

**Architectural Simulation Using gem5**  
**Dipika Deb, Ph.D Scholar**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology - Guwahati**

**Tutorial 4**  
**MARS Research Lab**

Hello, everyone. Welcome to the advanced computer architecture course. I am Dipika Deb, a Ph.D. scholar in the Department of Computer Science and Engineering, IIT, Guwahati. I work under the supervision of Dr. John Jose and my research area of interest is multi core architecture with special focus on prefetching and compression. This is the first video lecture on gem5 session. And we have planned 2 other such videos.

The first session is an introductory session on the gem5 simulator, while the second and the third deals with code optimization and memory optimization. So, with this, I start today's session. gem5 is a full system simulator that is extensively used by the computer architecture community. This is because gem5 simulates a real hardware, where one can adopt any modifications for experimental purpose. The need for such a modification also arises from the fact that fabricating all modifications on a real hardware is not actually possible.

This is because the fabrication cost is too high and also quite time consuming. Due to this reason, an architectural simulator, such as gem5 is used for the experimental purpose.

**(Refer Slide Time: 01:51)**

### Outline

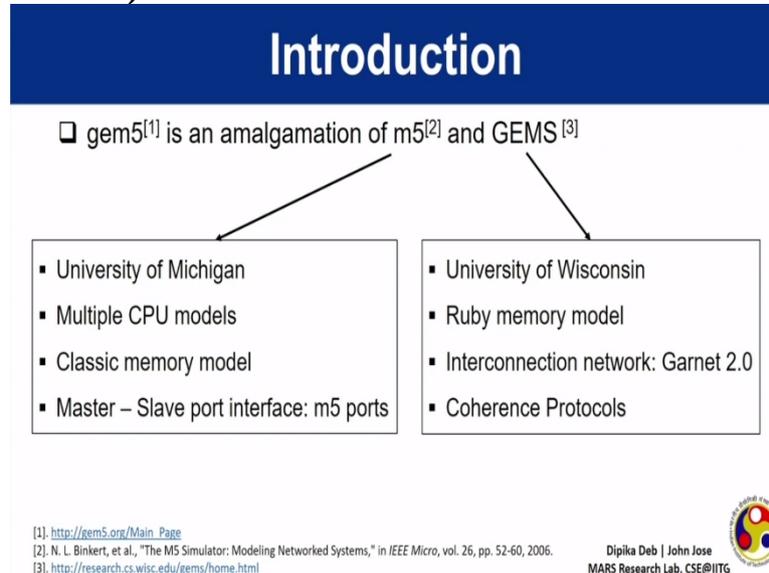
- Introduction to gem5
- Working of gem5
- Configure gem5
- Build and Run gem5
- Real Benchmarks
- Analysing gem5 output



Dipika Deb | John Jose  
MARS Research Lab, CSE@IITG

This is the outline of today's presentation, I will introduce to the readers the full system simulator that is gem5, followed by how it works. Then I will show you how to configure and build gem5 in your system. I will also show the real benchmarks that can also be used for evaluating the system performance in gem5. Finally, I will explain the output statistics that are generated from the simulated machine.

**(Refer Slide Time: 02:23)**



Gem5 is an amalgamation of m5 and GEMS. m5 is built by University of Michigan with multiple CPU models and instruction set architectures. It also has a classic memory model. The CPU and memory in m5 interacts with each other as master and slave using the available m5 ports. On the other hand, GEMS is built by University of Wisconsin with Ruby memory model and it also has a flexible interconnection backbone that is Garnet 2.0.

It also models different types of coherence protocols. In the previous slide, as I have shown, the CPU models available in gem5 will be described in details in the second video of gem5, while the memory models will be described in the third lecture of the gem5. gem5 is a modular discrete event driven simulator, that is all the actions in the system are represented as events that are time stamped. The events are stored in an event queue and a time is attached with an event. As per the time an event get scheduled.

**(Refer Slide Time: 03:24)**

# Introduction

□ gem5 is a modular discrete event driven simulator [1].

• For example:

1. Core requests for an address: "A"
2. Searches in L1 cache.
3. Hit in L1 cache.

Event Queue

Event:Request: 1

Event:Request: 2

Event:Reply: 3


[1]. [http://gem5.org/Main\\_Page](http://gem5.org/Main_Page)

Dipika Deb | John Jose  
MARS Research Lab, CSE@IITG



For example, let us suppose that this is an event queue and a core request for an address A. The request is processed as an event that is entered into the event queue. Assuming that the current clock cycle is 1, the event is time stamped with 1. As the event gets dequeued the normal procedure is whenever a core generates an address it is initially searched in the L1 cache.

