gem5 Tutorial
Getting started with gem5

Jason Lowe-Power & Bobby Bruce
jlowepower/bbruce@ucdavis.edu
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!

https://www.gem5.org/documentation/learning_gem5/introduction/
Source: https://gem5.googlesource.com/public/gem5-website/

See a problem?
Submit a change request or open an issue

Want to add new material? Let me know!

Want to do your own version of this? Let me know!
This tutorial

This is going to be interactive!

Trying something new with github codespaces and classroom

https://classroom.github.com/a/rV-jjuab

Work along with us for best results

Ask questions!!
Agenda

Introduction (8:30-8:45)

gem5 standard library (8:45-10:00)
  - Getting started with gem5
  - Understanding gem5 output
  - gem5 resources
  - Full system simulation
  - Extending the gem5 standard library

Developing with gem5 (10:00-10:30, coffee break, 11:00-11:30)
  - Building gem5
  - A simple SimObject
  - Debugging in gem5
  - Event-driven programming
  - Adding parameters

A bit of everything else (11:30-12:00)

The gem5 user’s workshop (1:30-5:00)
Introduction to gem5

What is “simulation” anyway?
Downloading/building gem5

```bash
> git clone https://gem5.googlesource.com/public/gem5
> cd gem5
> scons build/X86/gem5.opt -j<number of threads>
```
> git clone https://gem5.googlesource.com/public/gem5

**git**: Version control system

**googlesource**: Main gem5 repo location (not github, for now)

**stable**: The default branch for gem5. Updated at stable releases.

**develop** is updated more frequently (>1 per day)
> scons build/X86/gem5.opt -j17

**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, fast)
Let’s skip all that (for now)

https://classroom.github.com/a/rV-jjuab
From Computer Architecture Performance Evaluation Methods by Lieven Eeckhout

Computer architecture simulation!
Kinds of simulation

Functional simulation
  Executes programs correctly. Usually no timing information
  Used to validate correctness of compilers, etc.
  RISC-V Spike, QEMU, gem5 “atomic” mode

Instrumentation-based / Trace-based
  Often binary translation. Runs on actual hardware with callbacks
  *If execution depends on timing, this will not work!*
  PIN, CMP$im, NVBit

Execution-driven
  Functional and timing simulation is *combined*
  gem5 and many others
  gem5 is “execute in execute” or “timing directed”
Full system simulation

Components modeled with enough fidelity to run mostly unmodified apps

Often “Bare metal” simulation

All of the program is functionally emulated by the simulator

Often means running the OS in the simulator, not faking it

“Full system” simulators are often a combination of functional and full system
Nomenclature

**Host**: the actual hardware you’re using

Running things directly on the hardware:

**Native execution**

**Guest**: Code running on top of “fake” hardware

OS in virtual machine is guest OS

Running “on top of” hypervisor

Hypervisor is emulating hardware
Nomenclature

**Host:** the actual hardware you’re using

**Simulator:** Runs on the host
   Exposes hardware to the guest

**Guest:** Code running on *simulated* hardware
   OS running on gem5 is guest OS
   gem5 is simulating hardware

**Simulator’s code:** Runs natively
   executes/emulates the guest code

**Guest’s code:** (or benchmark, workload, etc.)
   Runs on gem5, not on the host.

---

**Your system**

- App
- Operating Sys.
- Hardware
- gem5/Simulator

---

**Diagram notes:**

- **Host:** The actual hardware you’re using.
- **Simulator:** Runs on the host and exposes hardware to the guest.
- **Guest:** Code running on simulated hardware (e.g., gem5).
- **Simulator’s code:** Runs natively and emulates the guest code.
- **Guest’s code:** (or benchmark, workload, etc.) Runs on gem5, not on the host.
Nomenclature

**Host**: the actual hardware you’re using

**Simulator**: Runs on the host
   Exposes hardware to the guest

**Simulator’s performance**: 
   Time to run the simulation on host
   Wallclock time as you perceive it

**Simulated performance**: 
   Time predicted by the simulator
   Time for guest code to run on simulator
gem5 architecture

gem5 consists of “SimObjects”

Most C++ objects in gem5 inherit from class SimObject

Represent physical system components
SimObject

Model
C++ code in src/

Parameters
Python code in src/
In SimObject declaration file

Instance or configuration
A particular choice for the parameters
In standard library, your extensions, or python runscript
gem5 architecture

gem5 is a **discrete event simulator**

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

**Event Queue**

- Event - 52
- Event - 50
- Event - 50
- Event - 20
- Event - 11
- Event - 10

All SimObjects can enqueue events to the event queue

We’ll cover more after the break
What we’ve learned

★ gem5 is a cycle-level full-system execution-driven simulator
★ To obtain gem5, you need to download the source with git
★ We’ll be using codespaces for this tutorial

Next up
★ How to configure and run gem5 simulations with the standard library