Learning gem5 Simulator

Karunakar R. Basireddy
krb1g15@ecs.soton.ac.uk
Electronics and Computer Science,
University of Southampton, UK
15 Feb. 2018
Outline

- Architectural simulator
- Why gem5?
- What is gem5?
- What can it do?
  - key features
- Getting started with gem5
  - Building gem5
  - compiling an application for SE and running
  - Understanding gem5 output
Architectural simulator

- **Architectural simulator**: is a software to model computer system/components to predict outputs and performance metrics on a given input (e.g. applications).

- The simulation could be targeting:
  - Specific components (e.g. Caches, DRAM, etc.)
  - Bare-metal (microprocessor)
  - Full System (processor cores, peripheral devices, memories, interconnection buses, etc. with OS)
Classification of simulators

- Functional - models functional characteristics of an ISA without providing any timing values, e.g. sim-safe
- Instrumentation-based - simulates the same target architecture of the host architecture. e.g. Intel’s pin tool
- Full system (trace- and execution-driven) - simulate the entire computer system (application and OS). e.g. gem5.
- Full trace-driven and execution-driven can be further divided into – cycle accurate and timing approximate
Why gem5?

- There are several other simulators which can do similar things but gem5 is complete with different aspects of architectural research covering aspects of:
  - Memory
  - CPU
  - Interconnect
  - Runtime optimization techniques:
    - DVFS, task mapping etc.

- Heavily used by many industrial/academic researchers for understanding architectural optimization or runtime management opportunities
  - ARM, Intel, AMD, HP and many other hardware/IP vendors use gem5
## Simulators comparison

<table>
<thead>
<tr>
<th>Simulator</th>
<th>ISAs Supported</th>
<th>Accuracy</th>
<th>License</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flexus</td>
<td>SPARC, x86 (requires simics)</td>
<td>Cycle-accurate</td>
<td>BSD</td>
</tr>
<tr>
<td>Simics</td>
<td>Alpha, ARM, MIPS, PowerPC, SPARC and x86</td>
<td>functional</td>
<td>Private</td>
</tr>
<tr>
<td>GEMS</td>
<td>SPARC (requires simics)</td>
<td>Timing</td>
<td>GPL</td>
</tr>
<tr>
<td>MARSS</td>
<td>X86 (requires QEMU)</td>
<td>Cycle-accurate</td>
<td>GPL</td>
</tr>
<tr>
<td>PTLsim</td>
<td>X86 (requires Xen and KVM/QEMU)</td>
<td>Cycle-accurate</td>
<td>GPL</td>
</tr>
<tr>
<td>Gem5</td>
<td>Alpha, ARM, MIPS, POWER, SPARC, X86</td>
<td>Cycle-accurate</td>
<td>BSD</td>
</tr>
<tr>
<td>VLAB ARM Tool Box</td>
<td>ARM</td>
<td>Timing approximate</td>
<td>Private</td>
</tr>
<tr>
<td>OVPsim</td>
<td>ARM, MIPS, X86</td>
<td>Functional</td>
<td>Open and private</td>
</tr>
</tbody>
</table>
What is gem5?

Michigan m5 + Wisconsin GEMS = gem5

• M5: CPU models, ISAs, I/O devices, infrastructure

• GEMS (essentially Ruby): cache coherence protocols, interconnect models

• gem5 is a **modular** platform for computer-system **architecture** research, including **system-level** architecture as well as **processor microarchitecture**

DOI=http://dx.doi.org/10.1145/2024716.2024718
What is gem5? (cont.)

- **Modular** – allows to change different modules (CPU, memory, interconnection)
- **Architecture** – supports connecting modules in various sizes and hierarchy
- **System level** – Capable of simulating HW/SW in the same platform
- **Microarchitecture** – modules can be refined with your own design
What can it do?

Almost everything related to computer architecture research

• Early architectural exploration:
  o Finding best configuration of a CPU
  o cache capacity, associativity, coherence protocols, etc.
  o best cache hierarchy (e.g. number of levels)

• HW/SW related design space exploration
  o Investigating power-performance trade-offs
  o Evaluation of application/system software performance

• Much more .............
Key features

- Homogeneous and heterogeneous multi-core
- Multiple CPU models: AtomicSimple, TimingSimple, InOrder, and O3
- Multiple ISA support: Alpha, ARM, SPARC, MIPS, POWER and x86.
- Two execution modes: System-call Emulation & Full-system
- Two memory system models: Classic & Ruby
- Cache coherence protocols: broadcast-based, directories, etc.
- Interconnection networks: Simple & Garnet (Princeton, MIT)
Key features (cont.)

