

Final Project Report Group 15-EEE 5764

Md. Habibur Rahman, UFID: 33268516,
Himanandhan Reddy Kottur, UFID: 40931757,
Sona Maria Jose, UFID: 72280930,
Shamit Gajanan Savant, UFID: 96800755

Introduction:

This paper details the simulation of multi-type CPU processors using the advanced capabilities of the Gem5 simulator. Gem5 is renowned for its modular and highly configurable nature, allowing for deep emulation of complex processor architectures. The study focuses on simulating a variety of multi-core configurations to analyze performance metrics under different scenarios. Key aspects of the research include the implementation of both functional and detailed cycle-accurate models to emulate real-world processor behavior accurately. The functional models provide a high-level overview of system performance, while the cycle-accurate models offer granular insights into the timing and operational efficiency of multi-core setups.

Our experiments cover a spectrum of parameters, including cpu type, cache hierarchies, and interconnect networks, to assess their impact on system throughput and latency. By leveraging Gem5's extensibility, we integrate custom benchmarking suites e.g. Ligr benchmark to simulate diverse workloads and stress test the multi-core processors under realistic conditions. The results highlight the effectiveness of multi-core architectures in enhancing computational performance, pinching light on potential bottlenecks and areas for optimization.

Project Outline:

- Multi-type CPU RISC-V simulation using Gem5
 - Although the RISC-V ecosystem includes functional-level, register-transfer-level, and FPGA simulation platforms, it currently lacks cycle-level simulation platforms crucial for early design-space exploration.
 - Gem5 is a widely-used cycle-level simulation platform known for its flexibility, speed, and accuracy.
 - This presentation showcases our recent advancements in simulating multi-type-CPU RISC-V systems within Gem5.
 - We evaluate the performance of the Gem5/RISC-V simulator and explore a design-space-exploration case study utilizing Gem5.

- Performance analysis of gem5/RISCV simulator and discuss design-space exploration and case study using Gem5

Changes from the original proposal: In the main proposal (inspired by the Cornell University paper), on the way of doing multi-threaded simulation, they suggested modifying the source code of gem5. We are focusing only on the application side of gem5. In our original proposal, we proposed that we would perform a multi-core simulation of the existing benchmarks, but we rather focused on performance analysis of different CPU types. Many extra benchmarks were suggested at the end of the paper, but we mainly focused on the Ligra benchmarks.

Ligra benchmark:

The Ligra benchmark is a crucial component in evaluating the performance of multi-core/multi-CPU type processors using the Gem5 simulator. Ligra, a lightweight graph processing framework, is specifically designed for shared memory systems and is used as a test binary in various simulations². When integrated with Gem5, Ligra is modified to work with different system configurations, including RISC-V architectures. This allows researchers to analyze cache behavior, scalability, and overall system performance under diverse workload¹. This comprehensive approach helps in identifying potential bottlenecks and optimizing multi-core/multi-type CPU processor designs for enhanced efficiency and performance.

Types of available simulation styles and approaches :

Functional Level Simulation

Pros:

- High-speed simulation: Functional-level simulations prioritize speed, allowing quick evaluations of system behavior.
- Verify application functionality: They are excellent for ensuring that applications compile and function as expected, identifying basic bugs and compatibility issues.

Cons:

- Lack of micro-architectural details: These simulations do not capture the intricacies of the processor's internal architecture, limiting the depth of performance insights.
- No timing accuracy: They do not provide accurate timing information, making them unsuitable for detailed performance analysis or optimization tasks.

Gem5: is this a solution?

Characteristics of Gem5

Gem5 is a versatile, open-source simulator for computer system architecture research. It supports various instruction set architectures (ISAs), including RISC-V, ARM, and x86. With its detailed simulation capabilities, Gem5 allows users to analyze hardware performance at multiple levels, from single instructions to entire system interactions.

Initial RISC-V Port in Gem5

RV64GC: The initial RISC-V port supports the 64-bit general-purpose RV64GC ISA.

Single-Core Simulation System: It began with single-core simulations.

