Project 4: Simulation of Directory-based Cache Coherence: Programming Project

About Coherency:

Coherency refers to the consistency of data shared between different cores or processing units in a multi-core processor. In a multi-core system, each core has its own local cache, which stores copies of data from the main memory. In a single program, different threads might run on separate cores to perform parallel tasks. These threads could be working on different aspects of the same problem and likely share the same data. When multiple cores must work with the same data, coherency mechanisms ensure that all cores see a consistent view of that data.

About Directories:

A Directory is just one entity that is commonly used in Network architectures to maintain data coherency between the cores. There are many ways of sizing and organizing the directory, and we consider a model where there is a directory entry for every block of the memory.

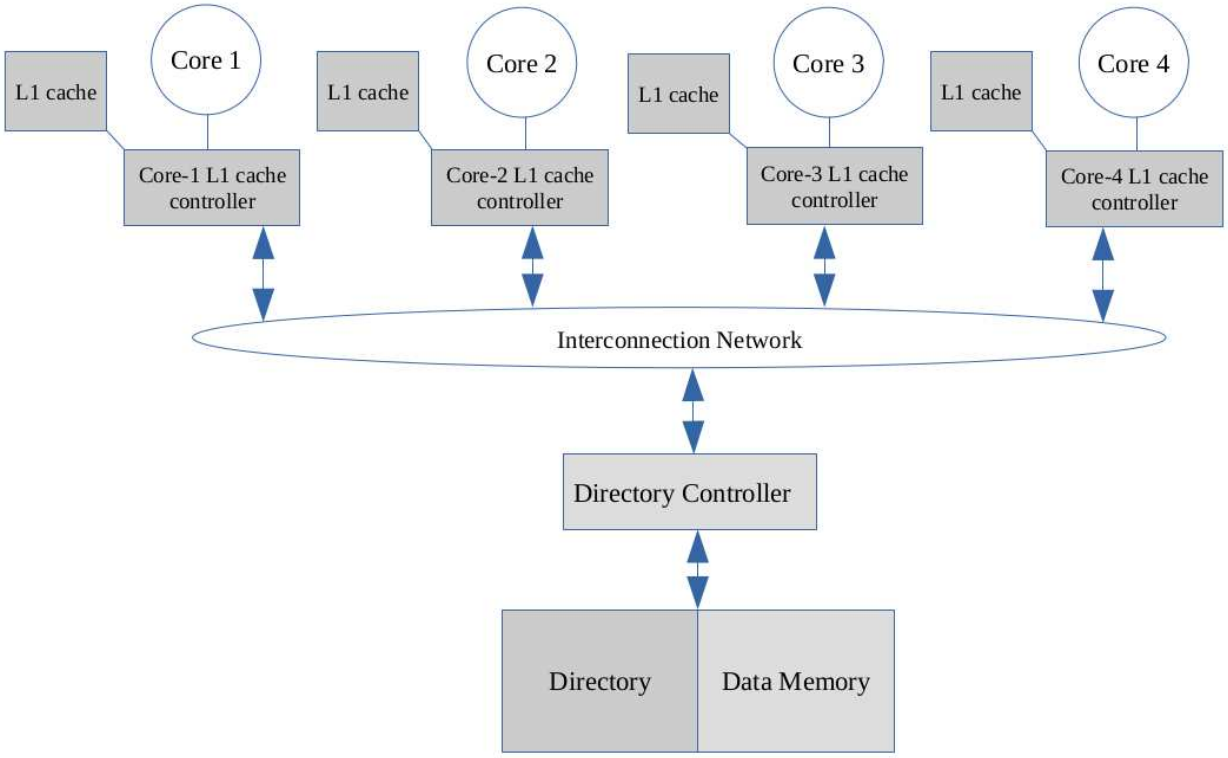

Fig. 1. The directory architecture to be Implemented

The directory will be maintained with the help of an entry for each memory location, as follows:

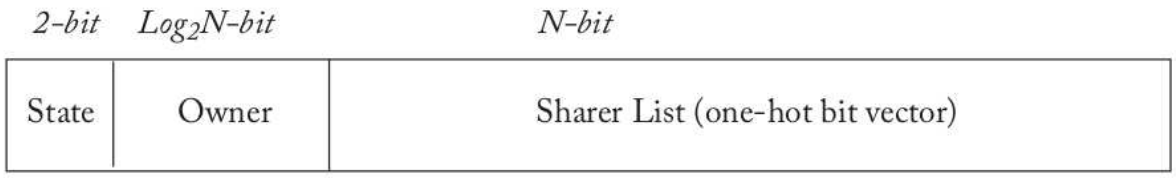

Fig. 2. Format for a directory entry (MSB to LSB going left to right)

Coherency Protocol:

