Learning gem5

Modeling Cache Coherence with Ruby and SLICC

Jason Lowe-Power

http://learning.gem5.org/

https://faculty.engineering.ucdavis.edu/lowepower/
gem5 history

M5 + GEMS

**M5**: “Classic” caches, CPU model, master/slave port interface

**GEMS**: Ruby + network
Learning gem5
http://learning.gem5.org/

Part 1: Getting started with gem5
Part 2: Modifying and extending gem5
Part 3: The Ruby cache coherence model
Part 4: Full system simulation

.... More coming
Outline

Ruby overview

SLICC controller details

Configuring Ruby

A few other small things
Ruby components

**Controller models** (e.g., caches)

**Controller topology** (how are caches connected)

**Network model** (e.g., on-chip routers)

- Covered in more detail in the next talk

**Interface** ("classic" ports in/out)

Main goal

**Flexibility, not usability**
Controller Models

Implemented in SLICC

Code for controllers is “generated” via SLICC compiler

SLICC: Specification Language including Cache Coherence
### TABLE 8.1: MSI Directory Protocol—Cache Controller

<table>
<thead>
<tr>
<th>Event</th>
<th>Load</th>
<th>Store</th>
<th>Replacement</th>
<th>FaultGetS</th>
<th>FaultGetM</th>
<th>Inv</th>
<th>Req-Ack</th>
<th>Data from Dir/S</th>
<th>Data from Dir/M</th>
<th>Data from Dir/1</th>
<th>Inv-Ack</th>
<th>Last-Inv-Ack</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>send GetS to Dir/S^3</td>
<td>send GetM to Dir/M^A</td>
<td>stall</td>
<td>stall</td>
<td>stall</td>
<td>stall</td>
<td>stall</td>
<td>/S</td>
<td>-M</td>
<td>-M/A</td>
<td>-M</td>
<td>/S/A</td>
</tr>
<tr>
<td>IM^A</td>
<td>stall</td>
<td>stall</td>
<td>stall</td>
<td>stall</td>
<td>stall</td>
<td>stall</td>
<td>stall</td>
<td>/M</td>
<td>-M/A</td>
<td>/M/A</td>
<td>/M</td>
<td>/M/A</td>
</tr>
<tr>
<td>IM^AD</td>
<td>stall</td>
<td>stall</td>
<td>stall</td>
<td>stall</td>
<td>stall</td>
<td>stall</td>
<td>stall</td>
<td>/M</td>
<td>-M/A</td>
<td>/M/A</td>
<td>/M</td>
<td>/M/A</td>
</tr>
<tr>
<td>S</td>
<td>hit</td>
<td>send GetM to Dir/S^2</td>
<td>send PutS to Dir/S^2</td>
<td>send Inv-Ack to Req/I</td>
<td>/M</td>
<td>-M/A</td>
<td>/M/A</td>
<td>/M/A</td>
<td>/M/A</td>
<td>/M</td>
<td>/M/A</td>
<td></td>
</tr>
<tr>
<td>SM^AD</td>
<td>hit</td>
<td>stall</td>
<td>stall</td>
<td>stall</td>
<td>stall</td>
<td>stall</td>
<td>stall</td>
<td>/M</td>
<td>-M/A</td>
<td>/M/A</td>
<td>/M</td>
<td>/M/A</td>
</tr>
<tr>
<td>SM^A</td>
<td>hit</td>
<td>stall</td>
<td>stall</td>
<td>stall</td>
<td>stall</td>
<td>stall</td>
<td>stall</td>
<td>/M</td>
<td>-M/A</td>
<td>/M/A</td>
<td>/M</td>
<td>/M/A</td>
</tr>
<tr>
<td>M</td>
<td>hit</td>
<td>hit</td>
<td>send PutM+data to Dir/S</td>
<td>send data to Req/I</td>
<td>/M</td>
<td>/M/A</td>
<td>/M/A</td>
<td>/M/A</td>
<td>/M/A</td>
<td>/M</td>
<td>/M/A</td>
<td></td>
</tr>
<tr>
<td>MI^A</td>
<td>stall</td>
<td>stall</td>
<td>stall</td>
<td>stall</td>
<td>stall</td>
<td>stall</td>
<td>stall</td>
<td>/M</td>
<td>-M/A</td>
<td>/M/A</td>
<td>/M</td>
<td>/M/A</td>
</tr>
<tr>
<td>SI^A</td>
<td>stall</td>
<td>stall</td>
<td>stall</td>
<td>stall</td>
<td>stall</td>
<td>stall</td>
<td>stall</td>
<td>/M</td>
<td>-M/A</td>
<td>/M/A</td>
<td>/M</td>
<td>/M/A</td>
</tr>
<tr>
<td>I^A</td>
<td>stall</td>
<td>stall</td>
<td>stall</td>
<td>stall</td>
<td>stall</td>
<td>stall</td>
<td>stall</td>
<td>/M</td>
<td>-M/A</td>
<td>/M/A</td>
<td>/M</td>
<td>/M/A</td>
</tr>
</tbody>
</table>

