CS 2410 Computer Architecture Project 2

Simulation of directory-based cache coherence protocol

DIS43, SPH34, MEG168

Overview

In this project, we were supposed to develop a simplified simulator to evaluate distributed directory-based MSI cache coherence protocol in CMPs(Chip multi-processors). In multiprocessors, every processor can have their own L1 caches and shared L2 caches and memory. The coherence protocol helps multi-processors to manage read and write cache/memory accesses by respecting the causal consistency. This protocol ensures that all the processors get the correct value of the cache/memory location they are accessing, and it can be achieved using a directory for every block of L2 cache with state of the block. The state of the block can either be 'Invalid', 'Shared' or 'Modified'.
In this simulator, we are assuming a tiled CMP with several processors (multiple of 2) connected by a mesh NoC(network on chip). Each tile includes a 32-bit CPU core with a private L1 cache and a shared L2 cache among all cores. The coherence protocol is put between L1 and L2 since L2 is the highest level of cache/memory which is shared by processors. We are also assuming that the mesh has a simple timing model with the delay of messages between any two nodes being calculated by Manhattan distance between them multiplied by network delay per hop(C cycles). This simulator is configurable with respect with size of caches, set associativity of caches, block size and delay required to communicate messages on the chip.

Configuration Details

As per given by the description of the project, the simulator has following configurable attributes –

- P (= 2^p) – Number of processors
- N1 (= 2^{n1}) – Size of L1 cache
- N2 (= 2^{n2}) – Size of L2 cache
- B (= 2^b) – Block size in Bytes
- A1(= 2^{a1}) - Set associativity of L1 cache
- A2 (= 2^{a1}) - Set associativity of L2 cache
- C – Network delay per hop (cycles)
- d – L2 cache access (cycles)
- d1 – Memory access latency (cycles)

Implemented Data Structure for Cache

a. Each block of L1 cache contains the information of
   - tag
   - flag of valid
   - flag of dirty
   - LRU number

   Notice that the combination of flag valid and flag dirty shows the status of block in L1 cache.

- Invalid: valid = 0, dirty = 0/1
- Shared: valid = 1, dirty = 0
- Modified: valid = 1, dirty = 1

b. Each block of L2 cache contains the information of:
   - tag
   - flag of valid
   - flag of dirty
   - LRU number
     (above are same as L1 cache)
   - Directory of block status

   Notice that the Directory of block status contains the information of:
   - Status: Invalid, Shared or Modified
   - BitMap which indicates if the correspondent L1 Cache owns this block or not. Its length is the number of processors.

Functional Components

1. Read input trace file and config file
   - In this stage, the simulator gets all the configuration data and reads the input trace file line by line. Each line is parsed and the cycle number, core id, type of request and memory location address is stored into a data structure.
2. Initialization of all components (Caches, processors, requests queue per core)
   - All the components of simulator like processors, L1 and L2 caches, and directories for every block in the L2 caches are initialized in this stage.
   - All requests from the trace file are divided into different queues with respect to their core ids.
   - For the request list in each core, compute the difference between cycle numbers of two sequential requests and save it into the data structure of request.
3. Issue Requests
   - In this stage, requests to access a memory location are issued.
   - Requests available at each core will be sent for processing sequentially. That means any request at one core will not be sent to access memory until that core's previous request is processed completely.
4. Read and write
   - Read Access –
     i. If it is read hit in L1, then it will be processed in the same cycle as we are assuming zero delay for L1 cache access. The status of local node becomes **shared**.
     ii. If it is a miss in L1, then the request is sent to the Home node cache
         - Add the delay of L2 hit time d cycles to the total processing time of this request.
         - If it is a miss in L2, a message is sent to the memory controller at tile 0. The block is brought into the Home node L2 cache using LRU replacement and, as the delay for a memory access is uniform, a delay of d1 cycles is added to the total processing time of this request. The initial Directory Status of the new

fetched L2 cache block is set to be **invalid**. After the data arrives at the Home node, the request is processed the same as an L2 hit.

