DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GEM5 SYSTEM EMULATOR MODE(SE) REPORT

SUBMITTED BY

VIKAS B

11CO100
CONTENTS:

This report contains various features, components, advantages and goals of gem5 architecture simulator. Report contains detailed steps to install gem5 architecture simulator in Ubuntu operating system from scratch.

After installation of gem5 simulator, steps are shown to build various architectures such as arm, alpha and x86 in gem5 architecture. Steps to run and obtain output stats in System Emulator mode for default cross compiled programs which comes as default in various built architectures are also explained.

Further, steps to cross compile a particular C program to architectures such as arm, alpha and x86 are also shown. Procedure to run cross compiled C program in various architectures such as alpha, arm and x86 in SE mode is also shown. Screenshots are attached to further aid the purpose.
INTRODUCTION

GEM5 (General Execution-driven Multiprocessor) Simulator is a computer architecture simulator used for computer architecture research purposes. The gem5 simulator is a modular platform for computer system architecture research, encompassing system-level architecture as well as processor microarchitecture. A **computer architecture simulator**, or an architectural simulator, is a piece of software to model computer devices (or components) to predict outputs and performance metrics on a given input. An architectural simulator can model a target microprocessor only or an entire computer system including a processor, a memory system, and I/O devices.

**Architectural simulators** are very useful for the following purposes:

- evaluating different hardware designs without building costly physical hardware systems.
- enabling the opportunities to access non-existing computer components or systems
- obtaining detailed performance metrics: A single execution of simulators can often generate a large set of performance data.
- debugging: Debugging on real hardware typically require re-booting and re-running the code to reproduce the problems. In contrast, some simulators have a fully controlled environment and allow software developers to run code backward once an error is detected.

Full system simulation can speed the system development process by making it easier to detect, recreate and repair flaws. The use of multi-core processors is driving the need for full system simulation, because it can be extremely difficult and time consuming to recreate and debug errors without the controlled environment provided by virtual hardware. This also allows the software development to take place before the hardware is ready, thus helping to validate design decisions. Thus Gem5 can also be made into a **full system simulator**.

WHAT IS GEM5

The gem5 simulation infrastructure is the merger of the best aspects of the M5 and GEMS simulators. M5 provides a highly configurable simulation framework, multiple ISAs, and diverse CPU models. GEMS complement these features with a detailed and flexible memory system, including support for multiple cache coherence protocols and interconnect models. Currently, gem5 supports most commercial ISAs (ARM, ALPHA, MIPS, Power, SPARC, and x86), including booting Linux on three of them (ARM, ALPHA, and x86). The project is the result of the combined efforts of many academic and industrial
institutions, including AMD, ARM, HP, MIPS, Princeton, MIT, and the Universities of Michigan, Texas, and Wisconsin. Over the past ten years, M5 and GEMS have been used in hundreds of publications and have been downloaded tens of thousands of times. The high level of collaboration on the gem5 project, combined with the previous success of the component parts and a liberal BSD-like license, make gem5 a valuable full-system simulation tool.

WHY GEM5

Gem5 is a modular discrete event driven computer system simulator platform. That means that:

1. gem5's components can be rearranged, parameterized, extended or replaced easily to suit your needs.
2. It simulates the passing of time as a series of discrete events.
3. Its intended use is to simulate one or more computer systems in various ways.
4. It's more than just a simulator; it's a simulator platform that lets you use as many of its premade components as you want to build up your own simulation system.

KEY FEATURES OF GEM5:

- **Pervasive object orientation.** Major simulation structures (CPUs, busses, caches, etc.) are represented as objects, both externally and internally. The gem5 configuration language allows flexible composition of these objects to describe complex simulation targets, e.g., multi-system networks where each system comprises multiple CPUs and a hierarchy of caches. The simulator's internal object orientation (using C++) provides in addition to the usual software engineering advantages.

- **Multiple interchangeable CPU models.** The gem5 simulator currently provides three interchangeable CPU objects: a simple, functional, one-CPI CPU; a detailed model of an out-of-order SMT-capable CPU; and a random memory-system tester. The first two models use a common high-level ISA description.

- **Event-driven memory system.** The gem5 simulator features a detailed, event-driven memory system including non-blocking caches and split-transaction busses. These components can be arranged flexibly, e.g., to model complex multi-level cache hierarchies. The caches support a separable coherence policy module; gem5 currently includes a simple snooping cache coherence protocol.

