ECEC 414
Simulation Assignment – Cache Coherence experiment

Due February 21st 2014

1 Overview

In this assignment, you will be simulating an X86 CMP and studying the scalability of two different types of cache coherency protocols, namely MESI and MOESI. Using the Ruby memory simulation model, you will simulate a distributed directory scheme for cache coherence. A block diagram of the system is shown in Figure **??**. The L2 cache is shared amongst the processors on chip. However, it is distributed and 'pieces' of the L2 reside on different parts of the chip. For the sake of simplicity and convenience, the number of 'pieces' of the L2 cache as will be the same as the number of CPUs. In similar fashion, the directory is distributed and is co-located with each piece of the L2. As a result, you will observe as many L1 and L2 cache controllers as there are CPUs. Your job will be to scale up the number of cores and explain the trends observed in all schemes. You will need to pick a sensible set of metrics from your results to explain the results you see.

A FEW IMPORTANT CONSIDERATIONS:

1. Since we are simulating multiple cores and threads with a more sophisticated memory model (Ruby), you can expect simulations to take several hours. Hence, it will not be possible to start this assignment late and hope to complete it.
2. The system is not 'hanging' when the simulation is running. You may stop seeing periodic messages and control will return to the terminal when the simulation has finished.
3. The steps in this assignment are not the same as the previous assignment, so please don't jump ahead in the instructions.
4. Note if you copy paste commands from this PDF file into your command line, underscores will be replaced with spaces automatically, so you will need to fix that manually.

Informative references: For more information on Ruby and the specific Directory scheme, please refer: http://www.m5sim.org/Ruby For more information on the network topology being employed, please refer: http://www.m5sim.org/Interconnection_Network For an overview of the MESI implementation see: http://www.m5sim.org/MESI_CMP_directory

2 Getting Started

In this assignment, we will use the latest version of the GEM5 simulator. Please remove your old GEM5 folder from the previous assignment if you still have it. See the following link if you need help: *http://www.cyberciti.biz/faq/delete-or-remove-a-directory-linux-command/*. To work with the latest version of the GEM5 simulator, you will need to download the simulator and untar it by following these steps:

1. Login to `xunil.coe.drexel.edu` via SSH with your Drexel user id and password. Note: You should use either the linux tool "screen" or computer with a reliable Internet connection; both build and simulation tasks can take over an hour.
2. To get the latest GEM5 version, download the GEM5 simulator from gem5.org to your home directory using the following command:
   `> wget http://repo.gem5.org/gem5-stable/archive/6a043adb1e8d.tar.bz2`