Hence searching in the L1 cache is another event and that is time stamped with clock cycle 2 and that is entered in the into the event queue if the L1 cache success, turns out to be a hit in the cache, the L1 cache controller sends the block back to the core. This is then generates another event which is known as a reply event with timestamp 2, this event is also inserted into the event queue.

Kindly note that this is a representative example to describe an event driven simulation and this is how the simulation occurs in gem5. Gem5 is also known as a full system simulator.

**(Refer Slide Time: 04:59)**

# Introduction

- ❑ gem5 can simulate a complete system including devices and operating system: *full system simulator or FS mode*.
- ❑ gem5 also operates on SE mode (SysCall Emulation mode)
- ❑ gem5 simulates a machine also known as the “*simulated machine*” on the host machine.
- ❑ It is written in C++ and python.
- ❑ All experiments are executed on the simulated machine.

[1]. [http://gem5.org/Main\\_Page](http://gem5.org/Main_Page)

Dipika Deb | John Jose  
MARS Research Lab, CSE@IITG



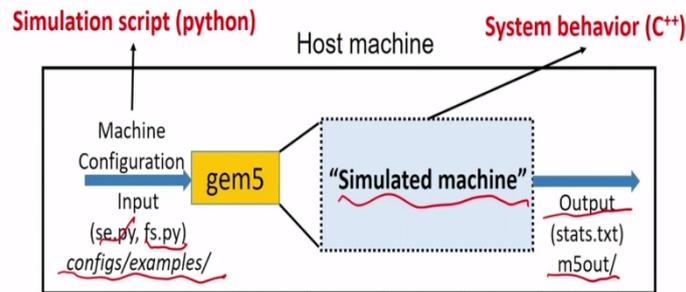
By full systems simulation I mean it executes both user level codes and kernel codes and hence models a complete system including devices and operating system. Gem5 also operates on SE mode or this is also known as Syscall emulation mode. In SE mode, it emulates most of the system level services that is kernel codes and just executes the user code. Hence this mode does not model any operating system or any devices.

Gem5 simulates a machine which we call a simulated machine and the machine on which gem5 is running is known as the host machine. Remember the term host machine and simulated machine as I will use this 2 terms throughout the video lecture. Gem5 is written in C++ and Python and all the operations and experiments are performed on the simulated machine.

The figure shows a diagram of how gem5 builds a simulated machine on installing gem5 on a host machine it takes an input parameter. This input parameter is actually a simulation script which is written in python. This script contains configuration of the simulated machine that is to be by gem5. Few such input simulation scripts are present in this path. One such script is se.py and the next one is fs.py. I will explain more about the simulation scripts in the next slide.

**(Refer Slide Time: 06:02)**

# Working of gem5



[http://gem5.org/Main\\_Page](http://gem5.org/Main_Page)

Dipika Deb | John Jose  
MARS Research Lab, CSE@IITG



Gem5 on taking the input generates and builds a simulated machine the behaviour of the simulated machine can be observed as an output file in the m5 out directory inside gem5 the behaviour of this machine is implemented in C++. As I have already mentioned, that gem5 simulator takes a simulation script which is written in Python as input that is se.py and the simulated machines behaviour is written in C++. Now coming to the simulation script, the Python script se.py contains 2 parts.

(Refer Slide Time: 07:08)

# Working of gem5

- ❑ gem5 simulator takes a simulation script which is written in **python** as input and the simulated machine's behavior is written in **C++**.
- ❑ The simulation script has two phase: *se.py*
  1. **Configuration phase**: simulated machine specification
  2. **Simulation phase**: simulates the required machine on the host machine.
- ❑ Simulated machine specifications (in python) are converted into C++ struct through SimObjects and passed to the C++ objects.

[http://gem5.org/Main\\_Page](http://gem5.org/Main_Page)

Dipika Deb | John Jose  
MARS Research Lab, CSE@IITG



The first part is configuration phase. This says what the configuration of the simulated machine. And the next phase is simulation phase. This phase mentions how to build the machine. The configuration from the python script are then converted into C++ structures through Sim Objects and these are then passed to the C++ objects which models the system behaviour. This Sim Objects generally represent some physical components such as cache, main memory, or DRAM.

All these components are represented using different Sim Objects. I will explain the whole procedure in greater detail in the subsequent slides. Now moving forward, gem5 operates on 2 modes. SE mode and full system mode. Gem5 broadly simulates 2 CPU models in order CPU and out of order CPU. And it has 2 memory models classic memory and Ruby memory. Gem5 has 2 interconnection networks simple and Garnet. In this course, we will mostly handle with CPU models and memory models.

