The gem5 architecture simulator

A tutorial for HPCA ‘23, presented by the Bobby R. Bruce
Today’s Agenda

Introduction (8:30 to 9:00)
• What is gem5?
• What can gem5 be used for?
• Nomenclature
• Obtaining and building gem5

“Hello World” in gem5 (9:00 to 9:30)
• Running a “Hello World” binary in SE Mode
• How does gem5 work?
  • Discrete Event Simulation
  • SimObjects

gem5 standard library (9:30 to 10:50)
• What is it?
• Where is it?
• Gem5 Resources
• Coffee Break at somepoint.
• Understanding the stats.
• Creating a traffic generator.
• Creating a FS simulation.
  • The Simulator Module

Speeding things up (10:50 to 11:20)
• What can we do?
• Checkpoints
• Simpoints and Looppoints

The GPU Model (11:20 to 12:05)

Wrap-up (12:05-12:15)

Tutorial on Looppoints in the afternoon (Outremont 1)

WIFI PW: hotel 900
GitHub Codespaces

https://github.com/gem5-hpca-2023/gem5-tutorial-codespace

“Code” -> “Codespaces” -> “Create Codespace on master”

This will take a minute to load a Visual Studio Code environment in your web-browser.
GitHub Codespaces

The “ALL/gem5.fast” binary comes pre-built and installed as “gem5”

“materials” contains everything you need.

This is completely open source. Feel free to pull a copy.
What is gem5?

The gem5 architecture simulator provides a platform for evaluating computer systems by modeling the behavior of the underlying hardware. It enables researchers to simulate the performance and behavior of complex computer systems, including the CPU, memory system, and interconnects. This makes it possible to study the performance of different microarchitectural and architectural choices, as well as the effects of different workloads, without having to build and test real systems.

By ChatGPT
A little bit of history

The m5 Simulator
“A tool for simulating systems”
(~2002)

The GEMS simulator
Provided a detailed memory system.
(~2000)
A true public infrastructure project

- Open Source
- Free (like beer)
- Massively Collaborative
Who uses gem5, and why?

- Education
- Academic Research and public research labs
- Industrial R&D
Problem: Students need to learn to design hardware but don’t have a multi-billion-dollar factory.
Research

(a) Scientific research
- Take hypothesis about environment
- Design experiment
  - Pick a random sample from a population
- Run experiment and quantify
- Interpret results
- If necessary, create new hypothesis

(b) Systems research
- Take hypothesis about environment
- Design experiment
  - Pick baseline design and workloads
- Run experiment and quantify
  - Run model or measure hardware
- Interpret results
- If necessary, propose new design
Researchers

We recently surveyed the top architecture conferences and found:

- 70% of all computer architecture research utilizes simulation.
- The gem5 simulator is by-far the most popular.

Room for improvement: Most users still “roll their own” simulation software. Only 20% use gem5 directly. We want to go above 50% by 2027.
Really, we don’t know exactly. We don’t track users and industrial users seldom make themselves known.
Industry

Big players we know use it
What languages do we use?

Your simulation configuration is written in Python which interacts with the core CPP simulator.

In this tutorial we’ll be working solely at the level of Python.

Adding CPP code is necessary for extending gem5’s capabilities.
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.
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

Your system

- App
- Operating Sys.
- Hardware
- App
- Operating Sys.
- gem5
- Hardware
Let’s hit the ground running

This example will show:

1. How someone obtains gem5.
2. How you build it.
3. Running a very basic “Hello World” simulation.
Downloading/building gem5

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

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

**develop** is updated more frequently (>1 per day)
Write a “hello world!” configuration (in Python!)

Open “materials/hello-world.py”.

