Simulating Multi-Core
RISC-V Systems in gem5

Tuan Ta, Lin Cheng, and Christopher Batten

School of Electrical and Computer Engineering
Cornell University

2nd Workshop on Computer Architecture Research with RISC-V
June 2018
### Task-Parallel System Design Space Exploration

#### Task-Parallel Runtimes
- OpenMP, Cilk, Intel TBB, etc.
- Static, Dynamic, Adaptive Task Scheduling, etc.
- Work-Stealing, etc.

#### Multi-Core Systems
- In-order superscalar cores
- Out-of-order cores
- Heterogeneous big.LITTLE system

---

### Applications
- Graph-processing application domain
- Irregular parallelism
- Ligra graph framework [J. Shun, PPoPP 2013]

**Many design points to consider!**
What Tools Are Available in RISC-V Ecosystem?

Functional-Level Simulators: Spike & QEMU

Pros
▶ Very fast simulation
▶ Verify applications compile and work correctly

Cons
▶ Capture no micro-architectural details
▶ Not timing accurate
What Tools Are Available in RISC-V Ecosystem?

RTL Simulators: Rocket & BOOM RTL models

Pros
▸ Provide low-level micro-architectural details
▸ Cycle-accurate

Cons
▸ Too slow to run many different simulations
  ▸ Simulate at the rate of 4,000 instructions per second
  ▸ Take 3 days to run a small application
▸ Limited to single-threaded application and single-core system
  ▸ Use a single-threaded proxy kernel
  ▸ Boot a full Linux image → not a practical solution!
▸ Limited to existing RISC-V RTL models
What Tools Are Available in RISC-V Ecosystem?

FPGA

Pros
- Fast execution
- Timing accurate
- Can boot a full Linux image

Cons
- Require physical FPGA boards
- Lengthy synthesis, place and route process
- Limited to existing RISC-V RTL models
Is gem5 a Solution?

What is gem5?

- Multiple ISAs
- Multiple processor models
- Multiple memory and network models
- Some advanced simulation features
- Strong support from gem5 developer and user community
Is gem5 a Solution?

Initial RISC-V port in gem5 [A. Roelke, CARRV 2017]
- RV64GC
- Single-core system simulation
- System call emulation (SE) mode

Our contribution to RISC-V port in gem5 [CARRV 2018]
- Multi-core system simulation in SE mode
- RISC-V testing infrastructure in gem5
Everything Is Open-Source!

% # Get all software dependencies
% sudo apt-get install scons python-dev m4 autoconf automake autotools-dev curl libmpc-dev libmpfr-dev
libgmp-dev gawk build-essential bison flex texinfo gperf libtool patchutils bc zlib1g-dev libexpat-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
% # 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/
% # 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 '/long rounds/a int num_cpu = P.getOptionIntValue("-n",1); setWorkers(num_cpu);' ligra.h.old > ligra.h
% cd $HOME/ligra/apps/
% ln -s $HOME/ligra/ligra/* .
% riscv64-unknown-linux-gnu-gcc -static -fopenmp -DOPENMP -Wall -00 -I. -c BFS.C -o BFS.o
% riscv64-unknown-linux-gnu-g++ -static -DOPENMP -L. -o BFS BFS.o -lgomp -lpthread -ldl
% # 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
We Can Explore Task-Parallel System Design Space!

Task scheduling policies

<table>
<thead>
<tr>
<th>Library</th>
<th>Static</th>
<th>Guided</th>
<th>Work stealing</th>
</tr>
</thead>
<tbody>
<tr>
<td>OMP-S</td>
<td>Fixed</td>
<td>Static</td>
<td>No</td>
</tr>
<tr>
<td>OMP-G</td>
<td>Adaptive</td>
<td>Dynamic</td>
<td>No</td>
</tr>
<tr>
<td>Cilk-WS</td>
<td>Fixed</td>
<td>Dynamic</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Heterogeneous system

In-order Cores

Out-of-order Cores

Shared Memory

Ligra graph-processing applications
We Can Explore Task-Parallel System Design Space!