(Refer Slide Time: 08:23)

## Working of gem5

- ❑ gem5 operates on two modes:  
    System Emulation (SE) mode, Full System (FS) mode.
- ❑ gem5 can simulate two CPU models:  
    In-order, Out-of-Order
- ❑ gem5 can have two memory models:  
    Classic memory model, Ruby memory model
- ❑ gem5 has two interconnection networks:  
    Simple, Garnet

  
Dipika Deb | John Jose  
MARS Research Lab, CSE@IITG

[http://gem5.org/Main\\_Page](http://gem5.org/Main_Page)

Throughout our assignment, we will use the system emulation mode or SE mode. We will use both the CPU models in order as well as out of order, which I will explain in greater detail in the second video lecture of gem5. And we will use Ruby memory model along with the garnet interconnection networks.

(Refer Slide Time: 09:17)

## gem5 Dependencies

- ❑ gem5 can be installed in different platforms:  
    Linux, BSD, MacOS, Solaris.
- ❑ gem5 Dependencies:
  - git/Mercurial (*download gem5 simulator*)
  - g++ / gcc (>= 4.8)
  - Python (>=2.7)
  - Scons (0.98.1+)
  - SWIG(2.0.4+)
  - M4
  - Zlib

  
Dipika Deb | John Jose  
MARS Research Lab, CSE@IITG

[http://gem5.org/Main\\_Page](http://gem5.org/Main_Page)

Gem5 is dependent on few packages. First gem5 can be installed in different plat forms. It can be installed in Linux, BSD Mac OS, Solaris, and it is dependent on the following packages: git on Mercurial, This is used to clone or download gem5. The G ++ or GCC compiler is used to build the C++ system files. Python is required to interpret the simulation scrips and scon is required to build the complete gem5 simulator.

And finally swig M4 Zlib at the extra packages that are required by gem5 for installation. Now let us see how do we configure gem5 in our system.

**(Refer Slide Time: 10:10)**

## Configure gem5

- 1. Install dependencies (Ubuntu)**  
`sudo apt-get install git build-essential python-dev, scon, swig, m4, zlibg-dev`
- 2. Download gem5:**  
`git clone https://github.com/dipika131991/gem5.git`
- 3. Build gem5:**  
`scons build/isa/binary -j no_of_CPUs+1`
- 4. Run gem5:**  
`<gem5_build_binary> <gem5_options> <path_to_simulation_script> \\  
<options_to_simulation_script>`

[http://gem5.org/Main\\_Page](http://gem5.org/Main_Page)

Dipika Deb | John Jose  
MARS Research Lab, CSE@IITG

These are the commands to install the dependencies type the following command in the terminal. I will use an Ubuntu system to show a demo. You can download gem5 using this command. We can then build gem5 using this command and to run gem5 we will use this command. I will explain the build and run command in greater details.

**(Refer Slide Time: 10:41)**

# Building gem5

3. `scons build/ALPHA/gem5 opt -j 9`

## Simulated Machine's ISA:

- **ALPA**
- ARM
- X86
- MIPS
- NULL
- SPARC
- PowerPC

- Number of cores/threads in the system + 1
- For Ubuntu: `lscpu`

## binaries: describes the optimization Level

- **debug**: used during debugging gem5 code without any optimization
- **opt**: debug + optimization
- **fast, prof, perf**

[http://gem5.org/Main\\_Page](http://gem5.org/Main_Page)

Dipika Deb | John Jose  
MARS Research Lab, CSE@IITG

This is the build command where the instruction set architecture used is alpha. Throughout our simulations and assignment, we will use the alpha instruction set architecture. So this is the simulated machine's instruction set architecture. And in gem5 there are several other ISAs available that are alpha, ARM, x86, MIPS, null, spark, and PowerPC. Gem5.opt is the binary that is commonly used during working with gem5, it is fast and it has almost all of the debug features.

There are other types of binaries available such as perf, prof, fast, and gem5.debug. Next, -j 9 indicates the number of cores or threads in the system + 1. For Ubuntu users, you can type this command to check the number of cores that you have in your system.