• Pervasive object-oriented design
  – Provides modularity, flexibility
  – Significantly leverages inheritance e.g. SimObject

• Python integration
  – Powerful front-end interface
  – Provides initialization, configuration, & simulation control

• Domain-Specific Languages
  – ISA DSL: defines ISA semantics
  – Cache Coherence DSL (a.k.a. SLICC): defines coherence logic
  – Standard interfaces: Ports and MessageBuffers
Getting started with gem5

• A full set of instructions is available on the website: [http://gem5.org/Documentation](http://gem5.org/Documentation)

• gem5 runs in a Unix/Linux environment (MacOS is Unix). If you're running Windows, you will need to install a virtual Linux machine.

• VirtualBox is a free virtualisation tool, available from: [https://www.virtualbox.org/](https://www.virtualbox.org/) It runs on Windows, Linux and MacOS.
Getting started with gem5 (cont.)

• You will then need to create a virtual machine. Download the (64-bit) Ubuntu Desktop installation disk image from: [http://www.ubuntu.com/download/download/desktop](http://www.ubuntu.com/download/download/desktop). Create a new virtual machine with at least ~20GB of disk space and boot it from the Ubuntu installation disk image. Follow the installation instructions.
building gem5 (ARM)

- [https://www.youtube.com/watch?v=SW63HJOnW90](https://www.youtube.com/watch?v=SW63HJOnW90)

$ sudo apt-get update; sudo apt-get upgrade
$ sudo apt-get install mercurial scons swig gcc m4 python python-dev libgoogle-perftools-dev g++
$ hg clone http://repo.gem5.org/gem5
$ cd gem5/
$ scons build/ARM/gem5.opt -j2

- `build/ARM/gem5.opt` includes basic optimizations and debugging facilities – good enough for our work
Sample compile (x86)

```bash
blue% scons build/X86_FS/gem5.opt
scons: Reading SConscript files ...
Checking for leading underscore in global variables... no
Checking for C header file Python.h... yes
Checking for C library pthread... yes
<nship...>
Reading /n/blue/z/binkert/work/m5/incoming/src/mem/ruby/SConsopts
Reading /n/blue/z/binkert/work/m5/incoming/src/mem/protocol/SConsopts
Reading /n/blue/z/binkert/work/m5/incoming/src/arch/arm/SConsopts
<nship...>
Building in /n/blue/z/binkert/work/m5/incoming/build/X86_FS
Variables file /n/blue/z/binkert/work/m5/incoming/build/variables/X86_FS not found,
using defaults in /n/blue/z/binkert/work/m5/incoming/build_opts/X86_FS
scons: done reading SConscript files.
scons: Building targets ...
  [ CXX] X86_FS/sim/main.cc -> .o
  [ CXX] X86_FS/sim/async.cc -> .o
  [ CXX] X86_FS/sim/core.cc -> .o
  [ TRACING] -> X86_FS/debug/Event.hh
Defining FAST_ALLOC_STATS as 0 in build/X86_FS/config/fast_alloc_stats.hh.
Defining FORCE_FAST_ALLOC as 0 in build/X86_FS/config/force_fast_alloc.hh.
Defining NO_FAST_ALLOC as 0 in build/X86_FS/config/no_fast_alloc.hh.
  [ CXX] X86_FS/sim/debug.cc -> .o
  [ TRACING] -> X86_FS/debug/Config.hh
  [ CXX] X86_FS/sim/eventq.cc -> .o
  [ CXX] X86_FS/sim/init.cc -> .o
  [ TRACING] -> X86_FS/debug/TimeSync.hh
[SO PARAM] Root -> X86_FS/params/Root.hh
[SO PARAM] SimObject -> X86_FS/params/SimObject.hh
...
Running gem5 binary

```bash
maize% ./build/ARM_FS/gem5.opt --help
Usage
=======
    gem5.opt [gem5 options] script.py [script options]

gem5 is copyrighted software; use the --copyright option for details.