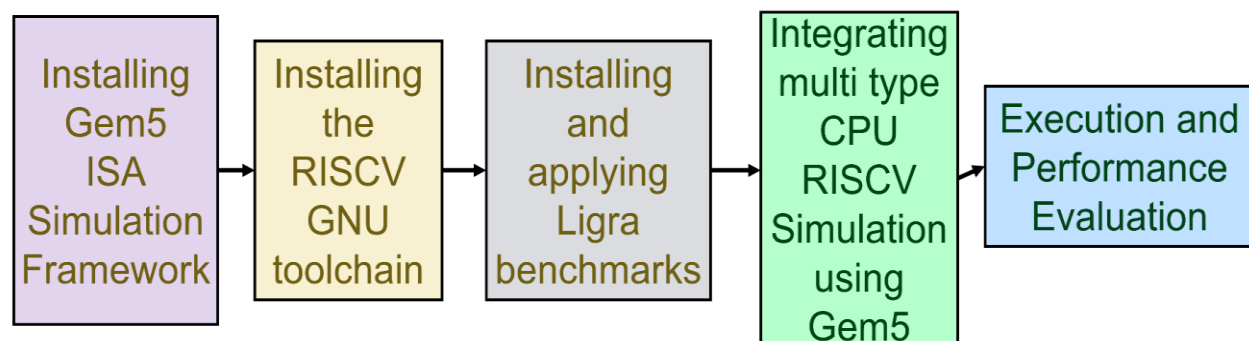
Contributions to the RISC-V Port

Multi-core/ cpu type System Simulation in SE Mode: Enhanced SE mode to support multi-core simulations, enabling concurrent core performance analysis.

RISC-V Toolchain and its implications on Gem5

The **RISC-V toolchain** is an essential suite of software tools designed to support the development and execution of applications on RISC-V architectures. It includes cross-compilers, assemblers, linkers, and libraries that enable developers to build, debug, and optimize code specifically for RISC-V systems. Developers can compile RISC-V applications with the toolchain and simulate their performance in Gem5, enabling thorough analysis and optimization of both hardware and software. This synergy significantly enhances the ability to test new RISC-V architectures and improve system performance.

Step-by-step procedure of our methodology



HW-SW Threading: In Gem5, each CPU features multiple hardware (HW) threads. When an application is executed, each software (SW) thread is allocated to a specific HW thread. These HW threads are tasked with maintaining the state of their assigned SW threads, which includes the program counter (PC), register values, and the activity status of the SW thread. The mechanisms for creating, synchronizing, and terminating threads depend on three key system calls: clone, futex, and exit. Thus, to run multi-threaded applications in System Emulation (SE) mode, it is crucial to support these system calls.

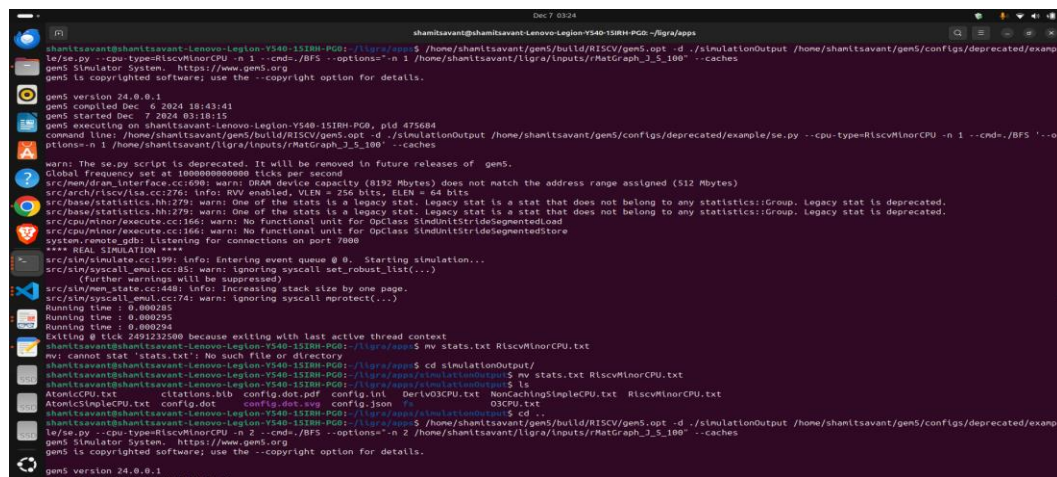
Validation models:

Instead of using C/C++ benchmarks to validate a model in Gem5, we opted for comprehensive, carefully designed assembly and low-level C unit tests. These tests, written in assembly language, focus on individual instructions or system calls, avoiding the added complexities of C/C++ libraries and compilers. We utilized low-level C unit tests to uncover missing functionalities in real-world libraries, such as the GNU thread library.