(Refer Slide Time: 11:59)

```
narslab@mars-Lenovo:~/Desktop/Dipika/gem5$ lscpu
Architecture:          x86_64
CPU op-mode(s):       32-bit, 64-bit
Byte Order:           Little Endian
CPU(s):               8
On-line CPU(s) list:  0-7
Thread(s) per core:   2
Core(s) per socket:   4
Socket(s):            1
NUMA node(s):        1
Vendor ID:            GenuineIntel
CPU Family:           6
Model:               94
Model name:          Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz
Stepping:            3
CPU MHz:             800.046
CPU max MHz:         3500.0000
CPU min MHz:         800.0000
BogoMIPS:            5184.00
Virtualization:      VT-x
Lid cache:           32K
L1i cache:           32K
L2 cache:            256K
L3 cache:            6144K
NUMA node0 CPU(s):  0-7
Flags:                fpu vme de pse tsc msr pae mce cx8 apic sep ntr pge mca cmov pat pse36 clflush dts acpi nx fxsr s
se sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp ln constant_tsc art arch_perfmon pebs bts rep_good nopl xtopology nonstop_t
sc cpuid aperfmperf tsc_known_freq pni pclmulqdq dtes64 monitor ds_cpl vmx est tm2 sse3 sdbg fma cx16 xtpr pdcm pcid sse4
_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm 3dnowprefetch cpuid_fault epb invpc
id_single pti ssbd ibrs ibpb stibp tpr_shadow vmmi flexpriority ept vpid fsgsbase tsc_adjust bmi1 hle avx2 smep bmi2 erms
invpcid rtm mpx rdseed adx smap clflushopt intel_pt xsaveopt xsavec xgetbv1 xsaves dtherm ida arat pln pts hwp hwp_notify
hwp_act_window hwp_epp flush_lid
narslab@mars-Lenovo:~/Desktop/Dipika/gem5$
```

After building gem5, you will notice that inside the gem5 directory a new directory named build is automatically generated. Now, we will run gem5 using the following command.

(Refer Slide Time: 12:14)

## Run gem5

4. build/ALPHA/gem5.opt configs/example/se.py \  
-c tests/test-progs/hello/bin/alpha/linux/hello

**Component:**

- ❑ gem5 binary: *gem5.opt*
- ❑ options for the simulation script: *can be found using "build/ALPHA/gem5.opt config/examples/se.py --help"*
- ❑ a simulation script: *se.py,*
- ❑ options for the simulation script: *-C tests/test-progs/hello/bin/arm/linux/hello = -c (your\_code)*

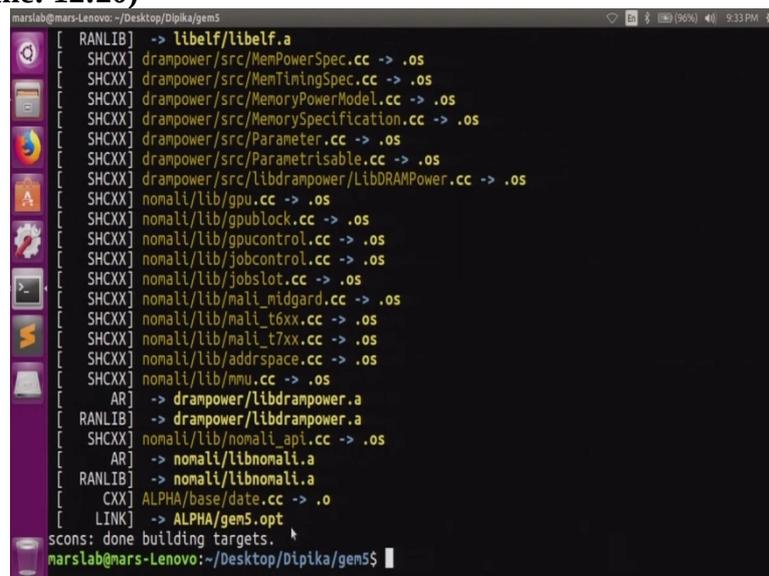
[http://gem5.org/Main\\_Page](http://gem5.org/Main_Page)

Dipika Deb | John Jose  
MARS Research Lab, CSE@IITG



Now I will show a demo of the complete process that is to be carried out in your system to build a gem5 system, we will type `scons build` we will use the alpha architecture `gem5.opt` binary and the number of cores that I have in my system is 8.

(Refer Slide Time: 12:20)