From: *A Primer on Memory Consistency and Cache Coherence*  
Daniel J. Sorin, Mark D. Hill, and David A. Wood
## SLICC original purpose

**Actual output**

<table>
<thead>
<tr>
<th>Load</th>
<th>Store</th>
<th>Replacement</th>
<th>FwdGetS</th>
<th>FwdGetM</th>
<th>Inv</th>
<th>PutAck</th>
<th>DataDirNoAcks</th>
<th>DataDirAcks</th>
<th>DataOwner</th>
<th>InvAck</th>
<th>LastInvAck</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>aAT gS pQ / IS D</td>
<td>aAT gM pQ / IM AD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>IS D</td>
<td>z</td>
<td>z</td>
<td>z</td>
<td>z</td>
<td></td>
<td></td>
<td>wd dT xLh pR / S</td>
<td></td>
<td></td>
<td></td>
<td>IS D</td>
</tr>
<tr>
<td>IM AD</td>
<td>z</td>
<td>z</td>
<td>z</td>
<td>z</td>
<td></td>
<td></td>
<td>wd dT xSh pR / M</td>
<td>wd sa pR / IM A</td>
<td></td>
<td></td>
<td>IM AD</td>
</tr>
<tr>
<td>IMA</td>
<td>z</td>
<td>z</td>
<td>z</td>
<td>z</td>
<td></td>
<td></td>
<td>da pR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>Lh pQ</td>
<td>aT gM pQ / SM AD</td>
<td>pS / SLA</td>
<td></td>
<td></td>
<td></td>
<td>iaR d pE / 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SM AD</td>
<td>Lh pQ</td>
<td>z</td>
<td>z</td>
<td>z</td>
<td></td>
<td></td>
<td>iaR pE / IM AD</td>
<td>wd dT xSh pR / M</td>
<td>wd sa pR / SM A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SMA</td>
<td>Lh pQ</td>
<td>z</td>
<td>z</td>
<td>z</td>
<td></td>
<td></td>
<td>da pR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>Lh pQ</td>
<td>Sh pQ</td>
<td>pM / MLA</td>
<td>cdR cdD pF / S</td>
<td>cdR d pF / 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MLA</td>
<td>z</td>
<td>z</td>
<td>z</td>
<td>cdR cdD pF / SLA</td>
<td>cdR pF / I LA</td>
<td>d pE / 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SLA</td>
<td>z</td>
<td>z</td>
<td>z</td>
<td>iaR pE / I LA</td>
<td>d pE / 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IIA</td>
<td>z</td>
<td>z</td>
<td>z</td>
<td></td>
<td></td>
<td></td>
<td>iaR d pE / I LA</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Jason Lowe-Power <jason@lowepower.com>
Examples

This is a very quick overview


Based on coherence protocols in Synthesis Lecture

*A Primer on Memory Consistency and Cache Coherence*

Daniel J. Sorin, Mark D. Hill, and David A. Wood
machine(MachineType:L1Cache, "MSI cache")
  : Sequencer *sequencer; // Incoming request from CPU come from this
  CacheMemory *cacheMemory; // This stores the data and cache states
  bool send_evictions; // Needed to support O3 CPU and mwait

  ...