- If it is an L2 hit and
    a. if the status of the block is **invalid** or **shared**, then the Home node will send the data to the requesting node(Local).
        i. Delay is calculated by adding both the control message delay from Local to Home and data message delay from Home to Local.
        ii. Status of Local block keeps **shared.** Directory Status of the home tile L2 cache (H) sets to **shared** and the local node number is added to list of sharers.
    b. If the status of the block is **modified**, then the Home node will send the id of the data block's owner (Remote) to the requesting (Local) node. The Local node will send a request to this Remote node. The Remote node will send the data back to the Local node and at the same time send a message to the Home node to revise the entry.
    i. Delay is calculated by adding the control message delay from Local to Home and from Home to Local, control message delay from Local to Remote, and the data message delay from Remote to Local or from Remote to Home (add the bigger one).
    ii. Status of Local block and Remote block are set to **shared**. Directory Status of the home tile L2 cache (H) sets to **shared** and the shared list will be Remote and Local.

- Write Access –
    i. If it is a write hit in L1 and the block was **modified** (dirty), then it will be processed in same cycle as we are assuming zero delay for L1 cache access.
    ii. If it is a write hit in L1 and the block was **shared**, then
        - A message will be sent to the Home node. The Home node will send the list of sharers of the block. The Local node will send invalidate message to all sharers (except for Local itself). These messages are sent in parallel. All Remote nodes on the sharer list will send acknowledgements to the Local node and then the Local node will revise entry to Home node.
        i. Delay is calculated by adding the control message delay from Local to Home and from Home to Local, the maximum control message delay from Local to all Remote nodes and the control message delay for acknowledgements, and the control message delay from Local to Home.
        ii. The status of Local L1 cache becomes **modified**. All Remote blocks sets to **Invalid**. Directory Status of the Home tile L2 cache (H) sets to **modified** and the owner is Home tile.
    iii. If it is a write miss in L1, then the request is sent to the Home node cache
        - Add the delay of L2 hit time d cycles to the total processing time of this request.
        - If it is a miss in L2, a message is sent to the memory controller at tile 0. The block is brought into the Home node L2 cache using LRU replacement and, as the delay for a memory access is uniform, a delay of

d1 cycles is added to the total processing time of this request. The initial Directory Status of the new fetched L2 cache block is set to be **invalid**. After the data arrives at the Home node, the request is processed the same as an L2 hit.

- If it is an L2 hit and
    a. If the status of the block is **invalid**, then the Home node will send the data to the requesting node (Local).
        i. Delay is calculated by adding the control message delay both from Local to Home and from Home to Local.
        ii. Status of Local block sets **modified.** Directory Status of the home tile L2 cache (H) sets to **modified** and the Local node number is the owner.
    b. If the status of the block is **modified**, then the Home node will send the id of the data block's owner (Remote) to the requesting (Local) node. The Local node will send a invalidate message to this Remote node. The Remote node will send the data back to the Local node and at the same time send a message to the Home node to revise the entry.
    i. Delay is calculated by adding the control message delay from Local to Home and from Home to Local, control message delay from Local to Remote, and the data message delay from Remote to Local or from Remote to Home (add the bigger one).
    ii. Status of Local block sets to **modified**. Remote block are set to **invalid**. Directory Status of the home tile L2 cache (H) sets to **modified** and the owner is the Local tile.
    c. A message will be sent to the Home node. The Home node will send the list of sharers of the block. The Local node will send invalidate message to all sharers. These messages are sent in parallel. All Remote nodes on the sharer list will send acknowledgements to the Local node and then the Local node will revise entry to Home node.
    i. Delay is calculated by adding the control message delay from Local to Home and the data message delay from Home to Local, the maximum control message delay from Local to all Remote nodes and the control message delay for acknowledgements, and the control message delay from Local to Home.
    ii. The status of Local L1 cache becomes **modified**. All Remote blocks sets to **Invalid**. Directory Status of the Home tile L2 cache (H) sets to **modified** and the owner is Home tile.
- Remarkable points –

    i. When a L1 cache block of tile T is replaced by the LRU strategy, we remove T from the shared/owner list of the Directory Status of the replaced block's Home node. If T is the only tile in the shared/owner list, then the Directory Status is set as **invalid**. This implementation makes sure that every sharers in L2 cache block Directory Status is definitely valid.

ii. Similarly, when a L2 cache block is replaced by LRU, we set all the L1 cache block in the shared/owner list of the Directory Status to **invalid**. This implementation makes sure that every valid block in L1 cache is definitely in L2 cache.