We have provided the imports:

```python
1 from gem5.components.boards.simple_board import SimpleBoard
2 from gem5.components.cachehierarchies.classic.no_cache import NoCache
3 from gem5.components.memory import SingleChannelDDR3_1600
4 from gem5.components.processors.simple_processor import SimpleProcessor
5 from gem5.components.processors.cpu_types import CPUPetypes
6 from gem5.resources.resource import obtain_resource
7 from gem5.simulate.simulator import Simulator
8 from gem5.isas import ISA
```
“hello world!”: Obtaining the components

```python
10    # Obtain the components.
11    cache_hierarchy = NoCache()
12    memory = SingleChannelDDR3_1600("1GiB")
13    processor = SimpleProcessor(cpu_type=CPUTypes.ATOMIC, num_cores=1, isa=ISA.X86)
```
“hello world!”: Adding to the board!

```python
# Add them to the board.
board = SimpleBoard(
    clk_freq="3GHz",
    processor=processor,
    memory=memory,
    cache_hierarchy=cache_hierarchy,
)
```
“hello world!”: Obtain the resource

```python
# Obtain a binary to run via gem5-resources.
binary = obtain_resource("x86-hello64-static")
board.set_se_binary_workload(binary)
```
“hello world!”: Load the board to the simulator

```
26 # Setup the Simulator and run the simulation.
27 simulator = Simulator(board=board)
28 simulator.run()
```
“hello world!”: In full

```python
from gem5.components.boards.simple_board import SimpleBoard
from gem5.components.cachehierarchies.classic.no_cache import NoCache
from gem5.components.memory.single_channel import SingleChannelDDR3_1600
from gem5.components.processors.simple_processor import SimpleProcessor
from gem5.components.processors.cpu_types import CPUType
from gem5.resources.resource import Resource
from gem5.simulate.simulator import Simulator

# Obtain the components.
cache_hierarchy = NoCache()
memory = SingleChannelDDR3_1600("1GiB")
processor = SimpleProcessor(cpu_type=CPUType.ATOMIC, num_cores=1)

# Add them to the board.
board = SimpleBoard(
    clk_freq="3GHz",
    processor=processor,
    memory=memory,
    cache_hierarchy=cache_hierarchy,
)

# Set the workload.
binary = Resource("x86-hello64-static")
board.set_se_binary_workload(binary)

# Setup the Simulator and run the simulation.
simulator = Simulator(board=board)
simulator.run()
```

A full example can be found in “materials/complete/hello-world.py”
“hello world”: Let’s run it!

```
> gem5 materials/hello-world.py
```

Resource 'x86-hello64-static' was not found locally. Downloading to '/home/bbruce/.cache/gem5/x86-hello64-static'...
Finished downloading resource 'x86-hello64-static'.

warn: The simulate package is still in a beta state. The gem5 project does not guarantee the APIs within this package will remain consistent across upcoming releases.

Global frequency set at 1000000000000 ticks per second

build/X86/mem/mem_interface.cc:791: warn: DRAM device capacity (8192 Mbytes) does not match the address range assigned (1024 Mbytes)

0: board.remote_gdb: listening for remote gdb on port 7001

build/X86/sim/simulate.cc:194: info: Entering event queue @ 0. Starting simulation...

build/X86/sim/syscall_emul.hh:1014: warn: readlink() called on '/proc/self/exe' may yield unexpected results in various settings.

Returning '/scr/bbruce/.cache/gem5/x86-hello64-static'


Hello world!
Wait, what just happened?

- We just did a “Hello World!”
- 1-Core, No Cache Hierarchy, 1GiB memory
- Using “SE Mode”
- Demonstrated the core library APIs
Ok, but how does it work?

Modern systems are very complex, and the design of gem5 simulations reflects this.

However, at its core, the simulator builds on a relatively simple model.
At its core: it’s a discrete event simulator

gem5 is a **discrete event simulator**

Event Queue

1) Event at head dequeued
2) Event executed
3) More events queued
At its core: it’s a discrete event simulator

gem5 is a **discrete event simulator**

1) Event at head dequeued
2) Event executed
3) More events queued
Discrete event simulation example

1. Fetch first inst
2. Send req to cache
3. Miss in L1, send to DRAM
4. Put in read Q
5. Get data from DRAM
6. Cache recvs data
7. Processor decodes instruction
8. Processor executes instruction
9. DRAM read latency
10. Response latency
11. One cycle
12. L1 tag latency
13. To DRAM latency
14. Fetch next inst
Discrete event simulation

"Time" needs a unit
In gem5, we use a unit called "Tick"

Need to convert a simulation "tick" to user-understandable time
E.g., seconds

This is the global simulation tick rate
Usually this is $1\text{ ps per tick}$ or $10^{12}$ ticks per second
Ok, but how do you schedule these events?