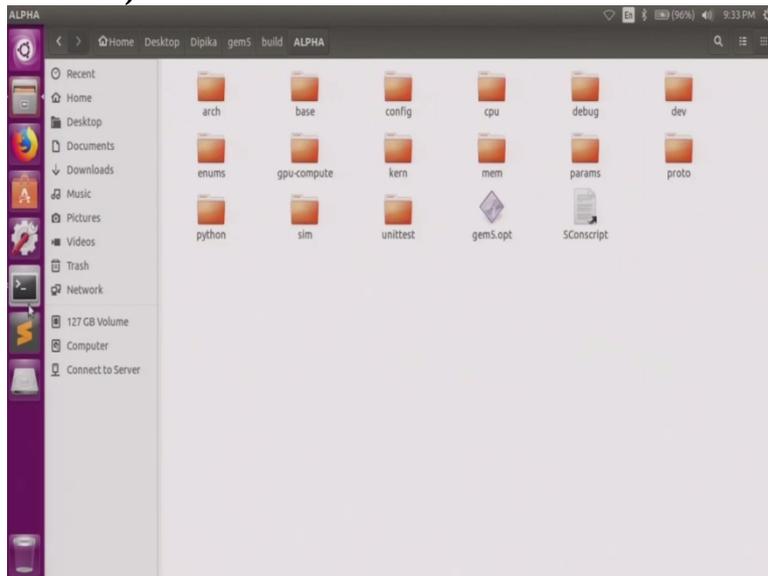


```
marslab@mars-Lenovo: ~/Desktop/Dipika/gem5
[ RANLIB ] -> libelf/libelf.a
[ SHCXX ] dranpower/src/MemPowerSpec.cc -> .os
[ SHCXX ] dranpower/src/MemTimingSpec.cc -> .os
[ SHCXX ] dranpower/src/MemoryPowerModel.cc -> .os
[ SHCXX ] dranpower/src/MemorySpecification.cc -> .os
[ SHCXX ] dranpower/src/Parameter.cc -> .os
[ SHCXX ] dranpower/src/Parametrisable.cc -> .os
[ SHCXX ] dranpower/src/libdranpower/LibDRAMPower.cc -> .os
[ SHCXX ] nomali/lib/gpu.cc -> .os
[ SHCXX ] nomali/lib/gpublock.cc -> .os
[ SHCXX ] nomali/lib/gpucontrol.cc -> .os
[ SHCXX ] nomali/lib/jobcontrol.cc -> .os
[ SHCXX ] nomali/lib/jobslot.cc -> .os
[ SHCXX ] nomali/lib/mali_midgard.cc -> .os
[ SHCXX ] nomali/lib/mali_t6xx.cc -> .os
[ SHCXX ] nomali/lib/mali_t7xx.cc -> .os
[ SHCXX ] nomali/lib/addrspace.cc -> .os
[ SHCXX ] nomali/lib/nmu.cc -> .os
[ AR ] -> dranpower/libdranpower.a
[ RANLIB ] -> dranpower/libdranpower.a
[ SHCXX ] nomali/lib/nomali_api.cc -> .os
[ AR ] -> nomali/libnomali.a
[ RANLIB ] -> nomali/libnomali.a
[ CXX ] ALPHA/base/date.cc -> .o
[ LINK ] -> ALPHA/gem5.opt
scons: done building targets.
marslab@mars-Lenovo: ~/Desktop/Dipika/gem5$
```

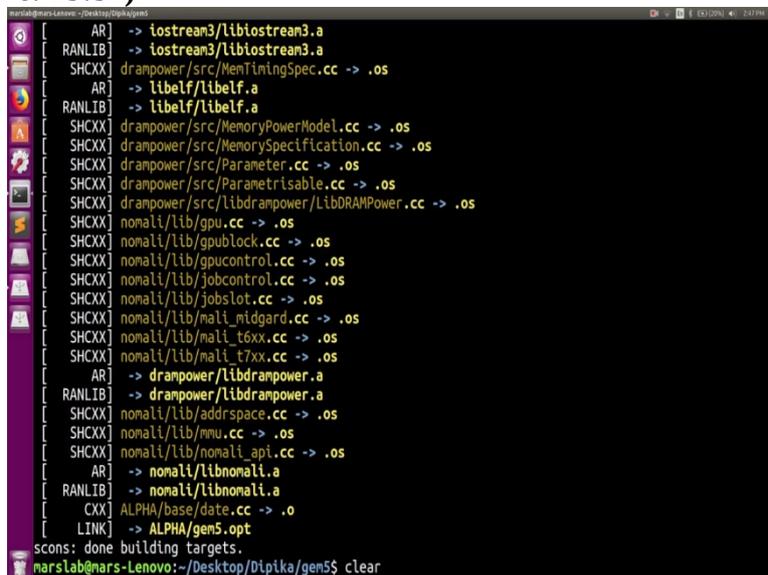
So I will use  $8 + 1$  that is 9 as the number of threads in my system. This process requires quite a time around 10 minutes to perform the complete building of gem5. So, at this stage, you can see that the `gem5.opt` binary has been created. If you look inside the folder you will find that inside `gem5` a build directories `build`. Inside the build directory you will find the

alpha architecture that has been build using this scon's command. Now, we will run a Hello program in the gem5 build directory.

