Learning gem5 – Part I

Getting started with gem5

Jason Lowe-Power

http://learning.gem5.org/
https://faculty.engineering.ucdavis.edu/lowepower/
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.”

Tutorial and book are open source!

http://learning.gem5.org/

https://github.com/powerjg/learning_gem5

See a problem?
Open a pull request or issue

Want to add new material? Let me know!

Want to do your own version of this? See
http://learning.gem5.org/book/#notes-for-presentations
This tutorial

This is going to 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
Learning Part IV & III 1:30 – 3:30
Break 3:30 – 4:00
gem5 Best Practices 4:00 – 5:00
Open forum 5:00 – 5:30

Learning Part III:
• Intro to Ruby
• Simple MSI protocol
• Configuring Ruby
• Debugging Ruby

Learning Part IV:
• ISAs and CPU models
• Overview of gem5’s CPUs
• Building a simple CPU
• Bench gem5 for Memory Research

Learning Part III:
• Contributing to gem5
• Ryota: Visualizing the O3 CPU
• Éder: gem5 for Memory Research

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

Learning Part II:
• Event-driven simulation
• SimObject parameters
• Memory system objects
• Simple cache model
• Configuring gem5's CPUs
• Building a simple CPU
• Bench gem5 for Memory Research

© Jason Lowe-Power <jason@lowepower.com>
Building gem5

Let’s get started!

> git clone https://gem5.googlesource.com/public/gem5
> cd gem5
> git checkout -b asplos
> scons build/X86/gem5.opt -j5

and now we wait (about 8 minutes)
scons: the build system that gem5 uses (like make). See http://scons.org/

build/X86/gem5.opt: “parameter” passed to scons. gem5’s Sconsript 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

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

configs/learning_gem5/part1/simple.py
Running gem5

> build/X86/gem5.opt

configs/tutorial/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
...
```

To register a connection between master and slave, use `=` in Python

Ports connect MemObjects

Master → Requests → Slave → Responses
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.

```
process = Process()
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

© Jason Lowe-Power <jason@lowepower.com>
It’s just 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  config.json  stats.txt

**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
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
  system.mem_ctrl.bytes_read::cpu.data 7167 # Number of
...
  system.cpu.committedOps 10314 # Number of ops 
  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
Example scripts

Switch!
Questions?

We covered

- gem5 history
- Downloading and building gem5
- gem5’s user interface: python
- How to write a configuration script
- gem5’s output
- Using the example scripts