CSC/ECE506: Architecture of Parallel Computers
Program 4: Simulating DSM Coherence
Due: Tuesday, April 22, 2025

1. Problem Description

In this project, you will need to code the **SSCI protocol** and run a simulator for a trace-driven DSM (distributed shared memory) system. You should run the simulator for various cache configurations. The purpose of this project is to give you an understanding of DSM architecture, evaluate it and interpret performance data.

2. Simulator

The specifications of the DSM system are as follows.

- The system will consist of 16 nodes. There will be a single-level cache for all the nodes, which follow the MESI invalidation coherence protocol. All the caches will have the same configuration, which will be given as input parameters to the simulator. This part is the same as in Program 3.
- In a DSM system, the memory is spread across all the nodes in the system. However, for the simulation, you don't have to take the memory organization into consideration. You only need to handle a directory structure along with the caches.
- In a real DSM system, the directory is also distributed among the nodes. However, in this simulation, for simplicity we will use a single directory structure. In operations on the directory, the processor number will be passed as a parameter.
- The directory information will be stored as a linked list. The directory structure will have a limited size, equal to the total number of blocks that can be cached. For example, if the cache size is 32768 (32KB) and block size is 64 , then the total number of blocks that can be cached at one processor is

  $$\frac{32768}{64} = 512$$

  Therefore, the total number of directory entries in the system is 512 × 16 = 8192. This format is known as a sparse directory format, which is described in the textbook. In Lecture 22, we called it "reducing the height" of the directory.

In order to run the simulator, you need to execute the following command:

```
./dsm ⟨cache_size⟩ ⟨assoc⟩ ⟨block_size⟩ ⟨num_processors⟩ ⟨dir_type⟩ ⟨trace_file⟩
```

where—

dsm: Executable of the DSM simulator generated

cache_size: Size of each cache in the system (as in Program 3, all caches are the same size, for example, 65536)

assoc: Associativity of each cache (all caches are of the same associativity)

block_size: Block size of each cache line (all caches are of the same block size)

num_processors: Number of processors in the system (represents how many caches should be instantiated)

dir_type: 1 – SSCI; 0 - FBV

trace_file: The input file that has the multithreaded workload trace.

The trace files are on Google Drive.

Each trace file consists of a sequence of memory transactions; each transaction consists of three elements:

⟨processor (0-15) ⟩ ⟨operation (r,w) ⟩ ⟨address (8 hex chars) ⟩

For example, if you read the line "`3 w 0xabcd`" from the trace file, that means processor 3 is writing to the address "`0xabcd`" to its local cache. You need to propagate this request down to cache 3, and cache 3 should take care of it.

The Streamcluster trace is about 0.4 GB, the Blackscholes trace is about 6.75 GB, and the Bodytrack trace is about 6.9 GB. The shell scripts include runs of all three bencharks. The shell scripts should take around 30 minutes or less each.

To compile the program, run make all. To remove the .o files when recompiling, please run make clean before rerunning make all.

To test your implementation of the SSCI protocol, use the test_ssci.sh shell script. To be able to run this shell script, please run chmod +x test_ssci.sh and then ./test_ssci.sh.

3. Simulator

For this problem, you will experiment with various cache configurations and measure the cache performance of a number of processors across the two different trace files mentioned above. The cache configurations that you should try are:

- Cache size: vary from 32KB, 64KB, 128KB, 256KB, 512KB, 1024KB, 2048KB while keeping the cache block size at 64B. (Please note that 1KB is 1024 bytes 128KB is 131072 bytes) (Not giving the proper values will result in points being deducted.)
- Cache associativity: keep it constant at 4-way.
- Cache block size: vary from 64B, 128B, 256B, 512B while keeping the cache size at 1024KB

Do all the above experiments for SSCI using the Streamcluster trace. For each simulation run, collect the following statistics:

1. Number of read transactions the caches have received.
2. Number of read misses the caches have suffered.
3. Number of write transactions the caches have received.
4. Number of write misses the caches have suffered.
5. The total miss rate of the caches.
6. Number of writebacks when lines are replaced (victimized) from a cache.
7. Number of cache-to-cache transfers.
8. Number of `SignalRd`s
9. Number of `SignalRdX`s
10. Number of `SignalUpgr`s
11. Number of invalidations
12. Number of interventions

For your report, you should only print out aggregate statistics (e.g., the total number of read transactions that all 16 caches have received), though you may find it useful to print out statistics on individual caches for debugging purposes. However, for the miss rate, take an average of this.

Present the statistics in tabular format as well as figures in your report. Discuss the trends with respect to change in the configuration of the system. Discuss the various aspects of the coherence protocols. You can discuss the bus traffic, number of memory transactions, and complexity of the protocols.

Answer the following questions, using the provided shell scripts for each question.

1. Experiment with SSCI increasing the cache line size. Find the optimal cache line size. If a minimum is found, why is this the case?
2. How does increasing associativity affect a small cache as compared to a large cache when using SSCI?
3. How does the number of cache invalidations change as cache size increases using SSCI?
4. Does increasing the cache line size benefit a workload where multiple processors are reading/writing the same data frequently using SSCI?

4. Submission

Submit a prose report on your results, including all the statistics as mentioned in Section 3.

5. Grading

Credit will be given on the statistics shown and discussion presented.

6. Suggestions

1. Read the main, cache, and directory classes carefully, and understand how a single cache works.
2. Start early and post your questions on Piazza.
3. Gain access to using the shell scripts by running chmod +x q#.sh. Then, run the scripts by running ./q#.sh.
4. Log on to the VCL at vcl.ncsu.edu and choose the Ubuntu 22 GPU with Cuda (GeForce RTX 2080 Ti) environment. Note: You can also use other VCL images of Ubuntu 22.
5. The trace files should be copied into the traces/ folder within program4.
6. The code file to edit is ssci.cc.