Options
=======
--version              show program’s version number and exit
--help, -h             show this help message and exit
--build-info, -B       Show build information
--copyright, -C        Show full copyright information
--readme, -R           Show the readme
--outdir=DIR, -d DIR   Set the output directory to DIR [Default: /tmp/m5out]
--redirect-stdout, -r  Redirect stdout (& stderr, without -e) to file
--redirect-stderr, -e  Redirect stderr to file
--stdout-file=FILE     Filename for -r redirection [Default: simout]
--stderr-file=FILE     Filename for -e redirection [Default: simerr]
--interactive, -i      Invoke the interactive interpreter after running the script
--pdb                  Invoke the python debugger before running the script
--path=PATH[:PATH], -p PATH[:PATH]
                        Prepend PATH to the system path when invoking the script
--quiet, -q            Reduce verbosity
--verbose, -v          Increase verbosity
```
Running gem5 binary (cont.)

Statistics Options
------------------
--stats-file=FILE  Sets the output file for statistics [Default: stats.txt]

Configuration Options
---------------------
--dump-config=FILE  Dump configuration output file [Default: config.ini]

Debugging Options
------------------
--debug-break=TIME[,TIME]  Cycle to create a breakpoint
--debug-help              Print help on trace flags
--debug-flags=FLAG[,FLAG]  Sets the flags for tracing (-FLAG disables a flag)
--remote-gdb-port=REMOTE_GDB_PORT  Remote gdb base port (set to 0 to disable listening)

Trace Options
------------
--trace-start=TIME       Start tracing at TIME (must be in ticks)
--trace-file=FILE         Sets the output file for tracing [Default: cout]
--trace-ignore=EXPR      Ignore EXPR sim objects

Help Options
-----------
--list-sim-objects       List all built-in SimObjects, their params and default values
Get a cross-compile tool chain

Installing the tool chain:

```
$ sudo apt-get install arm-linux-gnueabi-gcc
```

Compiling the application:

```
$arm-linux-gnueabi-gcc -DUNIX tests/test-progs/hello/src/hello.c -static -o tests/test-progs/hello/bin/arm/linux/hello
```
gem5 user-interface

- gem5 completely controlled by Python scripts
- Scripts define system to model
- All (C++) SimObjects exposed to Python
Running simulations

• We will use Syscall-emulation (SE)

$build/ARM/gem5.opt configs/example/se.py -c tests/test-progs/hello/bin/arm/linux/hello


gem5 is copyrighted software; use the --copyright option for details.

gem5 compiled Jun 2 2011 17:39:30

gem5 started Jun 3 2011 13:48:20

gem5 executing on maize

command line: ./build/ARM_SE/gem5.opt configs/example/se.py

Global frequency set at 1000000000000 ticks per second

0: system.remote_gdb.listener: listening for remote gdb #0 on port 7000

**** REAL SIMULATION ****

info: Entering event queue @ 0. Starting simulation...

Hello world!

hack: be nice to actually delete the event here

Exiting @ tick 3350000 because target called exit()
Basic syscall (SE) options

- [http://gem5.org/Running_gem5](http://gem5.org/Running_gem5)

```
command line: build/ALPHA/gem5.opt configs/example/se.py -h
Usage: se.py [options]

Options:
  -h, --help            show this help message and exit
  -c CMD, --cmd=CMD     The binary to run in syscall emulation mode.
  -o OPTIONS, --options=OPTIONS
                         The options to pass to the binary, use " " around entire string
  -i INPUT, --input=INPUT
                         Read stdin from a file.
  --output=OUTPUT       Redirect stdout to a file.
  --errout=ERROUT       Redirect stderr to a file.
  --ruby
  --detailed
  --t, --timing
  --inorder
  -n NUM_CPUS, --num-cpus=NUM_CPUS
  --caches
  --l2cache
  --fastmem
  --clock=CLOCK
  --num-dirs=NUM_DIRS
  --num-l2caches=NUM_L2CACHES
  --l1d_size=L1D_SIZE
  --l1i_size=L1I_SIZE
  --l2_size=L2_SIZE
  --l1d_assoc=L1D_ASSOC
  --l1i_assoc=L1I_ASSOC
  --l2_assoc=L2_ASSOC
...```
Modifying the scripts

• To configure the micro-architectural parameters which are not available to set through command-line
  e.g. pipeline length, instruction latencies, etc.

