Learning gem5

Jason Lowe-Power

http://learning.gem5.org/
What is gem5?

Michigan m5 + Wisconsin GEMS = gem5

“The gem5 simulator is a modular platform for computer-system architecture research, encompassing system-level architecture as well as processor microarchitecture.”

This tutorial

This is going to be interactive

Work along with me for best results

Ask questions!!
Schedule

Learning Part I 8:30 – 10:00
Break 10:00 – 10:30
Learning Part II 10:30 – 12:00
Lunch 12:00 – 1:30
Code Sprint I 1:30 – 3:00
Break 3:00 – 3:30
Code Sprint II 3:30 – 5:00

Learning Part I:
• Building gem5
• Config scripts
• gem5 output
• Simple SimObject

Learning Part II:
• Event-driven simulation
• SimObject parameters
• Memory system objects
• Other gem5 features

Code Sprint I:
• New gem5 contributions
[Andreas Sandberg]
• Begin sprints!

Code Sprint II:
• Continue sprints!
Building gem5

Let’s get started!

```bash
> git clone https://gem5.googlesource.com/public/gem5
> cd gem5
> git checkout hpca
> scons build/X86/gem5.opt -j5
```

and now we wait (about 8 minutes)
> scons build/X86/gem5.opt -j5

**scons**: the build system that gem5 uses (like make). See [http://scons.org/](http://scons.org/)

**build/X86/gem5.opt**: “parameter” passed to scons. gem5’s *Sconscript* interprets this. Also, the patch to the gem5 executable.

**X86**: Specifies the default build options. See `build_opts/*`

**opt**: version of executable to compile (one of debug, opt, perf, fast)
gem5 architecture

gem5 consists of “SimObjects”

Most C++ objects in gem5 inherit from class SimObject

Represent physical system components
gem5 architecture

gem5 is a **discrete event simulator**

**Event Queue**

1) Event at head dequeued
2) Event executed
3) More events queued

All SimObjects can enqueue events to the event queue

We’ll cover more after the break
gem5 configuration scripts

gem5 user interface

gem5 completely controlled by **Python scripts**

Scripts define system to model

All (C++) SimObjects exposed to Python

So... let’s make one!
Simple config script

Single CPU connected to a memory bus
Simple config script

Running gem5

> build/X86/gem5.opt
  configs/myscripts/simple.py

build/X86/gem5.opt: the gem5 binary to run
configs/.../simple.py: the configuration script (config script)
Port interface

system.cpu.icache_port = system.membus.slave
system.cpu.dcache_port = system.membus.slave
... 
system.mem_ctrl.port = system.membus.master

Ports connect MemObjects

To register a connection between master and slave, use ‘=’ in Python
Syscall Emulation (SE) mode

SE mode *emulates* the operating system (Linux) syscalls. No OS runs.

Full system mode runs a full OS as if gem5 is a “bare metal” system. Like full virtualization.

```python
process = LiveProcess()
process.cmd = ['tests/.../hello']
system.cpu.workload = process
...
root = Root(full_system = False)
```
Going further: Adding caches


Extending SimObjects in Python config

Object-oriented config files

Adding command-line parameters

2/5/2017

© Jason Lowe-Power <jason@lowepower.com>
It’s just Python!

```python
class L1Cache(Cache):
    ...

class L1ICache(L1Cache):
    def connectCPU(self, cpu):
        self.cpu_side = cpu.icache_port
    ...
```

Use good object-oriented design!

Debugging config files is easy. Just add some print statements!

Use Python builtins to provide support for command line parameters.

See text for details.
Understanding gem5 output

Understanding gem5 output

`> ls m5out`

- **config.ini**: Dumps all of the parameters of all SimObjects. This shows exactly what you simulated.

- **config.json**: Same as config.ini, but in json format.