While some are incredibly complex, at their core they only do two things:

1. Schedule events and process events.
2. Talk to other SimObjects.
It’s hard to scale this...

When done without the library you must define every part of your simulation.

This allows for maximum flexibility but can mean creating 100s of lines of Python to create even a basic simulation.
The solution: The gem5 standard library

The stdlib is a library which allows for users to quickly create systems with pre-built components.

The stdlib's module architecture allows for components (e.g. a memory system or a cache hierarchy setup) to be quickly swapped in and out without radical redesign.
The stdlib modular metaphor

Processor

Memory

SingleChannelID DR3_1600
SingleChannelID DR4_2400

Simple Processor
SwitchableProcessor

Cache Hierarchy

No Cache
PrivateL1 PrivateL2
MesiTwoLevel
The modular architecture
Where to find stuff: The directory structure
Where to find stuff: Importing in a script

```python
from gem5.components.boards.simple_board import SimpleBoard
from gem5.components.cachehierarchies.classic.no_cache import NoCache
from gem5.components.memory.single_channel import SingleChannelDDR3_1600
from gem5.components.processors.simple_processor import SimpleProcessor
from gem5.components.processors.cpu_types import CPUTypes
from gem5.resources.resource import Resource
from gem5.simulate.simulator import Simulator
```
Let’s go back and loop at our script again

```python
from gem5.components.boards.simple_board import SimpleBoard
from gem5.components.cachehierarchies.classic.no_cache import NoCache
from gem5.components.memory.single_channel import SingleChannelDDR3_1600
from gem5.components.processors.simple_processor import SimpleProcessor
from gem5.components.processors.cpu_types import CPUTypes
from gem5.resources.resource import Resource
from gem5.simulate.simulator import Simulator

# Obtain the components.
cache_hierarchy = NoCache()
memory = SingleChannelDDR3_1600("1GiB")
processor = SimpleProcessor(cpu_type=CPUTypes.ATOMIC, num_cores=1)

# Add them to the board.
board = SimpleBoard(
    clk_freq="3GHz",
    processor=processor,
    memory=memory,
    cache_hierarchy=cache_hierarchy,
)

# Set the workload.
binary = Resource("x86-hello64-static")
board.set_se_binary_workload(binary)

# Setup the Simulator and run the simulation.
simulator = Simulator(board=board)
simulator.run()
```
gem5 Resources

• gem5 resources is a repository providing sources for artifacts that are known to be compatible with gem5.

• These resources are not necessary for the compilation or running gem5 but may aid users in running simulations. E.g.: disk images, kernels, applications, cross-compilers, etc.

• Resources are held on gem5's Google Cloud Bucket, and sources for these resources are found at: https://gem5.googlesource.com/public/gem5-resources/

• The stdlib can be used to automatically obtain and use these resources.

• https://resources.gem5.org/resources.json
Looking up gem5 Resources

https://resources.gem5.org/resources.json

This is all machine-reachable for now. We're working on a web-portal.
Obtaining Resources in the stdlib

This complete script can be found in “materials/obtain-resources.py”

```python
from gem5.resources.resource import obtain_resource

resource = obtain_resource("riscv-disk-img")

print(f"The resource is available at {resource.get_local_path()}" )
```

> gem5 materials/obtain-resources.py
Obtaining Resources in the stdlib

The stdlib will use the cached resources if already downloaded.
Using a Custom Resource

You don't need to use the gem5 resources

You can specify a local resources (e.g., your own disk image)

```python
from gem5.resources.resource import CustomResource

CustomResource("tests/test-progs/hello/bin/x86/linux/hello")
```
More detailed output

Look into the more the “gem5/m5out” directory

- The “config” files detail your system configuration (various formats, “config.ini” most human-readable.

- The stats.txt shows the various simulation statistics.