A protocol is usually followed to effectively maintain data coherency. This is facilitated with the help of *transactions* between the cores and the directory over the interconnect. For the purpose of this project, you do not need to implement the interconnect network, but you can assume it is in place when making *requests, transactions, and responses* (explained later). The coherency protocol to be followed is *MOSI* (Note: subtle differences exist in the state implementations for MSI, MOSI, and other protocols). Each state in MOSI has been described below for clarity:

State symbol	State binary	State	Description
M	00	Modified	Data is up-to-date, can be read or written, and is exclusively owned by the core
S	01	Shared	An up-to-date read-only copy of data, shared by at least 2 cores (One of the cores can also be the owner)
I	10	Invalidated	A core having a copy of an address in state M invalidates the same address for other cores
O	11	Owned	Every cached directory entry has an owning core, which has the latest, read-only copy of the address data

Project Description:

Every core is identical and follows the same set of instructions to communicate with the cache controller to fetch data from the memory - into the cache. Following are the instructions, their description, and the reachable state(s) because of their behavior:

I-no	Instruction semantic	Instruction Description	Facilitates state
1	<Core> LS <address>	Fetches read-only address location from memory to L1 cache	O / S (only S for MSI protocol)
2	<Core> LM <address>	Fetches address location with read-write access - from memory to L1 cache	M / I
3	<Core> IN <address>	Invalidates address with the core	I
4	<Core> ADD [address] #immediate	Adds immediate value to the data stored in the memory location and overwrites the result into the same location	M / I

I.e., Each core can issue load requests to its cache controller; the cache controller will choose a block to evict when it needs to make room for another block.

The Cache and Memory Organization:

1. The main-memory contains 64 address locations, and each L1 cache has 4 address locations.
2. Initialize the memory array with zeros at all locations.
3. Cache line width = size of memory location = 1 byte (byte-addressable)
4. The L1 caches must be *2 way set-associative*, and the L1 controller follows the *LRU (Least-Recently Used) replacement policy*.
5. The L1 cache controller also follows the *write-through policy* in the same cycle the data in a particular address location is modified.

Requests, Transactions, and Responses:

Transactions between the cache controller and the directory controller are necessary to provide the cache read/read-write access to a particular block in one of the mentioned coherence states (M, O, S, or I).
The request for these transactions is made by the core and passed on to its cache controller, which initiates the appropriate transaction and gets a response from the destination. [2]
Acknowledgment from the destination as a response is omitted for the purpose of this project since every request does get fulfilled in this simulation.
Valid Transaction for each of the mentioned instructions:

Transaction	Instructions that initiate the transaction	Description
GetShared	LS	1. Checks whether address is owned and assigns ownership if address was unassigned. It provides read access to the core for a particular address. 2. In case of owned address - The latest copy is also provided by the owner as a *response* to a core's *request*
GetModified	LM, ADD	Provides read-write access and ownership of the address along with the latest copy from the owner - as a *response*
Put	IN, LM, ADD	The local copy is invalidated, and the data block is *evicted* from the cache to the main memory in the following cycle

Mid-Project Deliverables:

1) The simulator must be able to read and interpret the instructions file
2) Implement directory-based coherence for a dual-core system with MSI protocol (with the earlier mentioned cache and memory specifications, but memory should have 32 address locations, and the caches are *fully associative* following the *random replacement policy*)
3) The architecture should be represented accurately (as displayed in Fig1, but for a dual-core system) with all its entities and their functionality defined completely.
4) The interpreted instructions should generate the right requests, transactions, responses, and Directory updates.
5) Generate a log file with:
   (i) The cache memory dump after executing each instruction of the program
6) The final state of the directory entries and the corresponding memory dump should be printed after processing the entire program.
7) Make test cases to exhaustively test and prove the working of your architecture

End-Project Deliverables (in addition to points 3 - 7 of MPD) :

1) The architecture should be extended to a quad-core system with the MOSI protocol and should still be able to read and interpret the instructions file
2) The memory and cache specifications are mentioned on the previous page
3) Make the following Plots after executing the program:
   (i) Plot the average memory access latency and miss rates for each of the cores after processing the program. (assume miss penalty = 2 * hit time, time unit = clock cycle, and it takes 1 clock cycle to execute 1 instruction)
   (ii) Plot the number of directory updates with time (1 clock cycle to execute 1 instruction)
4) Generate another log file:
   (ii) Core Request, generated Transaction(s), and their Response(s) should be written in human-readable format - into a log file for each instruction of the program.
5) A README file should contain the following details:
   a) The working and usage of your simulator, along with the instruction semantics you followed.
   b) The building instructions (if any).
   c) The usage instructions use the command line interface.

Reference Material:

1. Chapters 6 (detailed definition of each state) and 8 (directory based implementation of the MSI/ MOSI protcol), A Primer on Memory Consistency and Cache Coherence
2. CMU Lecture on Directory-based coherence