  {
  ...
Important!

Never modify these files!

- L1Cache_Controller.py
- L1Cache_Controller.cc.hh
- L1Cache_Entry.cc.hh
- L1Cache_Transitions.cc.hh
- L1Cache_State.cc.hh
- L1Cache_Wakeup.cc.hh
- MSI-cache.sm

• SimObject “declaration file”
  - Inherits from AbstractController
• bool send_evictions
• Just a SimObject

Others…
Cache *state machine* outline

**Parameters:**

- **Cache memory:** Where the data is stored
- **Message buffers:** Sending/receiving messages from network

**State declarations:** The stable and transient states

**Event declarations:** State machine events that will be “triggered”

**Other structures and functions:** Entries, TBEs, get/setState, etc.

**In ports:** Trigger *events* based on incoming messages

**Actions:** Execute *single* operations on cache structures

**Transitions:** Move from *state to state* and execute *actions*
Cache memory

See src/mem/ruby/structures/CacheMemory

Stores the cache data (Entry) and the state (State)

cacheProbe() returns the replacement address if cache is full

Important!
Must call setMRU on each access!
Message buffers

Declaring is confusing!

MessageBuffer * requestToDir, network="To", virtual_network="0", vnet_type="request";
MessageBuffer * forwardFromDir, network="From", virtual_network="1", vnet_type="forward";

peek(): Get the head message
pop(): Remove head message (don’t forget this!)
isReady(): Is there a message?
recycle(): Move the head to the tail (better perf., but unrealisitic)
stallAndWait(): Move (stalled) message to different buffer
State declarations

state_declaration(State, desc="Cache states") {
    I, AccessPermission:Invalid, desc="Not present/Invalid";

    // States moving out of I
    IS_D, AccessPermission:Invalid, desc="Invalid, moving to S, waiting for data";
    IM_AD, AccessPermission:Invalid, desc="Invalid, moving to M, waiting for acks and data";
    IM_A, AccessPermission:Busy, desc="Invalid, moving to M, waiting for acks";

    S, AccessPermission:Read_Only, desc="Shared. Read-only, other caches may have the block";

    ...
}

AccessPermission: Used for functional accesses

IS_D -> Read: “Invalid transitioning to Shared waiting for Data”
Event declarations

eventation(Event, desc="Cache events") {
    // From the processor/sequencer/mandatory queue
    Load, desc="Load from processor";
    Store, desc="Store from processor";

    // Internal event (only triggered from processor requests)
    Replacement, desc="Triggered when block is chosen as victim";

    // Forwarded request from other cache via dir on the forward network
    FwdGetS, desc="Directory sent us a request to satisfy GetS. ";
        "We must have the block in M to respond to this.";
    FwdGetM, desc="Directory sent us a request to satisfy GetM. ";
    ...
Other structures and functions

**Entry**: Declare the data structure for each entry
Block data, block state, sometimes others (e.g., tokens)

**TBE/TBETable**: Transient Buffer Entry
Like an MSHR, but not exactly (allocated more often)
Holds data for blocks in *transient* states

get/set State, AccessPermissions, functional read/write
Required to implement AbstractController
Usually just copy-paste from examples
In ports

in_port(forward_in)

if (forward_in.isReady(clockEdge())) {
    peek(forward_in, RequestMsg) {
        Entry cache_entry := getCacheEntry(in_msg.addr);
        TBE tbe := TBEs[in_msg.addr];
        if (in_msg.Type == CoherenceRequestType:GetS) {
            trigger(Event:FwdGetS, in_msg.addr, cache_entry, tbe);
        } else
    }

    ...