5. Print statistics
   - While processing requests as read or write and miss or hit, all the misses, communication delays, and counts of control messages and data messages are updated into a statistics data structure which stores following data:
     i. Total cycles taken by each core for all requests
     ii. L1 misses for each core
     iii. L2 misses for each core
     iv. Total number of accesses (miss + hit) in each L1 cache module
     v. Total number of accesses (miss + hit) in each L2 cache module
     vi. Total miss penalty for each L1 miss in each core
     vii. Total number of control messages for all requests for all cores
     viii. Total number of data messages for all requests for all cores

Output - Using the above data, we have calculated

General Statistics:

- Total number of control messages used during processing of all requests
- Total number of data messages used during processing of all requests
- L2 cache miss rate

Statistics per tile:

- Cycles to complete all requests by each core,
- Miss rate of L1 and L2 cache for each module
- Average L1 miss penalty for each core

In debugging mode, a detailed log of messages is produced during execution and the content and state of each cache location is produced at the end of execution.

Assumptions –

1. We are assuming a tiled CMP with P processors connected by a mesh NoC(network on chip)
2. Each tile had a 32-bit CPU core with private L1 cache.
3. Each tile (and thus each core) has an L2 cache. These L2 modules form a shared L2 cache of size P * (size of L2 cache) which is shared by all cores.
4. There is a network on chip to connect all the processors and we will be sending messages over this network to communicate and send data between two different cores. This network has a large bandwidth and is contention-free.
5. We are assuming that this network has a simple timing model that means time required to send message or get data between any two nodes can be calculated by finding 'Manhattan' distance between two cores on this chip multiplied by some fixed number of cycles which can be configured.
6. Main memory access is uniform memory access(UMA) which takes some fixed number of cycles(d1).

7. Write-back policy for caches/main memory and large write buffers, which takes write-back operations out of the critical path.
8. For implementing distributed directory based MSI cache coherence protocol among L1 caches, we are assuming that there is a directory entry associated with every block in L2 cache.
9. L1 hit time is zero cycles. L2 hit time is d cycles. L1 miss penalty will be computed by using the delay required to achieve coherence in L1 caches and L2 cache and/or memory hit time.
10. During block replacement, invalid blocks will be prioritized for eviction, but in case of no invalid blocks, the LRU replacement strategy is used.
11. Message exchange between tiles for sending data or updating status are considered instantaneous. At the same cycle when a request is issued, all the block and directory status are assumed to be updated. We compute the delay presented in the section **Functional Components** as the miss penalty. The request is not considered satisfied until the number of cycles as the miss penalty have passed.
12. At each core, requests are processed in sequential manner and if there is difference of d cycles between two consecutive requests at one core, then the second request is issued only 'd' cycles after the first process is satisfied.
13. We don't count for statistics in our new version of code that the requesting status to Home node is a L2 access when Local node has a write hit because the Home node always get a L2 hit. We considered it as a L2 access in the old version.

Experiments –

When we set configuration parameters with following value –
P = 16 (p=4), n1= 13, n2=16, b = 5, a1 = a2 = 2, C = 2, d = 4 and d1 = 20
And when we compared results of above set of configurations on trace1, trace2 and some of our own trace files we found following results-

1) Doubling the network delay (C = 4) –
    a. The cycles taken by each core increased.
    b. The L1 miss rate at core is unaffected.
    c. The L2 miss rate is unaffected.
    d. The L1 miss penalty is increased.

2) Doubling the associativity of L2 (without changing the network delay)
    a. The cycles taken by each core unaffected.
    b. The L1 miss rate at core is unaffected.
    c. The L2 miss rate is unaffected.
    d. The L1 miss penalty is unaffected.