**(Refer Slide Time: 13:16)**



**(Refer Slide Time: 13:32)**



**(Refer Slide Time: 13:34)**

```
marslab@mars-Lenovo:~/Desktop/Dipika/gem5$ build/ALPHA/gem5.opt configs/example/se.py -c tests/test-progs/hello/bin/alpha/linux/hello
gem5 Simulator System. http://gem5.org
gem5 is copyrighted software; use the --copyright option for details.

gem5 compiled Jul 31 2019 14:37:08
gem5 started Jul 31 2019 14:48:16
gem5 executing on mars-Lenovo, pid 10862
command line: build/ALPHA/gem5.opt configs/example/se.py -c tests/test-progs/hello/bin/alpha/linux/hello

Global frequency set at 1000000000000 ticks per second
warn: DRAM device capacity (8192 Mbytes) does not match the address range assigned (512 Mbytes)
0: system.remote_gdb.listener: listening for remote gdb #0 on port 7000
warn: ClockedObject: More than one power state change request encountered within the same simulation tick

**** REAL SIMULATION ****
info: Entering event queue @ 0. Starting simulation...
info: Increasing stack size by one page.
Hello world!
Exiting @ tick 3239500 because target called exit()
marslab@mars-Lenovo:~/Desktop/Dipika/gem5$
```

So for this we will type this command. Here the gem5.opt is the gem5 binary. The simulation script that we use is se.py and the options for the simulation script can be found by typing this command in the terminal and -C provides the option for this simulation script. This can be any C code or this can be also any real benchmark. So after the successful run, you will find this as the output. This is a Hello World program and if this hello world prints it means the run was successful.

(Refer Slide Time: 14:28)

## Run gem5

4. `build/ALPHA/gem5.opt configs/example/se.py \\  
-c tests/test-progs/hello/bin/alpha/linux/hello`

**Component:**

- gem5 binary: *gem5.opt*
- options for the simulation script: *can be found using "build/ALPHA/gem5.opt config/examples/se.py --help"*
- a simulation script: *se.py,*
- options for the simulation script: *-C tests/test-progs/hello/bin/arm/linux/hello  
= -c (your\_code)*

[http://gem5.org/Main\\_Page](http://gem5.org/Main_Page)

Dipika Deb | John Jose  
MARS Research Lab, CSE@IITG



(Refer Slide Time: 14:55)

## Run gem5 (cont'd)

- ❑ Terminal output on successful run:

```
gem5 Simulator System. http://gem5.org
gem5 is copyrighted software; use the --copyright option for details.

gem5 compiled Mar 16 2018 10:24:24
gem5 started Mar 16 2018 15:53:27
gem5 executing on amarillo, pid 41697
command line: build/X86/gem5.opt configs/tutorial/simple.py

Global frequency set at 1000000000000 ticks per second
warn: DRAM device capacity (8192 Mbytes) does not match the address range assigned (512 Mbytes)
0: system.remote_gdb: listening for remote gdb on port 7000
Beginning simulation!
info: Entering event queue @ 0 / Starting simulation...
Hello world!
Exiting @ tick 507841000 because exiting with last active thread context
```

[http://gem5.org/Main\\_Page](http://gem5.org/Main_Page)

Dipika Deb | John Jose  
MARS Research Lab, CSE@IITG



You can also notice here is that it shows an entering event queue. This I have already explained that since gem5 is an event driven simulator it has an event queue where all the events are queued and de-queued as the simulation process. This line indicates that the first event was queued in the event queue at timestamp zero. So now coming to the output of the simulated machine, it can be found in the gem5/m5out directory. Next I will discuss about the output of the target machine.

(Refer Slide Time: 15:40)

## gem5 Output

- ❑ Output files from gem5 are saved in **m5out** directory.
- ❑ Files present in m5out directory are:
  - **stats.txt**: gem5 simulation statistics
  - **config.ini**: simulated machine configuration in gem5.
  - **config.json**: same as config.ini (in json format)

[http://gem5.org/Main\\_Page](http://gem5.org/Main_Page)

Dipika Deb | John Jose  
MARS Research Lab, CSE@IITG



Text file contains the statistics of the output files. Inside gem5 out directory the config.ini contains the machine configuration of the simulated machine. The config.Json is same as the config.ini, but it is in JSON format. So in step stop txt file looks somewhat like this. Here you can see that there is a sim seconds and something known as the host seconds. Sim seconds means that for how much time did the program hello world run in the simulated machine.

(Refer Slide Time: 16:17)