- **stats.txt**: Detailed statistic output. Each SimObject defines and updates statistics. They are printed here at the end of simulation.
---------- Begin Simulation Statistics ----------

**sim_seconds**: 0.000346 # Number of seconds simulated

**sim_ticks**: 345518000 # Number of ticks simulated

**final_tick**: 345518000 # Number of ticks from beginning...

**sim_freq**: 1000000000000 # Frequency of simulated ticks

**...**

**sim_insts**: 5712 # Number of instructions simulated

**sim_ops**: 10314 # Number of ops (including micro...

**...**

**system.mem_ctrl.bytes_read::cpu.inst**: 58264 # Number of bytes

**system.mem_ctrl.bytes_read::cpu.data**: 7167 # Number of bytes...

**...**

**system.cpu.committedOps**: 10314 # Number of ops (including...

**system.cpu.num_int_alu_accesses**: 10205 # Number of integer...

---

*sim_seconds*: name of stat. This shows *simulated guest* time

Every SimObject can have its own stats. Names are what you used in the Python config file.
A simple SimObject

gem5’s coding guidelines

Follow the style guide (http://www.gem5.org/Coding_Style)
  Install the style guide when scons asks
  Don’t ignore style errors

Use good development practices
  Historically mercurial queues
  Now: git branches?
Adding a new SimObject

Step 1: Create a Python class

Step 2: Implement the C++

Step 3: Register the SimObject and C++ file

Step 4: (Re-)build gem5

Step 5: Create a config script
Step 1: Create a Python class

HelloObject.py

```python
from m5.params import *
from m5.SimObject import SimObject

class HelloObject(SimObject):
    type = 'HelloObject'
    cxx_header = 'learning_gem5/hello_object.hh'
```

**m5.params**: Things like MemorySize, Int, etc.

**type**: The C++ class name

**cxx_header**: The filename for the C++ header file
Step 2: Implement the C++

```cpp
#include "params/HelloObject.hh"
#include "sim/sim_object.hh"

class HelloObject : public SimObject
{
public:
  HelloObject(HelloObjectParams *p);
};
```

`params/*.hh` generated automatically. Comes from Python `SimObject` definition.

Constructor has one parameter, the generated `params` object.
Step 2: Implement the C++

**hello_object.cc**

```cpp
HelloObject::HelloObject(HelloObjectParams *params) : SimObject(params)
{
    std::cout << "Hello World! From a SimObject!" << std::endl;
}
HelloObject*
HelloObjectParams::create()
{
    return new HelloObject(this);
}
```

**HelloObjectParams**: when you specify a `Param` in the Hello.py file, it will be a member of this object.

You must **define** this function (you’ll get a linker error otherwise). This is how Python config creates the C++ object.
Step 3: Register the SimObject and C++ file

**SConscript**

- `Import(*)`
- `SimObject('Hello.py')`
- `Source('hello_object.cc')`

**Import**: SConscript is just Python... but weird.

**SimObject()**: Says that this Python file contains a SimObject. Note: you can put pretty much any Python in here.

**Source()**: Tell scons to compile this file (e.g., with g++).
Step 4: (Re-)build gem5
Step 5: Create a config script

```
<table>
<thead>
<tr>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>system.hello = HelloObject()</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
```

Instantiate the new object that you created in the config file (e.g., simple.py)

```
> build/X86/gem5.opt configs/learning_gem5/hello.py
...
Hello world! From a SimObject!
...
```
Simple SimObject code

http://learning.gem5.org/book/_downloads/HelloObject.py
Debug support in gem5

Adding debug flags

Declare the flag: add the debug flag to the SConscript file in the current directory

DebugFlag('Hello')

DPRINTF(Hello, "Created the hello object");

DPRINTF: macro for debug statements in gem5

Hello: the debug flag declared in the SConscript. Found in “debug/hello.hh”

Debug string: Any C format string
Debugging gem5

> build/X86/gem5.opt --debug-flags=Hello configs/learning_gem5/hello.py
...