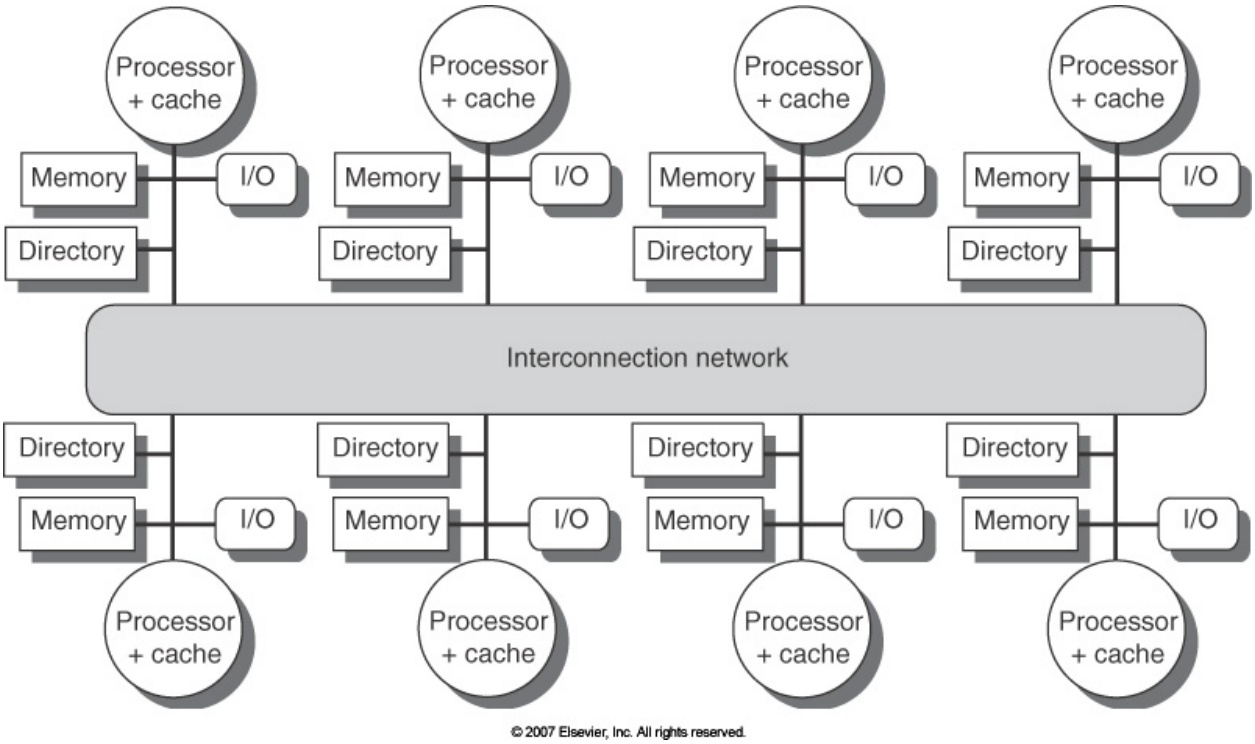

Figure 1: Diagram of simulated CMP. Pieces of the distributed shared L2 reside with each processor. *Ref: Computer Architecture, A Quantitative Approach, Fourth Edition*

3. Unpack the archive with the command:
   `> tar -xjvf tip.tar.bz2` (execute `man tar` to get the manual-page for an explanation of the `tar` command)

   You will now have a gem5-stable directory located in your home directory that contains the new GEM5 simulator.

4. Navigate to the gem5-stable directory.
   `> cd gem5-stable`

3 Building an Architecture

GEM5 is a robust architecture simulator that allows you to model many different ISAs, including ALPHA, ARM, POWER, MIPS, and x86. For many of these ISAs, you can configure either in-order and out-of-order (OoO or O3) processor models. The list of available ISAs is in the `gem5-stable/build_opts` directory.

For this assignment, we will be building an out-of-order X86 processor for two coherence protocols (MESI and MOESI). From your gem5-stable directory, execute the following command: `> scons build/X86_MESI_CMP_directory/gem5.opt`
`scons` is similar to `make`, and compiles the code for the simulator, building the gem5.opt executable in the `~/gem5-stable/build/X86_MESI_CMP_directory` directory

Next, you will need a build configuration for the MOESI protocol. Copy X86_MOESI_CMP_directory from my home folder to your build_opts folder using the following command:

```
> cp /home/DREXEL/sn446/X86_MOESI_CMP_directory build_opts/.
```

Then execute the following command to build the executable for the MOESI protocol:

```
> scons build/X86_MOESI_CMP_directory/gem5.opt
```

This will build another gem5.opt executable in the ∼/gem5-stable/build/X86_MOESI_CMP_directory directory.

You will also need to copy the seSplash.py file from my folder to your scripts folder in the GEM5 directory. From your gem5-stable directory, execute the following command:

```
> cp /home/DREXEL/sn446/seSplash.py configs/example/.
```

4 Using GEM5 for X86

As with Assignment1, we will execute the Splash-2 benchmarks on GEM5. Splash2 contains several standard algorithms:
Cholesky, FFT, LUContig, LUNoncontig, Radix, Barnes, FFM, OceanContig, OceanNoncontig, Raytrace, WaterNSquared, and WaterSpatial.

Here are the steps to run a splash-2 benchmark on the new GEM5 :

1. Navigate to your home directory using the following command:
   ```
   > cd ∼
   ```

2. Copy the Splash-2 benchmarks compiled for the X86 architecture from my home folder using the following command:
   ```
   > cp /home/DREXEL/sn446/splash-x86.tar.bz2 .
   ```
   You will need to delete, rename or move your old copy of Splash2 used for the previous assignment. Please do so before executing the next step.

