CSE471 Homework Assignment #2: Cache Coherence

Due Tuesday, May 15th, before lecture

The second milestone for the coherence homework requires you to implement your new protocol in a simulator, and to ensure it works by running it on a set of benchmarks. You'll also do a quantitative analysis of the 4-state protocol and write a report that summarizes your work and the conclusions you were able to draw from your quantitative analysis.

Implementing the 4-State Protocol

Getting Started with the Simulation Infrastructure

In this assigment, we will be using another Pin-based simulator called MultiCacheSim. You will be writing a cache coherence plugin for MultiCacheSim that implements your 4-state protocol. There is some shared infrastructure that we will be using, so you will need to work on attu, or any of the department machines that have access to the shared filesystem.

First, download the code archive from the course website. To get you started on your implementation, we have provided you with a plugin that implements the 3-state protocol we studied in class (MSI_SMPCache). It will benefit you to read this code carefully to understand how the simulator works. Brandon will discuss the 3-state protocol's implementation during Section.

To build the 3-state protocol plugin, run `make MSI_SMPCache.so`.

To run MultiCacheSim and load this cache coherence plugin, run:
`/cse/courses/cse471/12sp/PIN/pin-2.9-39599-gcc.3.4.6-ia32_intel64-linux/pin -t`
`/cse/courses/cse471/12sp/Assignments/HW2-CacheCoherence/Release/MultiCacheSim_PinDriver.so`
`-protos ./MSI_SMPCache.so -numcaches 8 -- /usr/bin/gcc`

This command will run `/usr/bin/gcc` in a simulation of 8 processors with caches that are kept coherent by the protocol implemented in MSI_SMPCache.so. You can vary the number of caches that are simulated by changing the *numcaches* command line argument. You can specify a number of protocol plugins as a command line argument to the *-protos* option as a comma separated list.

Protocol Plugins

A protocol plugin keeps caches in a MultiCacheSim simulation coherent. The provided MSI_SMPCache is an example that illustrates the essential components of a protocol plugin.

A protocol plugin must implement 4 interface methods:

1. readLine(readPC, addr)

2. writeLine(writePC, addr)

3. readRemoteAction(addr)

4. writeRemoteAction(addr)

`readLine()` **and** `writeLine()`

The readLine and writeLine methods implement protocol actions taken in the event of a read or write. These methods must do several things:

1. check the tags of the block to determine if the access is a miss

2. call `readRemoteAction()` or `writeRemoteAction()`, which implements the snoop in remote caches

3. determine the correct state in which to cache the block being accessed

4. cache the block

`readRemoteAction()` **and** `writeRemoteAction()`

These two methods implement snooping of remote transactions on the bus. The methods are called by a simulated processor when it makes a memory access. In these methods the processor iterates through the other caches, updating their state as though they have snooped its memory access. `writeRemoteAction` is called from within `writeLine()` and `readRemoteAction` is called from within `readLine()`.

These methods return messenger objects that provide information about coherence state. `readRemoteAction` returns an object with two fields: `isShared` and `providedData`. `isShared` should be true if the line being accessed is in *shared* state in a remote processor. `providedData` should be true if the snooping processor put its data on the bus for the accessing processor. The fields of this messenger object can be used to determine whether an access was serviced by a remote cache, or by memory.

`writeRemoteAction` returns an object with one field, `empty`, which is not necessary for your simulations.

Implementing Your Protocol Plugin When building your protocol plugin, you should work from the provided protocol plugin as a base. It implements most of the functionality you will need, and illustrates many helpful simulator mechanisms (how to get a cache line's state, for example). You should be able to implement your protocol by understanding the implementation of the four methods that we've given you, and changing the protocol logic to implement the fourth state.

We will provide you with a reference solution. You can run your plugin alongside the reference solution plugin, and the two implementations should emit the same final output. The reference solution will give you a way to check your work, and be sure that the protocol you end up implementing is correct.