3) Doubling the L2 cache size (without changing the network delay or L2 associativity
    a. The cycles taken by each core is unaffected.
    b. The L1 miss rate at core is unaffected.
    c. The L2 miss rate is unaffected.
    d. The L1 miss penalty is unaffected.

Analyzing the statistics on the standard parameters, some straightforward information about the simulation can be inferred from the results. Primarily, in many of the shorter test cases (t1-6, s1-8, s10), all of the memory accesses had the same home tile. This can be observed from the L2 miss rate per tile (only tile 0 has any miss rate). Additionally, observing L1 miss penalties in simpler trace files (e.g. s3) can give information about what occurred beyond the L1 miss. For example, in trace file s3, the L1 miss penalty is greater in tile 0 than in tiles 2 and 3. This seems counterintuitive, because all the memory accesses use home tile 0 for L2 access, but the L1 miss in core 0 results in a L2 miss as well. Since there are no other memory accesses on tile 0, it does not take advantage of this L2 proximity. Since tiles 2 and 3 use blocks already in L2 (stored in L2 due to accesses from tiles 0 and 1), tiles 2 and 3 do not pay the increased penalty of an L2 miss.

Analyzing the statistics using different parameters, doubling the network delay had the expected result of increasing the cycles taken by each core. The network delay is the largest contributor to the total time needed by each core to complete all its requests, and so doubling this delay will significantly affect the total time. The network is only used on L1 misses, and so the average L1 miss penalty increases as well. The L1 and L2 miss rates should not be affected by the network delay, and our simulation reflects that.

Doubling the associativity and/or the size of L2 did not affect any of the statistics above because the storage capacity of L2 is not a bottleneck of the system. L2 misses were entirely compulsory, caused by never having used the block before, and our L2 was sufficiently large such that blocks were not being evicted when they still needed to be used. In some traces, this resulted in a very high L2 miss rate, as L1 was also not evicting many blocks. L1 misses would result in L2 misses (both were compulsory) but a block stored in L1 would be reused as needed and few L2 accesses would be hits. We designed some other traces that would result in L1 evictions, increasing the L1 miss rate, but this drastically reduced the L2 miss rate. Each L1 miss would result in an L2 access, but because L2 was still not evicting blocks, any repeat access to a block was an L2 hit. Increasing the capacity of L2 would not affect any of the misses, as they are again all compulsory.

Test Results –

Trace File Name	What is it trying to test?	Test Status (Successful/Seg Fault/Infinite Loop)	Reason behind Test Status	Did you fix the error?	How did you fix the error?	Additional Comment
trace0	basically read miss and write miss	Successful	\	\	\	
trace1	long trace file	Successful	\	\	\	
trace2	some cores have no instructions	Successful	\	\	\	
t1	read miss, read hit and L1 cache associativity replacement	Successful	\	\	\	
t2	read miss, read hit and L1 cache associativity	Successful	\	\	\	

t3	write hit on shared block	Successful	\	\	\	
t4	read miss and write miss on uncached block	Successful	\	\	\	
t5	write miss on shared block	Successful	\	\	\	
t6	write miss on modified block	Successful	\	\	\	
t7	read miss on invalid or shared block	Successful	\	\	\	
s1	write miss and read miss on invalid block, block size	Successful	\	\	\	
s2	read hit and write hit, block size	Successful	\	\	\	
s3	read and write hit	Successful	\	\	\	
s4	read hit and read miss on shared block	Successful	\	\	\	
s5	read miss and write hit on shared block, check L2 hit	Successful	\	\	\	
s6	read miss and write miss on shared block, check L2 hit	Successful	\	\	\	
s7	check the requests are issued after considering delay in the same core	Successful	\	\	\	
s8	check L2 hit	Successful	\	\	\	
s9	read and write miss on modified block	Successful	\	\	\	
s10	read and write miss on invalid block	Successful	\	\	\	

Changes of code in the final version:

1. Initial cache.h has a bug in function updateLRU, fixed it by changing < to =< which means it should also add equivalent LRU to 1.
2. Remove the L2 access counter when Local node gets write hit and it requests to Home node because this L2 access is always a L2 hit.