3. Extract the splash-2 benchmarks:
   ```
   > tar -xjvf splash-x86.tar.bz2
   ```
   You now have a directory ∼/splash2 containing the benchmarks

4. Navigate to one of the two build directories depending on whether you want to run the MESI or MOESI protocol `> cd ∼/gem5-stable/build/X86_MESI_CMP_directory` or `> cd ∼/gem5-stable/build/X86_MOESI_CMP_directory`

5. Invoke the GEM5 simulator with the following command line:
   ```
   > ./gem5.opt ∼/gem5-stable/configs/example/seSplash.py --ruby --num-cpus=4
   --l1d_size=32kB --l1d_assoc=4 --l1i_size=32kB --l1i_assoc=4 --l2_size=256kB
   --l2_assoc=8 --cacheline_size=64 -b FFT --topology=Mesh --mesh-rows=1
   --rootdir=/home/DREXEL/< drexel_id >/splash2/codes
   ```
   –ruby tells GEM5 to use the ruby memory model. You will always need to use this for this assignment –l1d_size configures the L1 data cache size
   –l1d_assoc configures the L1 data cache associativity
   –l1i_size configures the L1 instr. cache size
   –l1i_assoc configures the L1 instr. cache associativity
   –l2_size configures the shared L2 cache size
   –l2_assoc configures the shared L2 cache associativity

Coherence Protocol	Network Topology
MESI	Crossbar
MESI	Mesh
MOESI	Crossbar
MOESI	Mesh

Table 1: 4 configurations to be simulated

–cacheline_size configures the cache line size, the amount of data transferred from memory on a cache miss.
–num-cpus=4 tells GEM5 to create a system with 4 CPUs and invokes the benchmark with 4 threads
-b FFT tells GEM5 to run the FFT benchmark
–rootdir tells GEM5 where to find the benchmarks, replace $< drexel\_id >$ with your username
–topology tells GEM5 which topology to use. You will be using the 'Crossbar' and 'Mesh' topologies for this assignment
–mesh-rows needs to be specified when using the Mesh topology. It should not be used for the Crossbar topology
Please read up more information on configurations in the links provided at the top of the assignment.

6. You should see something like "Exiting @ tick no. because target called exit()" when the simulation completes. At this point, control of the command line will be returned to you. If there is too much load on the server, your simulations could take longer than usual. In that case, you are allowed to add the following command line option to stop the simulation after a fixed number of instructions: `--maxinsts=400000000`

The simulator will run and generate output in the directory ./m5out. Here you will find `config.ini` containing the simulation parameters and `ruby.stats` containing the results of the simulation.

5 Thread scaling in NUCA experiment

The goal of this experiment is to evaluate how various cache coherence protocols and topologies scale with the number of threads. In the CMP architecture you will be simulating, the shared structures (L2 and Directory) will be distributed and connected by a network instead of a bus. Such a configuration could result in a Non-Uniform Cache Architecture (NUCA) as in the case of the Mesh topology. In NUCA, we can expect variable latency in shared cache accesses. You will use GEM5 to investigate the tradeoffs in network traffic and execution times in a few configurations.

5.1 Procedure

Keep the L1s capacity fixed at 32KB, L2s capacity fixed at 256KB, block size fixed at 64B and vary number of processors (4, 8, 16). Run the following 4 different coherence combinations specified in Table **??**. In the m5out folder that is generated, you will find a ruby.stats file that contains the results you are interested in. You must find the appropriate metrics that inform the trade-offs for the various configurations.

Pick any two benchmarks of your choosing out of the Splash2 suite for this experiment: `Radix, Barnes, FMM, OceanContig, OceanNoncontig, WaterNSquared,` or `WaterNSquared`.

5.2 Report

Write a brief report that outlines your results and conclusions. You should have a section introducing and explaining your configuration and a section that explains your results with the help of plots. Plot both benchmarks in one figure, if possible (note: normalizing a plot will help easily display multiple benchmarks on the same graph). In your analysis, address the following questions:

1. How did varying the number of processors impact Performance and Network messages? Is there a trade-off involved here?

2. Assuming that 10% increase in network messages costs roughly 1% increase in energy, which topology and cache coherence combination would you pick for your particular set of workloads and why?

3. What does doubling the L1 cache size (both I and D) do to both the network traffic and the execution cycles? Why? (Assume 8 processors for this study)