- In Full-System simulations the terminal output can be found in this directory.
More detailed output

Look into the more the “gem5/m5out/stats.txt” file

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>simSeconds</td>
<td>0.000005</td>
</tr>
<tr>
<td>simTicks</td>
<td>4979349</td>
</tr>
<tr>
<td>finalTick</td>
<td>4979349</td>
</tr>
<tr>
<td>simFreq</td>
<td>1000000000000</td>
</tr>
<tr>
<td>hostSeconds</td>
<td>0.08</td>
</tr>
<tr>
<td>hostTickRate</td>
<td>64410071</td>
</tr>
<tr>
<td>hostMemory</td>
<td>1169600</td>
</tr>
<tr>
<td>simInsts</td>
<td>6546</td>
</tr>
<tr>
<td>simOps</td>
<td>12944</td>
</tr>
<tr>
<td>hostInstRate</td>
<td>84513</td>
</tr>
<tr>
<td>hostOpRate</td>
<td>167067</td>
</tr>
</tbody>
</table>
Modifying our design!

Remember: gem5 is modular!

In general, you can replace components with components of the same type.

Let’s convert our basic “hello world” board into a Traffic generator board.
Traffic Generator

Generator -> Memory System
The ‘TestBoard’

Generator

TestBoard

Memory System

And gem5 does the rest!
Let’s build one!

Go to “materials/traffic-generator.py”

Here we have imports, and some boilerplate code to run the simulation. You’re code will go in-between.
Traffic generator: Setup our components

```python
9 # Setup the components.
10 memory = SingleChannelDDR3_1600("1GiB")
11 generator = RandomGenerator(
12    duration="250us",
13    rate="40GB/s",
14    num_cores=1,
15    max_addr=memory.get_size(),
16 )
17 cache_hierarchy = NoCache()
```
Traffic generator: Connect them to the TestBoard

```python
19  # Add them to the Test board.
20  board = TestBoard(
21      clk_freq="3GHz",
22      generator=generator,
23      memory=memory,
24      cache_hierarchy=cache_hierarchy,
25  )
```
Running the traffic generator.

> gem5 materials/traffic-generator.py

This should be quite fast. What have we done?

- Generated traffic to evaluate a memory component
- This does not require a workload or even a real processor
- Users would typically consult the stats.txt
Let’s add a ‘fun’ memory system

```
from gem5.components.memory import HBM2Stack
```

Complete version can be found in “materials/complete/traffic-generator-hbm2stack.py”
More complex designs: An X86 full system simulation in the stdlib

Move to "materials/x86-full-system.py. You should see the following provided for you:

```python
from gem5.utils.requires import requires
from gem5.components.boards.x86_board import X86Board
from gem5.components.memory.single_channel import SingleChannelDDR3_1600
from gem5.components.cachehierarchies.ruby.mesi_two_level_cache_hierarchy import (MESITwoLevelCacheHierarchy,
)
from gem5.components.processors.simple_switchable_processor import (SimpleSwitchableProcessor,
)
from gem5.coherence_protocol import CoherenceProtocol
from gem5.isas import ISA
from gem5.components.processors.cpu_types import CPUTypes
from gem5.resources.workload import Workload
from gem5.simulate.simulator import Simulator
from gem5.simulate.exit_event import ExitEvent
```
Adding the 'requires' function

```python
15 requires(
16     isa_required=ISA.X86,
17     coherence_protocol_required=CoherenceProtocol.MESI_TWO_LEVEL,
18 )
```

This adds a check for the gem5 binary parsing the script. In this case:

1. The binary supports the X86 ISA.
2. The binary supports the MESI Two Level coherence protocol.
Extending the gem5 library

```
21 cache_hierarchy = MESITwoLevelCacheHierarchy(
22   l1d_size="32KiB",
23   l1d_assoc=8,
24   l1i_size="32KiB",
25   l1i_assoc=8,
26   l2_size="256kB",
27   l2_assoc=16,
28   num_l2_banks=1,
29 )
```

```
35 memory = SingleChannelDDR3_1600("2GiB")
```
Extending the gem5 library

The SimpleSwitchingProcessor allows for different types of cores to be swapped during a simulation with `processor.switch()`.