- OMP-G and Cilk-WS are designed to balance workload between heterogeneous cores
- OMP-G and Cilk-WS offered better throughput in most of Ligra applications
- gem5 simulated all Ligra apps at the speed of 175 KIPS (vs. 4 KIPS if using Chisel C++ RTL simulator)
Multi-Core RISC-V Support in gem5

- Thread-managing system calls
- Synchronization instructions
- Release consistency
Multi-Core RISC-V Support in gem5

Thread-managing system calls

- clone
- futex
  - FUTEX_WAIT
  - FUTEX_WAKE
- exit
Multi-Threading in gem5 System Call Emulation

- System Call Emulation (SE)
  - No OS code is simulated
  - All system calls are emulated

- Software thread (SWT)
  - User-level thread

- Hardware thread (HWT)
  - Execution unit (e.g., CPU core)

- SWT - HWT mapping
  - Done by gem5
  - SWT can be mapped to and unmapped from a HWT
  - HWT maps to at most one SWT at a time
  - No SWT context switching
Clone System Call

- Spawn a new SWT
- gem5 finds a free HWT for the new SWT
- gem5 initializes and allocates resources for the new SWT
  - Copy pointers to shared resources (e.g., page table) from the parent to the child SWT
  - Allocate non-shared resources (e.g., stack and thread-local storage)
- gem5 activates the HWT
- Supported RISC-V clone system call interface in gem5 SE
- Initialized RISC-V registers upon clone system call
futex System Call

- Synchronize threads using user-level futex variables
  - `FUTEX_WAIT`: put calling threads into sleep
  - `FUTEX_WAKE`: wake up threads waiting on a futex variable

- gem5 maintains a list of HWTs waiting on each futex variable

- gem5 suspends a HWT when it goes to sleep

- gem5 resumes execution of a HWT when it is waken up by `FUTEX_WAKE`

- Supported some variants of `FUTEX_WAIT` and `FUTEX_WAKE`

- Fixed bugs in how HWT is suspended and resumed in all CPU models in gem5
exit System Call

- Terminate a running SWT
- gem5 cleans up micro-architectural states of the terminating SWT
- gem5 unmaps SWT from HWT and frees up the HWT
- Fixed bugs in thread termination in all CPU models in gem5
Multi-Core RISC-V Support in gem5

Thread-managing system calls

Synchronization instructions

Release consistency

Synchronization instructions

- AMO
- LR & SC
Atomic Memory Operation Instructions

- Added new AMO memory request type to all CPU models
- AMO requests carrying AMO operations are issued to memory system like normal LOAD and STORE requests
- Modified gem5 cache models to execute AMO operations directly in L1 caches

1. AMO request
2. Exclusive memory fetch
3. In-L1 AMO processing
4. AMO response

CPU 0

CPU 1

L1$

L1$

Shared Mem

(1) AMO request

(2) Exclusive memory fetch

(3) In-L1 AMO processing

(4) AMO response
Load-Reserved & Store-Conditional Instruction

- Address reservation list per HWT
  - Load-reserved
    - Invalidate any active reservation of target variable through memory coherence bus
    - Put the variable in reservation list
  - Store-conditional
    - Succeed if target variable is still being reserved
    - Otherwise, fail
- Livelock prevention
  - Defer invalidation requests in L1 cache in a bounded period of time
Multi-Core RISC-V Support in gem5

Thread-managing system calls

Synchronization instructions

Release consistency
Release Consistency

- Break \texttt{amo}, \texttt{lr}, and \texttt{sc} instructions into micro-operations
- Insert \texttt{fence} micro-operations to ensure correct memory orderings

\begin{verbatim}
amoadd.aq  amoadd.rl  amoadd.aqrl
amoadd    fence    amoadd
fence     amoadd   fence
\end{verbatim}
Functional Validation

- Did not exist in gem5 before

- Single-threaded testing
  - Ported RISC-V assembly test suite into gem5

- Multi-threaded testing
  - Built a minimal threading library in assembly
  - Tested individual system calls
  - Tested individual synchronization instructions

- pthread functionality testing
  - Detected missing functionality used by GNU pthread library
  - Tested commonly used pthread functions (e.g., pthread_create, pthread_join, pthread_mutex_lock, etc.)
Timing Validation

- CPU models in gem5 are generic and NOT validated against an actual microarchitecture

- We validated gem5’s multiplier model against an iterative multiplier in Rocket chip
  - Used a micro-benchmark that executed 500 \texttt{mul} instructions back-to-back
  - No RAW dependency between these \texttt{mul} instructions
  - No loop to minimize interference from branch predictor
  - Warmed up instruction cache
  - Measured the CPI of the 500 \texttt{mul} instruction sequence in both gem5 and Rocket models
  - Adjusted gem5 multiplier’s configuration

- Similar approach can be applied to validate other HW units (e.g., floating point unit, branch predictor, etc.)
Take-Away Point

- Multi-threaded RISC-V binaries can run on gem5 out of the box
- gem5 is a good cycle-level modeling tool for **efficient** early system design space exploration
- RISC-V port development in gem5
  - Initial RISC-V port in gem5 [A. Roelke, CARRV 2017]
  - Our contribution to RISC-V port in gem5 [CARRV 2018]
  - Future contributions from RISC-V and gem5 community ...

![RISC-V and gem5 logos]

This work was partially supported by the NSF, AFOSR, SRC, and donations from Intel