- **Multiple ISA support.** The gem5 simulator decouples ISA semantics from its timing CPU models, enabling effective support of multiple ISAs. The gem5 simulator currently supports the Alpha, ARM, SPARC, MIPS, POWER and x86 ISAs.
- **Full-system capability.**
  - **Alpha:** The gem5 simulator models a DEC Tsunami system in sufficient detail to boot unmodified Linux 2.4/2.6, FreeBSD, or L4Ka::Pistachio.
  - **ARM:** The gem5 simulator can model up to four cores of a Real view ARM development board with sufficient detail to boot unmodified Linux 2.6.35+ with a simple or out-of-order CPU.
  - **SPARC:** The gem5 simulator models a single core of a Ultra SPARC T1 processor with sufficient detail to boot Solaris in a similar manner as the Sun T1 Architecture simulator tools (building the hypervisor with specific defines and using the HSMID virtual disk driver).
  - **x86:** The gem5 simulator supports a standard PC platform

- **Multiprocessor / multi-system capability.** Thanks to gem5's object orientation, instantiation of multiple CPU objects within a system is trivial. Combined with the snooping bus-based coherence protocol supported by the caches, gem5 can model symmetric multiprocessor systems. Because a complete system is just a collection of objects (CPUs, caches, memory, etc.), multiple systems can be instantiated within a single simulation process. In conjunction with full-system modelling, this feature allows simulation of entire client-server networks.

**CAPABILITIES OF GEM5:**
The gem5 simulator has a wide range of simulation capabilities ranging from the selection of ISA, CPU model, and coherence protocol to the instantiation of interconnection, networks, devices and multiple systems. This section describes some of the different options available in these categories.

- **ISA (INSTRUCTION SET ARCHITECTURE):** The gem5 simulator currently supports a variety of ISAs including Alpha, ARM, MIPS, Power, SPARC, and x86. The simulator's modularity allows these different ISAs to plug into the generic CPU models and the memory system without having to specialize one for the other.

- **Execution Modes.** The gem5 simulator can operate in two modes: System-call Emulation (SE) and Full-System (FS). In SE mode, gem5 emulates most common system calls (e.g. read()). Whenever the program executes a system call, gem5 traps and emulates the call, often by passing it to the host operating system. In FS mode, gem5 simulates a bare-metal environment suitable for running an OS. This includes support for interrupts, exceptions, privilege
levels, I/O devices, etc. Because of the additional complexity and completeness required, not all ISAs currently support FS mode.

**CPU Models.** The gem5 simulator supports four different CPU models: Atomic Simple, Timing Simple, In-Order, and O3. Atomic Simple and Timing Simple are non-pipelined CPU models that attempt to fetch, decode, execute and commit a single instruction on every cycle.

The Atomic Simple CPU is a minimal, single IPC CPU which completes all memory accesses immediately.

The In-Order model is an "execute-in-execute" CPU model emphasizing instruction timing and simulation accuracy with an in-order pipeline.

Finally, the O3 CPU is a pipelined, out-of-order model that simulates dependencies between instructions, functional units, memory accesses, and pipeline stages.

**Cache Coherence Protocols.** SLICC enables gem5's Ruby memory model to implement many different types of invalidation-based cache coherence protocols, from snooping to directory protocols and several points in between. SLICC separates cache coherence logic from the rest of the memory system, providing the necessary abstraction to implement a wide range of protocol logic.

**Interconnection Networks.** The Ruby memory model supports a vast array of interconnection topologies and includes two different network models. In essence, Ruby can create any arbitrary topology as long as it is composed of point-to-point links. A simple Python file declares the connections between components and shortest path analysis is used to create the routing tables. Once Ruby creates the links and routing tables, it can implement the resulting network in one of two ways.

**Devices.** The gem5 simulator supports several I/O devices ranging from simple timers to complex network interface controllers. Base classes are available that encapsulate common device interfaces such as PCI to avoid code duplication and simplify implementing new devices.

**Modeling Multiple Systems.** Because of the simulator's object-oriented design, it also supports simulating multiple complete systems. This is done by instantiating another set of objects (CPU, memory, I/O devices, etc.).
GOALS OF GEM5:

FLEXIBILITY:
Flexibility is a fundamental requirement of any successful simulation infrastructure. For instance, as an idea evolves from a high-level concept to a specific design, architects need a tool that can evaluate systems at various levels of detail, balancing simulation speed and accuracy. Different types of experiments may also require different simulation capabilities. For example, a fine-grain clock gating experiment may require a detailed CPU model, but modelling multiple cores is unnecessary. Meanwhile, a highly scalable interconnect model may require several CPUs, but those CPUs don't need much detail. Also, by using the same infrastructure over time, an architect will be able to get more done more quickly with less overhead.

AVAILABILITY
There are several types of gem5 user; each has different goals and requirements. These include academic and corporate researchers, engineers in industry, and undergraduate and graduate students

COLLABORATION
Computer architecture researchers also need a simulation framework that allows them to collaborate with their colleagues in both industry and academia. However, a simulator's licensing terms and code quality can inhibit that collaboration. Some open source software licenses can be too restrictive, especially in an industrial setting, because they require publishing any simulator enhancements.

INSTALLING GEM5 SIMULATOR:

IN WINDOWS:
Steps:
1. Install Mercurial platform from selenic.com
2. Run cmd
3. Type the command `hg clone http://repo.gem5.org/gem5`

IN UBUNTU:
Steps:
Can be installed either with Mercury or Structural Simulation Kit
**Mercury ARM Steps:**
1. Install Mercury by command `sudo apt install mercurial scons swig gcc m4 python python-dev libgoogle perftools -dev g++`
2. Install gem5 by typing the following command in terminal `hg clone http://repo.gem5.org/gem5`
3. Type `scons build/ARM/gem5.opt`. The command is for arm architecture and `opt` stands for optimization. `gem5.opt` is one of the four `gem5` options.