0: system.hello: Hello world! From a debug statement

d debug-flags: Comma separated list of flags to enable. Other options include --debug-start=<tick>, --debug-ignore=<simobj name>, etc. See gem5.opt --help
Event-driven programming

Simple event callback

```cpp
class HelloObject : public SimObject {
private:
  ...
  void processEvent();
  EventWrapper<HelloObject, &HelloObject::processEvent> event;
public:
  HelloObject(HelloObjectParams *p);
  void startup();
};
```

**EventWrapper**: Convenience class for simple events with no parameters.

**processEvent**: Callback function to run when event fires.

**startup**: Called after all SimObjects instantiated. Schedule local events here.
Simple event callback

```cpp
void HelloObject::processEvent()
{
    timesLeft--;  
    DPRINTF(Hello, "Hello world!
"         " Processing the event! %d left\n", timesLeft);
    if (timesLeft <= 0) {
        DPRINTF(Hello, "Done firing!\n");
    } else {
        schedule(event, curTick() + latency);
    }
}
```

**schedule**: Put an event instance on the event queue. An absolute tick used for when the event is processed.

**curTick**: Returns the current simulator time. Useful for relative time computations.
Event SimObject code

SimObject parameters

Adding parameters

```python
class HelloObject(SimObject):
    type = 'HelloObject'
    cxx_header = "learning_gem5/hello_object.hh"

    time_to_wait = Param.Latency("Time before firing the event")
    number_of_fires = Param.Int(1, "Number of times to fire the event before "
                               "goodbye")
```

**Param.<TYPE>:** Specifies a parameter of type `<TYPE>` for the SimObject

**Param.<TYPE>():** First parameter: default value. Second parameter:
Going further: More parameters


Included types (e.g., MemorySize, MemoryBandwidth, Latency)

Using a SimObject as a parameter

SimObject-SimObject interaction
MemObjects

MemObject

Object that is part of gem5’s memory system
both classic caches and Ruby are MemObjects

Allowed to have MasterPorts and SlavePorts
Packets

Unit of transfer between MemObjects

Packets pass between Master and Slave ports

Packets have
  Request
  Command
  Data
  Much more...
Master and slave ports

**sendTimingReq**: send a Packet containing a request from a master to a slave

**recvTimingReq**: function that is called to handle the request in the slave port.

**return true**: The slave can handle the request.

Time

Master

Slave

sendTimingReq

recvTimingReq

returns: true

Slave executes request

recvTimingReq: function that is called to handle the request in the slave port.

return true: The slave can handle the request.
Master and slave ports

sendTimingReq: The slave finishes processing the request, and now sends a response (same packet).

recvTimingResp: Handles the response from the slave. Returning true means the packet is handled.
Master and slave ports

Master

Slave

sendTimingReq  recvTimingReq

returns: false

recvReqRetry  sendReqRetry

sendTimingReq  recvTimingReq

returns: true

return false: Slave cannot currently process the Packet. Resend the packet later. The Master’s responsibility to track Packet.

recvReqRetry: Can now retry the request by calling sendTimingReq.

sendReqRetry: Tell the master it can retry the stalled Packet.
Master and slave ports

Master
recvTimingResp sendTimingResp
returns: false
sendRespRetry recvRespRetry
recvTimingResp sendTimingResp
returns: true
Slave
return false: Master cannot currently process the Packet. Resend the packet later. The Slave's responsibility to track Packet.
sendRespRetry: Slave can now retry the response.
Master and slave port interface

**Master**
recv Timing Resp
recv Req Retry
recv Range Change

**Slave**
recv Timing Req
recv Resp Retry
recv Functional
get Addr Ranges
Simple MemObject

Switch!
Overview of SimpleMemobj
Going further: Implementing a cache

Full implementation of a uniprocessor cache (no coherence support)

Emulation concepts (using STL for associative data)