Code Snippet:

```
Install all dependencies ← % # Get all software dependencies
                           % sudo apt-get install scons python-dev m4 autoconf automake autotools-dev curl libmpc-dev libmpfr-dev
                           % # Download and build gem5
                           % cd $HOME && git clone https://gem5.googlesource.com/public/gem5 && cd gem5
                           % # Skip this step when this change is fully merged in upstream gem5
                           % git pull https://gem5.googlesource.com/public/gem5 refs/changes/26/9626/4
                           % # skip this step when this change is fully merged in upstream gem5
                           % git pull https://gem5.googlesource.com/public/gem5 refs/changes/44/9644/3
                           % scons build/RISCV/gem5.opt -j8
Gem5 Installation ← % # Download and build RISC-V GNU toolchain
                    % cd $HOME && git clone --recursive https://github.com/riscv/riscv-gnu-toolchain
                    % cd riscv-gnu-toolchain/ && mkdir ./build && cd ./build
                    % ../configure --prefix=$HOME/riscv-gnu-toolchain/build/
                    % make linux -j8
                    % export PATH=$PATH:$HOME/riscv-gnu-toolchain/build/bin/
RISCV-Toolchain Installation ← % # Download and build Ligra applications
                              % cd $HOME && git clone https://github.com/jshun/ligra.git
                              % cd $HOME/ligra/ligra/
                              % # Modify Ligra to work with gem5
                              % mv ligra.h ligra.h.old
                              % sed 's/long rounds/a int num_cpu = P.getOptionIntValue("-n",1); setWorkers(num_cpu);' ligra.h.old >
                              % ligra.h
Ligra Installation ← % cd $HOME/ligra/apps/
                    % ln -s $HOME/ligra/ligra/* .
                    % riscv64-unknown-linux-gnu-gcc -static -fopenmp -DOPENMP -Wall -O0 -I. -c BFS.C -o BFS.o
                    % riscv64-unknown-linux-gnu-g++ -static -DOPENMP -L. -o BFS BFS.o -lgomp -lpthread -ldl
Running an application in modified Gem5 ← % # Run BFS on gem5
                                          % $HOME/gem5/build/RISCV/gem5.opt $HOME/gem5/configs/example/se.py --cpu-type DerivO3CPU -n 4 -c ./BFS.o
                                          "n 4 ./inputs/rMatGraph.J5.100" --caches
```

Simulation Results:



The result shows that gem5's performance scales well with the number of simulated CPU cores. When simulating more CPU cores, gem5 does slow down a little bit since it simulates more thread communication events between cores.

Sample stat file:

```
----- Begin Simulation Statistics -----
simSeconds          0.001093          # Number of seconds simulated (Second)
simTicks            1093490500         # Number of ticks simulated (Tick)
finalTick           1093490500         # Number of ticks from beginning of simulation (restored from checkpoints and never reset) (Tick)
simFreq             1000000000000      # The number of ticks per simulated second ((Tick/Second))
hostSeconds         12.84              # Real time elapsed on the host (Second)
hostTickRate        85166818           # The number of ticks simulated per host second (ticks/s) ((Tick/Second))
hostMemory          713928             # Number of bytes of host memory used (Byte)
simInputs           3251468            # Number of instructions simulated (Count)
simOps              3251782            # Number of ops (including micro ops) simulated (Count)
hostInstRate        253238            # Simulator instruction rate (inst/s) ((Count/Second))
hostOpRate          253264            # Simulator op (including micro ops) rate (op/s) ((Count/Second))
system_clk_domain.clock 1000        # Clock period in ticks (Tick)
system.cpu.numCycles 2186994          # Number of cpu cycles simulated (cycle)
system.cpu.cpi       0.672622          # CPI: cycles per instruction (core level) ((cycle/count))
system.cpu.ipc       1.486720          # IPC: Instructions per cycle (core level) ((Count/cycle))
system.cpu.numWorkItemsStarted 0        # Number of work items this cpu started (Count)
system.cpu.numWorkItemsCompleted 0      # Number of work items this cpu completed (Count)
system.cpu.instsAdded 3731014          # Number of instructions added to the IQ (excludes non-spec) (Count)
system.cpu.nonSpecInstsAdded 1124      # Number of non-speculative instructions added to the IQ (Count)
system.cpu.instsIssued 3618081        # Number of instructions issued (Count)
system.cpu.squashedInstsIssued 1151    # Number of squashed instructions issued (Count)
system.cpu.squashedInstsExamined 480355 # Number of squashed instructions iterated over during squash; mainly for profiling (Count)
system.cpu.squashedOperandsExamined 210242 # Number of squashed operands that are examined and possibly removed from graph (Count)
system.cpu.squashedOpRate 303          # Number of squashed non-spec instructions that were removed (Count)
system.cpu.numIssuedDist:mean 1893922  # Number of insts issued each cycle (Count)
system.cpu.numIssuedDist:stddev 1.919780 # Number of insts issued each cycle (Count)
```