# Analysing gem5 Output

```

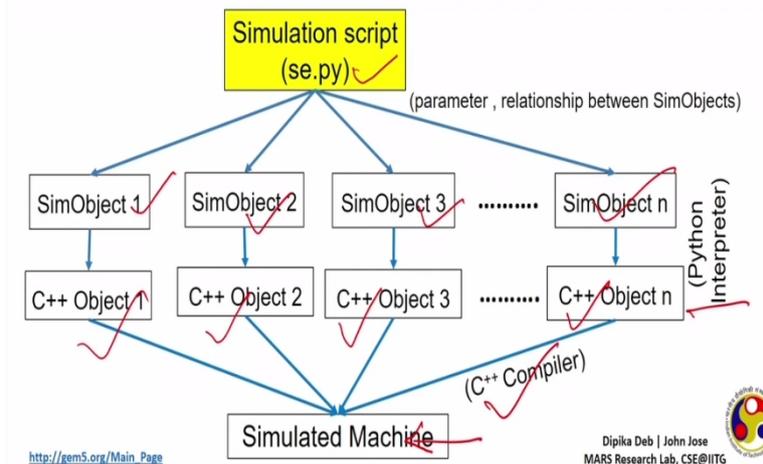
sim_seconds: 0.800348 # Number of seconds simulated
sim_ticks: 97515000 # Number of ticks simulated
final_tick: 345518000 # Number of ticks from beginning
sim_freq: 100000000000 # Frequency of simulated ticks
host_inst_rate: 144400 # Simulator instruction rate (in
host_op_rate: 260550 # Simulator op (including micro
host_tick_rate: 871825183 # Simulator tick rate (ticks/s)
host_memory: 144400 # Number of bytes of host memory
host_seconds: 0.80 # Real time elapsed on the host
sim_insts: 975 # Number of instructions simulat
sim_ops: 10314 # Number of ops (including micro

system_clk_domain.voltage_domain.voltage: 1 # Voltage in Volts
system_clk_domain.clock: 1000 # Clock period in ticks
system_mem_ctrl.pwrStateResidencyTicks::UNDEFINED: 507841000 # Cumulative time (in t
system_mem_ctrl.bytes_read::cpu.inst: 58264 # Number of bytes read from this
system_mem_ctrl.bytes_read::cpu.data: 7167 # Number of bytes read from this
system_mem_ctrl.bytes_read::total: 65431 # Number of bytes read from this
system_mem_ctrl.bytes_inst_read::cpu.inst: 58264 # Number of instructions bytes r
system_mem_ctrl.bytes_inst_read::total: 58264 # Number of instructions bytes r
system_mem_ctrl.bytes_written::cpu.data: 7160 # Number of bytes written to thi
system_mem_ctrl.bytes_written::total: 7160 # Number of bytes written to thi
system_mem_ctrl.num_reads::cpu.inst: 7283 # Number of read requests respon
system_mem_ctrl.num_reads::cpu.data: 1084 # Number of read requests respon
system_mem_ctrl.num_reads::total: 8367 # Number of read requests respon
system_mem_ctrl.num_writes::cpu.data: 941 # Number of write requests respon
system_mem_ctrl.num_writes::total: 941 # Number of write requests respon
system_mem_ctrl.bw_read::cpu.inst: 114728823 # Total read bandwidth from this
system_mem_ctrl.bw_read::cpu.data: 14112685 # Total read bandwidth from this
system_mem_ctrl.bw_read::total: 128841507 # Total read bandwidth from this
system_mem_ctrl.bw_inst_read::cpu.inst: 114728823 # Instruction read bandwidth fro
system_mem_ctrl.bw_inst_read::total: 114728823 # Instruction read bandwidth fro
system_mem_ctrl.bw_write::cpu.data: 14099901 # Write bandwidth from this memo
system_mem_ctrl.bw_write::total: 14099901 # Write bandwidth from this memo
system_mem_ctrl.bw_total::cpu.inst: 114728823 # Total bandwidth to/from this
system_mem_ctrl.bw_total::cpu.data: 28211586 # Total bandwidth to/from this
system_mem_ctrl.bw_total::total: 142940409 # Total bandwidth to/from this
  
```

While hosts second means that for how much time gem5 took to run in the host machine that is your machine. This figure summarizes the complete procedure of how gem5 works. First, the input parameters that are given to the gem5 simulator, that is se.py in the previous slide are pursued by Python interpreters. These parameters are then passed on different SimObjects. As mentioned, each of these SimObjects represents 1 physical quantity, this may represent cache, the second SimObjects may represent DRAM.

(Refer Slide Time: 16:46)

# Summarize gem5



The next one, SimObjects 3 may represent network and so on. This SimObjects are also written in python fighting. So we required a python interpreter. For example, in cache implementation in Ruby, it has a SimObject known as Ruby cache.PY. Next, the C++ objects which define this system behaviour. That is cache organization implementation of the cache inherits the values from the same objects. And finally, the system is built using GCC or G++ compiler.

And the output of the simulated machine can be found in stats.text. And the configurations that we used for the simulated machine can be seen in config.ini file, and config.JSON file. Now along with running user programs, one can also run real benchmarks to verify the behaviour of a simulated machine with respect to the real world programs. There are 2 types of benchmarks, multi programmed and multi-threaded benchmarks.

