of writeback requests, while the *Forward* queue is used to separate normal requests with forward requests to avoid potential deadlock.

The new priority level is *Reply* > *Writeback* > *Forward* > *Request*.

* 1. **Internal Buffers**
     1. **IU to Network**

Two internal queues are introduced for invalidation broadcasting (*to\_net\_inv\_q*) and for buffering network request(s) if it/they can’t be sent out in the current cycle (*net\_buffer*).

The size of *to\_net\_inv\_q* is set to 32 (requests), which allows broadcasts to all sharers to be processed in the same cycle. If there is not enough space in *to\_net\_inv\_q*, the directory will send out no-ack to let the requester retry the operation.

The size of *net\_buffer* is 2 (requests). It allows for temporary storage if the corresponding network queue is full, and it buffers the second request generated within this cycle to meet the request that only one network request can be sent out per cycle. If there are any pending request in *net\_buffer*, all directory operations will be blocked.

There is only one processor command from cache to IU per cycle. If the command triggers replacement, one write-back request will be generated and processed in the same cycle the new cache line comes in.

* + 1. **Cache to IU**

A *proc\_cmd\_buffer* with two entries is introduced due to the interface restrictions between Cache and IU. The first entry functions as the original *proc\_cmd*, whereas the second one temporally holds the data that will be written back (flushed) and sent back to the corresponding memory location (if the memory location doesn’t associate with node). There is only one processor command from cache to IU per cycle. If the load/store command triggers a cache line replacement, a write-back request will be generated, buffered in the second entry of the *proc\_cmd\_buffer*, and processed in the same cycle the new cache line comes in.

* 1. **Directory and Requests**

|  |  |  |  |
| --- | --- | --- | --- |
| Req Type Dir State | Read | RWITM | Write-back |
| Invalid | If there are no sharers,  Dir -> Owned   1. Update sharer list and owner 2. Send reply to the requester with data   Else,  Dir: Invalid  Reply to the requester with non-acknowledge | If there are no sharers,  Dir -> Owned   1. Update sharer list and owner 2. Send reply to the requester with data   Else,  Dir: Invalid   1. Reply to the requester with non-acknowledge | X |
| Shared | Dir: Shared   1. Update sharer list 2. Send reply to the requester with data | Dir -> Owned   1. Send invalidation requests to all the sharers 2. Update sharer list and owner 3. Send reply to the requester with data | Dir: Shared   1. Update sharer list |
| Owned | If sharer list shows that other nodes except for the owner have the copy,  Dir: Owned   1. Reply to the requester with non-acknowledge   Else,  Dir -> Shared-no-data   1. Forward the request to the owner | Dir: Owned   1. If the requester is one of the sharers, reject the request to the requester 2. If the requester is not one of the sharers, forward the request to the owner | Dir -> Invalid   1. Write data into related memory 2. Update sharer list and owner |
| Shared-no-data | Dir: Shared-no-data   1. Forward the request to the owner | Dir: Shared-no-data   1. Reject the request to the requester | Dir -> Invalid   1. Write data into related memory 2. Update sharer list and owner |

* 1. **Directory and Replies**

|  |  |  |  |
| --- | --- | --- | --- |
| Req Type Dir State | Forward Read Reply | Forward RWITM Reply | Invalidation Reply |
| Invalid | X | X | Dir: Invalid   1. Update sharer list (Change 1 to 0) |
| Shared | X | X | Dir: Invalid  Update sharer list (Change 1 to 0) |
| Owned | Dir -> Shared   1. Copy the data into local memory 2. Update sharer list | Dir: Owned   1. Update the owner id in the directory entry | Dir: Invalid  Update sharer list (Change 1 to 0) |
| Shared-no-data | X | X | Dir: Invalid  Update sharer list (Change 1 to 0) |

* 1. **Cache and Requests**

|  |  |  |  |
| --- | --- | --- | --- |
| Req Type Dir State | Forward Read | Forward RWITM | Invalidation |
| Invalid | Cache: Invalid   1. Reply to the requester with non-acknowledgement | Cache: Invalid   1. Reply to the requester with non-acknowledgement | X |
| Shared | Cache: Shared   1. Reply to the requester with data | X | Cache -> Invalid   1. Invalidate related cache line 2. Update replacement status of the set |
| Exclusive | Cache -> Shared   1. Reply to the requester with data 2. Reply to related directory with data | Cache -> Invalid   1. Reply to the requester with data 2. Reply to related directory with data | X |
| Modified | Cache -> Shared   1. Reply to the requester with data 2. Reply to related directory with data | Cache -> Invalid   1. Reply to the requester with data 2. Reply to related directory with data | X |

* 1. **Cache and Replies**

|  |  |  |
| --- | --- | --- |
| Reply Type Dir State | Read Reply | RWITM Reply |
| Invalid | Cache: Based on permit tag (Shared, Modified, Exclusive)   1. Find related set in the cache 2. Fill the cache (May cause replacement) 3. If there is replacement occurred, send the request to the IU for write-back | Cache -> Modified   1. Find related set in the cache 2. Fill the cache (May cause replacement) 3. If there is replacement occurred, send the request to the IU for write-back |
| Shared | X | X |
| Exclusive | X | X |
| Modified | X | X |

1. **Test infrastructure**

Test infrastructure is built based on four vectors. They are test case vector, test record vector, test result vector, and test golden vector. The first one is to control the processor behavior (load/store) based on specific cycles. The second one is to control the processor to record the internal state in directory and cache. This vector will be only used in microarchitecture tests since it needs specific interface of cache and directory. The third and the fourth ones are the internal state/data and expected state/data. It will automatically compare the results with golden and show errors depending on different values. This infrastructure can reduce the test complexity by limiting test into cache coherence protocol (Processor Load/Store, Cache state, Directory state).

1. **Test cases**

There are 33 microarchitecture tests that compare the internal state and its strict cycles. These tests include four tests for invalid directory state (local and global), seven tests for owned directory state (local and global), twelve tests for shared directory state (local and global), five tests for shared-no-data directory state (local and global), three tests for checking invalidation queue, and one random test.

For random test, the checker will not care about its correctness, but it checks the stability of the simulator to avoid any possible assertion errors. There is also a script to translate random tests into a specific access patterns so that debugging problem should be more convenient.

To make the number of bugs converge, all the tests will be used in regression process for each commit. This can greatly help us control the debugging process to make simulator work as expected.

There are 26 architectural-level tests that only use old interface (No probe interface to cache and directory). Most of them are translated from microarchitecture tests to provide a high-level verification of the protocol. Due to time limitations, there are not comments in these tests, but it’s similar to the microarchitecture tests (There are comments for microarchitecture tests).

There are also many assertions in the codes to keep internal state as expected and they are also used in random test to test the stability of the whole system.