Benchmarks In this assignment you will be using the PARSEC benchmark suite to both test your implementation and gather data for the quantitative analysis that will go into your report. We used PARSEC in the branch prediction assignment, but here is a reminder of how to use Pin with PARSEC. The command for this assignment is:

```
/cse/courses/cse471/12sp/Benchmarks/parsec-2.1/bin/parsecmgmt
-a run -p <benchmarkname>
-d <workdirectory>
-n 8 -s '/cse/courses/cse471/12sp/PIN/pin-2.9-39599-gcc.3.4.6-ia32_intel64-linux/pin
-t /cse/courses/cse471/12sp/HW2-CacheCoherence/Release/MultiCacheSim_PinDriver.so
-protos ./MSI_SMPCache.so,./FOO_Protocol.so -numcaches 8 -- '
```

where <workdirectory> is a writable working directory, and <benchmarkname> is one of the benchmarks.

The `parsecmgmt` command is self-documenting if you run it with no options. Don't forget the single quotes in the `-s` option.

Your quantitative analysis should compare the performance and the scalability of the 3-state and 4-state protocols. You'll use the output provided by the simulator to do this analysis. Be sure to utilize the component metrics, so that you know not just which protocol works best, but *why*.

Evaluating the 4-State Protocol

Experiments

To better understand the impact of the 4-state protocol, you will conduct a series of experiments.

When you run your simulations, you should run your 4-state protocol implementation alongside the provided 3-state protocol implementation. By doing so, you will get numbers that came from the same execution trace, and can, therefore, be compared directly. Recall that you can specify a comma-separated list of protocol plugins to MultiCacheSim to simulate multiple protocols on the same execution.

Part of your analysis will be to evaluate the relative scalability of the 4-state protocol to the 3-state protocol. In order to do this analysis, you will have to run your experiments three times – once with 2 threads, once with 4 threads, and once with 8 threads. Take note to change the thread count option that you are providing to `parsecmgmt` (*e.g.*, `-n 8`) and the one you are providing to MultiCacheSim (*e.g.*, `-nCaches 8`).

One goal of this assignment is to give you experience dealing with the heaps of semi-structured data produced by architectural simulators. It is up to you to keep your data organized. Keeping things neat will help you stay focused on the interesting parts of the experiments.

The Analysis

After completing your experiments, you will analyze the data collected from these experiments. Use this data to evaluate the relative scalability of the 4- and 3-state protocols and to support or refute your hypothesis about their performance.

The first step to your analysis should be to identify the simulation outputs that vary between the 3-state and 4-state protocols. You should try to be as inclusive as possible during this step – don't limit yourself to analyzing only the outputs that you expected to vary. It will probably be helpful during this step to focus on a subset of your data. For example, to try to decide what the variables of interest are, you might want to look at only one or two benchmarks at first. You also probably want to restrict your intial attention to a single thread-count. Plotting the data you've collected will help you to see trends in your data.

Once you've decided which variables are of interest, you should figure out why you are seeing the difference you are seeing.

Next, you should look at how the differences in these variables of interest change as you vary the number of threads in the simulation. For example, does the 4-state protocol lead to a *bigger* difference in some output value when there are more threads? What conclusions can you draw regarding the performance scalability of these two protocols from the data you've collected?

The Report

You should write up a report that summarizes the work you've done, and the conclusions you are able to draw from your results. Your report should include the following:

- A brief introduction and problem statement

- A description of your 4-state protocol design, and the shortcomings of the 3-state design that it addresses. This section can be kept brief, as your milestone 1 report should have elaborated on your design decisions in detail.

- A description of your evaluation. You should describe what you intend to show with your experiments. You should describe what experiments you performed. You should discuss your experimental methodology.

- A discussion of your results. This should be a write-up of the quantitative analysis. You should include at least one plot, showing at least one simulator output that varies between protocols. The plot should show how that output varies between protocols with two, four, and eight threads. You should discuss

the implications of the 4-state protocol on the performance scalability of systems. You should also use this section to describe any other interesting properties of your data, such as suprising differences between the protocols, benchmark-specific characterization (if you decide to do any), etc.

The report is a large fraction of your grade for this milestone. Please keep that in mind when you are allocating your time. It is important that you communicate clearly – converting your analysis from raw data to English prose should be a top-level concern when you are budgeting your time, not just an afterthought.