Figure: Sample stat file for deriveO3CPU

Exploration of different CPU designs and their performance analysis:

Metric	At1.txt	AtomeS1.txt	Deriv1.txt	Metric	At1.txt	AtomeS1.txt	NonC1.txt
Simulated Time (s)	0.001958	0.001958	0.001093	simSeconds	0.001958	0.001958	0.001958
Simulated Ticks	1,957,934,500	1,957,934,500	1,093,490,500	simTicks	1,957,934,500	1,957,934,500	1,957,934,500
Host Time (s)	2.20	2.22	12.84	finalTick	1,957,934,500	1,957,934,500	1,957,934,500
Host Tick Rate	891,455,717 ticks/s	882,030,350 ticks/s	85,166,818 ticks/s	hostSeconds	2.20	2.22	2.53
Host Inst Rate	1,480,302 inst/s	1,464,612 inst/s	253,238 inst/s	hostTickRate	891,455,717	882,030,350	775,133,802
Host Op Rate	1,480,450 ops/s	1,464,758 ops/s	253,264 ops/s	hostMemory (Bytes)	707,776	707,780	707,776
Instructions Simulated	3,251,431	3,251,431	3,251,448	simInsts	3,251,431	3,251,431	3,251,431
Operations Simulated	3,251,765	3,251,765	3,251,782	simOps	3,251,765	3,251,765	3,251,765
Cycles Simulated	3,915,870	3,915,870	2,186,994	hostInstRate	1,480,302	1,464,612	1,287,132
CPI	1.204	1.204	0.673	hostOpRate	1,480,450	1,464,758	1,287,261
IPC	0.830	0.830	1.487	system.cpu.numCycles	3,915,870	3,915,870	3,915,870
Memory Used (Bytes)	707,776	707,780	713,928	system.cpu.cpi	1.204309	1.204309	1.204309
Integer Instructions	3,220,015	3,220,015	Not available in this file	system.cpu.ipc	0.830352	0.830352	0.830352
Float Instructions	166	166	Not available in this file	system.cpu.numFpInsts	166	166	166
Load Instructions	823,814	823,814	Not available in this file	system.cpu.numIntInsts	3,220,015	3,220,015	3,220,015
Store Instructions	442,884	442,884	Not available in this file	system.cpu.numLoadInsts	823,814	823,814	823,814
				system.cpu.numStoreInsts	442,884	442,884	442,884

At1.txt refers to AtomicCPU, AtomeS1.txt refers to AtomicSimpleCPU and Deriv1.txt refers to DerivO31CPU

In this study, we analyzed the performance of the Breadth-First Search (BFS) algorithm from the Ligma framework on various CPU architectures under the RISC-V64 platform using the gem5 simulator. The BFS program was compiled under Ligma/apps and tested with the dataset rMatGraph_J_5, ensuring consistency across all simulations. The CPU models evaluated were AtomicSimpleCPU (AtomS1), DerivO3CPU (Deriv1), NonCachingSimpleCPU, O3CPU (O3p1), and RiscvMinorCPU. Due to gem5 constraints, the simulations were conducted using a single core. To ensure statistical robustness, each CPU type was simulated 100 times, and the results were averaged. The accompanying bash script automated this process, generating comprehensive performance metrics stored in stats.txt files for each configuration.

The comparison focused on key performance metrics such as Simulated Time, Simulated Ticks, Host Time, Host Tick Rate, Host Instruction Rate, Cycles Per Instruction (CPI), Instructions Per Cycle (IPC), Number of CPU Cycles, and Number of Instructions. A summary of the results revealed significant architectural differences. For instance, AtomS1 achieved a Host Instruction Rate of 1,464,612 instructions per second and an IPC of 0.830, while Deriv1 and O3p1 demonstrated higher efficiencies with identical IPCs of 1.487 and Host Instruction Rates of 253,238 and 265,901, respectively. Additionally, Simulated Time was consistent at 0.001093 seconds for Deriv1 and O3p1, outperforming AtomS1's 0.001958 seconds. These results reflect the performance trade-offs among CPU types, with AtomS1 prioritizing simplicity over efficiency, while Deriv1 and O3p1 showcased advanced pipeline optimizations for improved execution rates.

This study highlights the distinct performance characteristics of RISC-V64 CPU architectures for BFS workloads, offering insights into their computational efficiency and suitability for graph processing.