Vector ports, clocked objects, upgrading and creating packets

Other gem5 concepts

A random assortment
ISA support

gem5 supports many ISA

**ARM**: ARM Research often contributes and pretty stable

**x86**: Very complicated, there are a few bugs

**RISC-V**: Recent contribution. SE mode working. Thanks Alec!

**Alpha, MIPS, SPARC, Power**: No maintainers. Not very supported.
CPU Models

gem5 exposes a flexible CPU interface

**AtomicSimpleCPU**: No timing. Fast-forwarding & cache warming.

**TimingSimpleCPU**: Single-cycle (IPC=1) except for memory ops.

**O3CPU**: Out-of-order model. Highly configurable.

**MinorCPU**: In-order model (not fully tested with x86)

**kvmCPU**: x86 and ARM support for native execution
Full system support

Full system is like a virtual machine.

gem5 exposes a “bare metal” interface

Requires a kernel, disk image, etc.

Coherence modeling (Ruby)

“Classic cache” vs. Ruby

**Classic**: MOESI only, **Ruby**: any protocol

**Classic**: Hierarchical-snooping, **Ruby**: Snooping or directory or...

**Classic**: Flexible hierarchy, **Ruby**: Hierarchy baked into protocol

**Classic**: Good baseline CMP, **Ruby**: Required for coherence study

Ruby protocols in `src/mem/protocols`

- SLICC is a DSL for coherence
- Not much documentation, but many examples!
- Very flexible network simulation (Garnet)
GPU and device models

AMD recently released a HSAIL GPU model (src/gpu-compute)

Many devices supported for FS simulation
   Ethernet (and multi-system simulation)
   VNC for graphics
   IDE controllers for disks
   No Mali GPU for ARM
   VirtIO

Most devices are functional-only
Other features

Probes and tracing

Remote GDB

Dynamically-linked binaries in SE mode

Power modeling and PMU

And many, many, many others
Caveats

gem5 is a tool, not a panacea

Most models are not validated against “real” hardware

See “Architectural Simulators Considered Harmful”
https://doi.org/10.1109/MM.2015.74

There are bugs!
Getting (more) help

Main gem5 wiki: http://gem5.org/
My book:  
   http://learning.gem5.org  
   https://github.com/powerjg/learning_gem5
Mailing lists: http://gem5.org/Mailing_Lists  
   gem5-users: General user questions  
               (you probably want this one)  
   gem5-dev: Mostly code reviews and high-level  
             dev talk  
   gem5 QA (like StackOverflow): http://qa.gem5.org/
Learning gem5 Coding Sprint!

Jason Lowe-Power

http://learning.gem5.org/
Backup slides

Mostly from other versions of this tutorial
Adding caches

The previous example was boring

Let’s add some caches!

Switch!
from m5.params import *
...

class BaseCache(MemObject):
    type = 'BaseCache'
    abstract = True
    cxx_header = 'mem/cache/base.hh'

    size = Param.MemorySize("Capacity")
    assoc = Param.Unsigned("Associativity")
...
    tags = Param.BaseTags(LRU(), "Tag ...")
    ...
    cpu_side = SlavePort("Upstream port...")
    mem_side = MasterPort("Downstream...")
...

class Cache(BaseCache):
    type = 'Cache'
    cxx_header = 'mem/cache/cache.hh'
    ...
from m5.params import *
...

class BaseCache(MemObject):
    type = 'BaseCache'
    abstract = True
    cxx_header = 'mem/cache/base.hh'

    size = Param.MemorySize("Capacity")
    assoc = Param.Unsigned("Associativity")
    ...
    tags = Param.BaseTags(LRU(), "Tag ...")
    ...
    cpu_side = SlavePort("Upstream port...")
    mem_side = MasterPort("Downstream...")
    ...

class Cache(BaseCache):
    type = 'Cache'
    cxx_header = 'mem/cache/cache.hh'
    ...

Ports that connect the cache to the memory system.
Again, masters send requests, slaves receive requests.

These cannot have defaults. Must specify the sender/receiver in Python config files.
Cache scripts

http://learning.gem5.org/book/_downloads/two_level.py
Next time...

How to make changes to gem5
   Adding new SimObjects
   What’s in gem5

Let me know (powerjg@cs.wisc.edu) anything specific to cover!