• Folder: config/
  e.g. common/Caches.py; common/cores/O3_ARM_v7a.py
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.

- **cpt.<number>/** [when using checkpoints]
Compiling a benchmark for SE

- Do all these experiments with queens.c
- Very old benchmark, but it’s easy to get and understand
- $wget https://llvm.org/svn/llvm-project/test-suite/tags/RELEASE_14/SingleSource/Benchmarks/McGil/queens.c
- $arm-linux-gnueabi-gcc -DUNIX -o queens queens.c -static
- All binaries must be compiled with static flag
Running compiled program

Under /<dir>/gem5

- $./build/ARM/gem5.opt configs/example/se.py -c queens -o 16

- Running with caches and detailed CPU

- $./build/ARM/gem5.opt configs/example/se.py -c queens -o 16 --caches --l2cache --cpu-type=ex5_big

Error?

FATAL: kernel too old

- Check cross-compiler version ($arm-linux-gnueabi-gcc --version)
- Change the #version on line #72 of src/arch/arm/linux/process.cc (strcpy(name->release, "x.y.z"));
Statistics output

- `m5out/stats.txt`

```plaintext
---------- Begin Simulation Statistics ----------

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>sim_seconds</td>
<td>6048.007619</td>
</tr>
<tr>
<td>sim_ticks</td>
<td>6048007619094688</td>
</tr>
<tr>
<td>final_tick</td>
<td>6048007619094688</td>
</tr>
<tr>
<td>(checkpoints and never reset)</td>
<td></td>
</tr>
<tr>
<td>sim_freq</td>
<td>1000000000000</td>
</tr>
<tr>
<td>host_inst_rate</td>
<td>171692</td>
</tr>
<tr>
<td>host_op_rate</td>
<td>212634</td>
</tr>
<tr>
<td>host_tick_rate</td>
<td>378817251850</td>
</tr>
<tr>
<td>host_mem_usage</td>
<td>2926108</td>
</tr>
<tr>
<td>host_seconds</td>
<td>15965.50</td>
</tr>
<tr>
<td>sim_insts</td>
<td>2741152858</td>
</tr>
<tr>
<td>sim_ops</td>
<td>3394816356</td>
</tr>
<tr>
<td>system.voltage_domain.voltage</td>
<td>1</td>
</tr>
<tr>
<td>system.clk_domain.clock</td>
<td>1000</td>
</tr>
</tbody>
</table>

# Number of seconds simulated
# Number of ticks simulated
# Number of ticks from beginning of simulation (restored from
# Frequency of simulated ticks
# Simulator instruction rate (lnst/s)
# Simulator op (including micro ops) rate (op/s)
# Simulator tick rate (ticks/s)
# Number of bytes of host memory used
# Real time elapsed on the host
# Number of instructions simulated
# Number of ops (including micro ops) simulated
# Voltage in Volts
# Clock period in ticks
```
MiBench and Parsec Benchmark

- Source Code:  
  [http://wwwweb.eecs.umich.edu/mibench/source.html](http://wwwweb.eecs.umich.edu/mibench/source.html)

- Using two apps from *mibench*, FFT from telecom and SUSAN from automotive categories, to simulate on ARM and calculate the runtime.


Additional topic: ARM’s big.LITTLE architecture

- ARM’s big.LITTLE – a solution for power efficiency
- big – high performance, but power hungry
- LITTLE – low performance, but power efficient
- gem5 now supports simulation of big.LITTLE\textsuperscript{1}

\texttt{/<dir>/gem5/configs/example/arm/}

\textsuperscript{1}https://pdfs.semanticscholar.org/520e/555592f88df7eb009c3512de155462dc3b1e.pdf
Suggested resources

- **Book:**
  - [https://github.com/powerjg/learning_gem5](https://github.com/powerjg/learning_gem5)

- **Video:** [https://www.youtube.com/watch?v=5UT41VsGTsg](https://www.youtube.com/watch?v=5UT41VsGTsg)

- **Main gem5 wiki:** [http://gem5.org/](http://gem5.org/)

- **Mailing lists:** [http://gem5.org/Mailing_Lists](http://gem5.org/Mailing_Lists)
  - **gem5-users:** General user questions
  - **gem5-dev:** Mostly code reviews and high-level dev talk
Thank You