**Mercury x86 Steps: (Assuming gem5 installed)**
1. Change to gem5 folder by `cd gem5`. Type `scons build/X86/gem5.opt`. The command is for arm architecture and `opt` stands for optimization. `gem5.opt` is one of the four `gem5` options.

**Mercury ALPHA Steps: (Assuming gem5 installed)**
1. Change to gem5 folder by `cd gem5`. Type `scons build/ALPHA/gem5.opt`. The command is for arm architecture and `opt` stands for optimization. `gem5.opt` is one of the four `gem5` options.

**NOW LET US RUN SIMPLE (DEFAULT) HELLO WORLD PROGRAM IN GEM5 SIMULATOR IN SYSTEM CALL EMULATION (SE) MODE.**

**ARM Steps:**

1. Open the Terminal and type `cd gem5`
2. Type `./build/ARM/gem5.opt configs/example/se.py -c tests/test-progs/hello/bin/arm/linux/hello`
3. Output will be shown as follows
X86 Steps:

1. Type `.build/X86/gem5.opt configs/example/se.py -c tests/test-progs/hello/bin/x86/linux/hello`
ALPHA STEPS:
1. Type `./build/ALPHA/gem5.opt configs/example/se.py -c tests/test-progs/hello/bin/alpha/linux/hello`

NOW LET US RUN ANY PROGRAM IN GEM5 SIMULATOR IN SYSTEM CALL EMULATION (SE) MODE.

Let us run a factorial problem to find the factorial of 10 named fact.c and a subdirectory vikas under gem5. fact.c is inside vikas.

ARM Steps:
1. Open the terminal
2. Type `cd gem5`
3. Let us assume that sub directory was vikas and filename was fact.c. Enter the following command `arm-linux-gnueabi-gcc -DUNIX -o ./vikas/fact ./vikas/fact.c -static`. The command is to cross compile the file
4. To run the compiled program in arm architecture type
5. `./build/ARM/gem5.opt configs/example/se.py -c vikas/fact(path to exe)`
6. Output Shown Below in Screenshot
7. To trace the output. Change directory to gem5
8. 
```
./build/ARM/gem5.opt --debug-flags=Exec,ExecTicks configs/example/se.py -c vikas/fact | more
```
9. To find the statistics of simulation use the command

```
./build/ARM/gem5.opt configs/example/se.py -c vikas/fact -- options="16"
```

11. Now go to m5out solder inside gem5 folder and open stats.txt
X86 Steps:
1. Open the terminal
2. cd gem5
3. We have created a folder vikas and fact.c inside vikas
4. cd vikas
5. now type following command
6. gcc -static -o fact.x86 fact.c This is to cross compile so as to run the program on x86 architecture
7. Now run the program
8. Now type cd … Change directory to gem5.
9. ./build/X86/gem5.opt configs/example/se.py -c vikas/fact.x86 where vikas/fact.x86 is path to the extension fact.x86
10. trace the compiled program in gem5
11. ./build/ARM/gem5.opt --debug-flags=Exec,ExecTicks configs/example/se.py -c vikas/fact(path to exe file) | more

12. To find statistics
13. ./build/X86/gem5.opt configs/example/se.py -c vikas/fact.x86(path to exe file) --options="16"
14. Now go to m5out solder inside gem5 folder and open stats.txt
ALPHA STEPS:

1. Open the Terminal
2. Cd gem5
3. Make a directory CROSSCOMPILER under gem5 by command mkdir CROSSCOMPILER.
4. Download the alpha cross compiler package(gcc 3.4.3) for Ubuntu 32 or 64 bit depending on Ubuntu.(From GEM5 Website).
5. Extract the file using tar jxf alphaev67-unknown-linux-gnu-x86-32.tar.bz2
6. Now Make a new directory called vikas under gem5 folder. You can create a directory anywhere and paste the file in it. But Make sure you Point out the path properly in later stages.
7. Paste the fact.c file in vikas folder
8. Change to directory vikas to type the command
9. `/home/vikas/gem5/CROSSCOMPILER/alpha-unknown-linux-gnu/bin/alpha-unknown-linux-gnu-gcc -o fact fact.c -static (/home/vikas/gem5/CROSSCOMPILER…is the path name to the crosscompiler package we created)
10. To run the fact file change to gem5 directory and type the following command ./build/ALPHA/gem5.opt configs/example/se.py -c vikas/fact
11. To trace the fact file change to gem5 directory and type the following command ./build/ALPHA/gem5.opt --debug-flags=Exec,ExecTicks configs/example/se.py -c vikas/fact | more
To find statistics

12. `./build/ALPHA/gem5.opt configs/example/se.py -c vikas/fact(path to exe file) --options="16"

13. Now go to m5out solder inside gem5 folder and open stats.txt
NOTE FOR ALL ARCHITECTURES TO RUN A THREADED PROGRAM ONE HAS TO ADD –
lpthread COMMAND IN THE COMMAND LINE