QPoints: QEMU to gem5 ARM Full System Checkpointing

PRINCETON UNIVERSITY
* Northwestern University

Bhargav Reddy Godala, Ishita Chaturvedi, Yucan Wu, Simone Campanoni*, David I August

THE LIBERTY RESEARCH GROUP
ARCANA Rethinks Compiler Abstractions for New Applications

Introduction

- Simulating long-duration workloads on gem5 takes **prohibitively long time**.
- Even with gem5's fastest Atomic model, reaching a steady state can be a time-consuming task.

Application / Host Runtime	# Instruction (Billions)	Approximate Simulation Time
Linux Boot	2.4 B	~24 min
1 second	2 B	~20 min
1 minute	120 B	~20 hours
10 minutes	1200 B	~8 days

Approx. simulation time with gem5's Atomic CPU

Workflow of creating gem5 checkpoints using QEMU

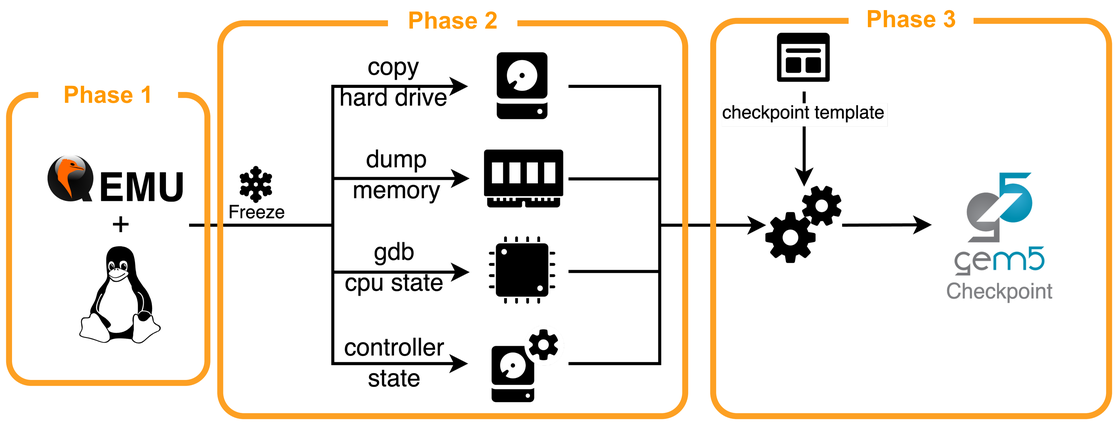


Figure 1. Workflow of creating checkpoint on QEMU running Linux System

Results

We collected checkpoints of 13 benchmarks using Apple's **M1 Mac mini (HVF acceleration)**

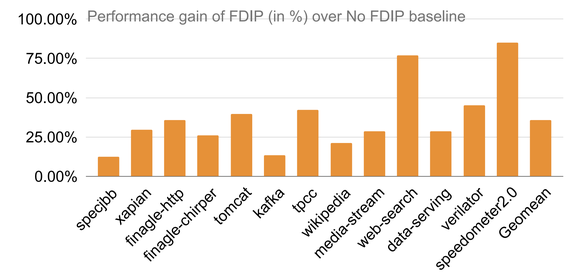
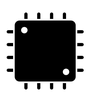
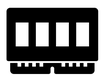


Figure 2. Performance gain of FDIP over No FDIP baseline of 13 workloads

Checkpoint State


CPU Registers


Physical Memory


Hard Disk Image


Controller State

A minimal subset of hardware state needed to create resumable checkpoint

Platform Configuration

- Device mapping in gem5 achieved through QEMU Virtual Configuration.
- The configuration is exposed as new system configuration.
- GIC v2 is enabled to support Apple M1 Checkpoints.
- Disk is attached as VirtIO device.
- The subset of VirtIO features supported by gem5 should be used during QEMU emulation.

Steps to create a checkpoint

Phase 1: Emulate disk image using QEMU
- Enable hardware acceleration **KVM** or **HVF** if available
- From console run the benchmark
- Wait for it to reach region of interest

Phase 2: Extract system state
- Freeze the system state from QEMU Monitor
- Attach gdb to emulated system and collect CPU register state
- Make copy of disk image
- Dump physical memory file
- Extract disk interface state

Phase 3: Create gem5 checkpoint
- Process raw files from phase 2 using template file to create gem5 compatible checkpoint

Limitations

- Supports ARM 64-bit platforms only.
- No symbol table mapping.
  - Makes debugging really hard.
- gem5's COW and QEMU's QCOW2 image formats are not supported.
  - Each checkpoint contains full disk image

Features

- *Expanding Horizons*: Hardware acceleration support extended beyond Linux to include platforms like **Apple's HVF**, an equivalent of **KVM**.
- *Efficiency in Sync*: Support for gem5 compatible **multi-core checkpoints**.

Platforms and Features

Advantages

- Capability to execute intricate software with various needs
  - Handles heavy runtime environments.
  - All system calls are supported.
- Near native wait times.
- Elimination of the need for building gem5 on ARM system and faster QEMU emulation.
- Support for ARM-powered **M1 Mac** and free hardware acceleration beyond **KVM**.
- **No changes to the QEMU source code required**!

Future Work

- Support m5 like utility.
- X86 Platform Support.
- Support for compressed disk image formats like QCOW2.
- Enable network device support.

Published Works

- N. P. Nagendra, B. R. Godala, I. Chaturvedi, A. Patel, S. Kanev, T. Moseley, J. Stark, G. A. Pokam, S. Campanoni, and D. I. August, "**Emissary**: Enhanced Miss Awareness Replacement policy for l2 instruction caching," in Proceedings of the 50th Annual International Symposium on Computer Architecture (ISCA '23), June 17–21, 2023, Orlando, FL, USA, 2023

Links

QPoints:
https://github.com/PrincetonUniversity/QPoints

gem5:
https://github.com/PrincetonUniversity/gem5_FDIP

QPoints gem5 EMISSARY