**(Refer Slide Time: 18:19)**

## Real Benchmarks

1. **Multithreaded Benchmarks:**  
threads of an application spawns across multiple cores  
PARSEC, SPLASH
2. **Multiprogrammed Benchmarks:**  
separate application for each core  
Eg: SPEC CPU 2006

[http://gem5.org/Main\\_Page](http://gem5.org/Main_Page)

Dipika Deb | John Jose  
MARS Research Lab, CSE@IITG

In multithreaded benchmarks, thread of an applications spawns across multiple cores parallel second splash are 2 such benchmarks. On the other hand, in multi program benchmarks, separate applications runs on each core like in spec cpu 2006 benchmark suit. We will use this benchmark throughout the gem5 assignments provided in the course. So next we go through a quick directory tour of gem5. Inside the gem5 directory you will find this directories. Build contains the build directory which is generated after typing Scons build command.

**(Refer Slide Time: 19:08)**

## gem5 Directory Tour

[http://gem5.org/Main\\_Page](http://gem5.org/Main_Page)

Dipika Deb | John Jose  
MARS Research Lab, CSE@IITG

**(Refer Slide Time: 19:27)**

## gem5 Directory Tour

holds files that define default settings for build different build configurations.

[http://gem5.org/Main\\_Page](http://gem5.org/Main_Page)

Dipika Deb | John Jose  
MARS Research Lab, CSE@IITG

The build opt hold files that define default settings for built different building configurations. This folder contains simulation configurations. This contains gem5 dependency, but this is not a part of actual gem5. This directory contains include files that are used to simulate and compile gem5. The output files are present in this folder. And source contains all the gem5 implementations like caches course, CPU models, interconnection models and everything.

**(Refer Slide Time: 19:34)**

# gem5 Directory Tour



❑ contains gem5 dependency but this is not a part of gem5.

[http://gem5.org/Main\\_Page](http://gem5.org/Main_Page)

Dipika Deb | John Jose  
MARS Research Lab, CSE@IITG

(Refer Slide Time: 19:44)

# gem5 Directory Tour



❑ contains include files

[http://gem5.org/Main\\_Page](http://gem5.org/Main_Page)

Dipika Deb | John Jose  
MARS Research Lab, CSE@IITG

(Refer Slide Time: 19:53)

# gem5 Directory Tour



❑ contains output files : stats.txt, config.json, config.ini

[http://gem5.org/Main\\_Page](http://gem5.org/Main_Page)

Dipika Deb | John Jose  
MARS Research Lab, CSE@IITG

(Refer Slide Time: 19:57)

## gem5 Directory Tour



contains gem5 source file.

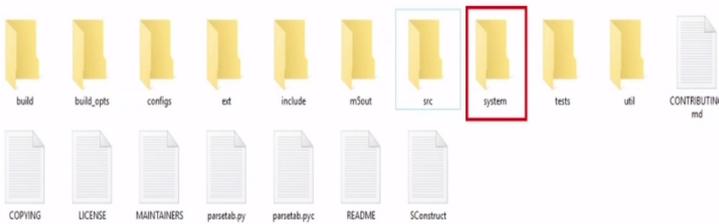
[http://gem5.org/Main\\_Page](http://gem5.org/Main_Page)

Dipika Deb | John Jose  
MARS Research Lab, CSE@IITG



**(Refer Slide Time: 20:08)**

## gem5 Directory Tour



contains low level softwares like firmware or bootloaders in simulated system.

[http://gem5.org/Main\\_Page](http://gem5.org/Main_Page)

Dipika Deb | John Jose  
MARS Research Lab, CSE@IITG



This directory contains low level softwares like firmware or boot loaders in simulated system. This directory contains files related to gem5 regression test. While this directory contains utilities scripts, our programs and useful files we will mostly deal with this folder and the config folders. All the references are mentioned in the slide. With this we come to the end of today's video for any further queries, writer us in the discussion forum of our course web page.

**(Refer Slide Time: 20:16)**

# gem5 Directory Tour



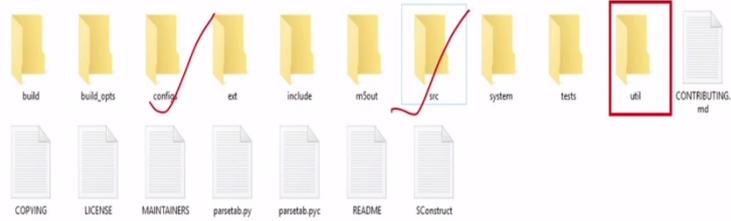
□ contains files related to gem5 regression test

[http://gem5.org/Main\\_Page](http://gem5.org/Main_Page)

Dipika Deb | John Jose  
MARS Research Lab, CSE@IITG

(Refer Slide Time: 20:22)

# gem5 Directory Tour



□ contains utility scripts, programs and useful files

[http://gem5.org/Main\\_Page](http://gem5.org/Main_Page)

Dipika Deb | John Jose  
MARS Research Lab, CSE@IITG

Thank you.