This can be useful when wanting to switch to and from a detailed form of simulation. (Timing = less detailed but fast; O3 = detailed but slow).
Extending the gem5 library

As usual, we add the components to the board, in this case an `X86Board`.

```python
board = X86Board(
    clk_freq="3GHz",
    processor=processor,
    memory=memory,
    cache_hierarchy=cache_hierarchy,
)
```
Extending the gem5 library

```python
workload = Workload("x86-ubuntu-18.04-boot")
```
Extending the gem5 library

```
"resources": [ 
  {
    "type": "workload",
    "name": "x86-ubuntu-18.04-boot",
    "documentation": "A full boot of Ubuntu 18.04 with Linux 5.4.49 for X86. It runs an `m5 exit` command when the boot is completed unless the readfile is specified. If specified the readfile will be executed after booting.",
    "function": "set_kernel_disk_workload",
    "resources": {
      "kernel": "x86-linux-kernel-5.4.49",
      "disk_image": "x86-ubuntu-18.04-img"
    },
    "additional_params": {}
  },
  
http://resources.gem5.org/resources.json
```
Extending the gem5 library

```python
command = (  
    "m5 exit;"
+  "echo 'This is running on 03 CPU cores.';"
+  "sleep 1;"
+  "m5 exit;"
)  

workload.set_parameter("readfile_contents", command)
```
Extending the gem5 library

```
board.set_workload(workload)
```
Exit Events

**Note:** This module is still considered to be in Beta. The API may change in future versions of gem5.

During a simulation you can have "Exit Events".

In this example there are two. These return the simulation to the Python Script.
The Simulation Loop

"simulate.run()"

X

"simulate.run()"

Simulation

Exit Event

Simulation

Exit Event
The Simulator Module handles the loop!

```python
31 simulator = Simulator(board=board)
32 simulator.run() # Runs up to the first `m5 exit` event
33 # Here we can do things between the exit event
34 processor.switch()
35 simulator.run() # Run up to the final `m5 exit` event
```

Here we can run up to an exit event, do things, and then continue the run.

In this case we want to switch the CPU cores.
The Simulator Module: We can do better

Here we can specify exactly what to do on each exit event type via Python generators.

The Simulator had default behavior for these events, but they can be overridden.

- ExitEvent.EXIT
- ExitEvent.CHECKPOINT
- ExitEvent.FAIL
- ExitEvent.SWITCHCPU
- ExitEvent.WORKBEGIN
- ExitEvent.WORKEND
- ExitEvent.USER_INTERRUPT
- ExitEvent.MAX_TICK
Your done! You can now run your full-system simulation

**Warning:** This will take a long time to complete execution.
Simulation’s major pitfall: It’s slooooww

Simulating 1 Second

>> 100k seconds on the host
Fortunately, there are some workarounds

Key idea: You don’t need to simulate everything perfectly, or at all.
Some techniques we provide

- CPU Models
- KVM Mode
- SE Mode
- Checkpoints
- Simpoints
Simpoints/Looppoints

There’s a tutorial on Looppoints this afternoon in Outremont 1

Discuss: What is a Simpointing?
Simpoints

Open "materials/simpoints-checkpoint.py"
# Setup the Simpoints workload
board.set_se_simpoint_workload(
    binary=obtain_resource("x86-print-this"),
    arguments=['print this', 15000],
    simpoint=SimpointResource(
        simpoint_interval=1000000,
        simpoint_list=[2, 3, 4, 15],
        weight_list=[0.1, 0.2, 0.4, 0.3],
        warmup_interval=1000000,
    ),
)
dir = Path("simpoint-checkpoint-dir")
dir.mkdir(exist_ok=True)

# Here we use the Simpoints generator to take the checkpoints.
# When a Simpoint region, or warmup region, begins, a checkpoint is generated.
simulator = Simulator(
    board=board,
    on_exit_event={ExitEvent.SIMPOINT_BEGIN: save_checkpoint_generator(dir)},
)

Simpoints

`> gem5 materials/simpoint-checkpoint.py`

Generates checkpoints at each warmup/Simpoint region start.
Simpoints: Restoring

> gem5 materials/simpoints-restore.py
The gem5 GPU Model
Caveats

gem5 is a tool, not a panacea

Most models are not validated against “real” hardware

See “Architectural Simulators Considered Harmful”
There are bugs!
Getting (more) help

Main gem5 website: http://gem5.org/
More like this:
https://www.gem5.org/documentation/
learning_gem5/introduction/

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 slack: https://join.slack.com/t/gem5-workspace/shared_invite/zt-1c8go4yjo-LNb7I~BZ0FagwmVxX08y9g
References