Weird syntax!
Automatically populates “in_msg” in the following block

Trigger() looks for a transition. It also ensures resources available.
Like “peek”, but populates out_msg

Some variables are implicit in actions. These are passed in via trigger() in in_port. address, cache_entry, tbe
Transitions

transition(I, Store, IM_AD) {
    allocateCacheBlock;
    allocateTBE;
    sendGetM;
    popMandatoryQueue;
}

transition({IM_AD, SM_AD}, {DataDirNoAcks, DataOwner}, M) {
    writeDataToCache;
    deallocateTBE;
    externalStoreHit;
    popResponseQueue;
}
More details at

Ruby config scripts

Don’t follow gem5 style closely :(  

Require lots of boilerplate
Ruby config scripts

1. Instantiate the controllers
   Here is where you pass all of the options from the *.sm file

2. Create a Sequencer for each CPU
   More details in a moment

3. Create and connect all of the network routers
Creating the topology

Usually hidden in “create_topology” (see configs/topologies)

**Problem:** These make assumptions about controllers
Inappropriate for non-default protocols

Point-to-point example
self.routers = [Switch(router_id=i) for i in range(len(controllers))]

self.ext_links = [SimpleExtLink(link_id=i, ext_node=c, int_node=self.routers[i]) for i, c in enumerate(controllers)]

link_count = 0
self.int_links = []
for ri in self.routers:
    for rj in self.routers:
        if ri == rj: continue  # Don't connect a router to itself!
        link_count += 1
    self.int_links.append(SimpleIntLink(link_id=link_count, src_node=ri, dst_node=rj))
Ports -> Ruby interface

“Classic” ports

“Classic” ports

CPU
CPU
CPU
DMA
Other

DRAM Ctrl
DRAM Ctrl

RUBY

Jason Lowe-Power <jason@lowepower.com>
Ruby -> Memory

Any controller can connect its “memory” port. Usually, only “directory controllers.

You can send messages on this port in SLICC with queueMemoryRead/Write

Responses come on special message buffer (responseFromMemory)
CPU->Ruby: Sequencers

Confusing: Two names, same thing: RubyPort and Sequencer

Sequencer is a MemObject (classic ports)

Converts gem5 packets to RubyRequests

New messages delivered to the “MandatoryQueue”
Where is . . . ?

Configuration

- configs/network: Configuration of network models
- configs/topologies: Default cache topologies
- configs/ruby: Protocol config and Ruby config

Ruby config: configs/ruby/Ruby.py

Entry point for Ruby configs and helper functions
Selects the right protocol config “automatically”
Where is . . . ?

SLICC

csrc/mem/slicc
  Code for the compiler

csrc/mem/ruby/slicc_interface
  Structures used only in generated code
  AbstractController

Don’t be afraid to dig into the compiler! It’s often necessary.
Where is . . . ?

src/mem/ruby/structures
  Structures used in Ruby (e.g., cache memory, replace policy)

src/mem/ruby/system
  Ruby wrapper code and entry point
  RubyPort/Sequencer
  **RubySystem**: Centralized information, checkpointing, etc.
Where is . . . ?

<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>src/mem/ruby/common</td>
<td>General data structures, etc.</td>
</tr>
<tr>
<td>src/mem/ruby/filters</td>
<td>Bloom filters, etc.</td>
</tr>
<tr>
<td>src/mem/ruby/network</td>
<td>Network model (see next talk!)</td>
</tr>
<tr>
<td>src/mem/ruby/profiler</td>
<td>Profiling for coherence protocols</td>
</tr>
</tbody>
</table>
Current protocols (src/mem/protocol)

GPU rfo (Read for ownership GPU-CPU protocol)
GPU VIPER (“Realistic” GPU-CPU protocol)
GPU VIPER Region (HSA paper)
Garnet standalone (No coherence, just traffic injection)
MESI Three level (like two level, but with L0 cache)
MESI Two level (private L1s shared L2)
MI example (Example: Do not use for performance)
MOESI AMD (??)
MOESI CMP directory
MOESI CMP token
MOESI hammer (Like AMD hammer protocol for opteron/hyper transport)
Things not covered

Writing a coherence protocol
  Virtual networks
  Stalling requests
  Extra transient states

Debugging a coherence protocol
  RubyRandomTester + ProtocolTrace
  Other Ruby debug flags also useful
Many more details in the book:
learning.gem5.org/book
github.com/powerjg/learning_gem5

If you find a mistake:
Create an issue on github
Fork on github and fix it!
More resources

http://learning.gem5.org/book
http://gem5.org/SLICC
http://gem5.org/Ruby