Cycle-Accurate HSA Simulator

Chien-Chih Chen
Advisor: Tien-Fu Chen

SoC & ESW Lab

國立交通大學
National Chiao Tung University
2015/5/18
Outline

- Why do we need a fast cycle-accurate HSA simulator
- Basics of SW simulator
- Our event-driven architectural simulator w/ QEMU
- Our fast cycle-accurate HSA simulator
Heterogeneous multicore issues

Compute Engine

Graphic Engine

Asymmetric memory architecture

Cache hierarchy management

LPDDR

DDR2
We need a simulator

- Simulator is models computer devices (or components) to predict outputs and performance metrics on a given input.

- Used for three aspects
  - Application development, System verification, System architecture exploration, System performance evaluation

---

Guest App 0
Guest App 1
Guest O.S.
HSA Driver

Virtual Platform
Emulation core (QEMU)
Simulation core
Vertical Prof.

Host O.S.
Host

---

CPU
CPU
GPU
GPU
Private $, Shared $,
$ coherence, Local memory,
Interconnect, DRAM, …

---

Statistics
# Different Abstraction Levels

<table>
<thead>
<tr>
<th>Abstraction Level</th>
<th>Simulation Throughput</th>
<th>CPU</th>
<th>Cycle Accurate</th>
<th>Bus model</th>
<th>Software</th>
<th>Target OS</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Only</td>
<td>1~2 GIPS</td>
<td>None</td>
<td>No</td>
<td>No</td>
<td>C, C++</td>
<td>none</td>
<td>30 s</td>
</tr>
<tr>
<td>ISS simulation</td>
<td>1~10 MIPS</td>
<td>ISS</td>
<td>Maybe</td>
<td>possible</td>
<td>C, C++</td>
<td>possible</td>
<td>5 hr</td>
</tr>
<tr>
<td>SystemC simulation</td>
<td>10~100 KIPS</td>
<td>cycle-accurate model</td>
<td>Maybe</td>
<td>TLM/timed model</td>
<td>C, C++ assembly</td>
<td>almost not practical</td>
<td>18 hr</td>
</tr>
<tr>
<td>HDL Emulation</td>
<td>10~1000 IPS</td>
<td>HDL</td>
<td>Yes</td>
<td>Yes</td>
<td>assembly</td>
<td>Not Practical</td>
<td>1200 days</td>
</tr>
<tr>
<td>FPGA Emulation</td>
<td>50~100 MIPS</td>
<td>Real or hard core</td>
<td>Yes</td>
<td>Yes</td>
<td>C, C++ assembly</td>
<td>real</td>
<td>10 m</td>
</tr>
<tr>
<td>Silicon Prototype</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Actual</td>
<td></td>
</tr>
</tbody>
</table>
Abstraction and Accuracy

Pre-silicon embedded software development

ISS+turbo

ISS

FPGA

SystemC

System integration and verification

HDL-Verilog, VHDL

Architectural exploration and system analysis

Functionally accurate

Cycle approximate

Cycle accurate

CA (Cycle Accurate)  PVT (Programmer View with Timing)  PV (Programmer View)
Categories of SW Simulators

- **Scope**: micro-architecture vs. full-system
  - Only one microprocessor or the whole computer system.

- **Detail**: functional vs. timing (or performance) simulators.
  - Functional simulators emphasize achieving the same function as the modeled components (what is done)
  - Timing simulators strive to accurately reproduce the performance/timing features (when is it done) of the targets in addition to their functionalities.

- **Input**: trace-driven vs. execution-driven simulators.
  - Traces are pre-recorded streams of instructions with some fixed input.
  - Execution-driven simulators allow dynamic change of instructions to be executed depending on different input data.
QEMU
A full-system functional simulator
What is the Qemu?
- A generic and open source machine emulator and virtualization machine.

Feature
- Simulation speed is fast!!! (DBT)
- Support many ISAs
- Portable: can run on different machines
- Two mode: user-level simulation & Full-system simulation
- Open source
How to support many ISAs on different machines?

- **TCG (Tiny Code Generator)**
  - A generic backend for a C compiler.

```
| ARM | x86 | MIPS | .......... | ISA_n |
```

IR (intermediate representation) code

```
| ARM | x86 | MIPS | .......... | ISA_n |
```

Target(Guest) code

```
Target Code
  | ARM code |
|------|---------|

Host code

```
Host Code
  | x86 code |
|------|---------|
```

TCG Operations

```
  | IR code |
|------|---------|
```

gen_intermediate_code()
tcg_gen_code()
How to translate the ISA₁ to ISA₂?

- **DBT (Dynamic binary translation)**
  - Looks at a short sequence of code (usually a basic block) then translates it and caches the resulting sequence.

- **TB (translation block)**
  - The translation unit in Qemu. (as a basic block)

```
mov    %esp,%eax
mov_i32 tmp0,esp
mov_i32 eax,tmp0 
mov    0x10(%ebp),%ebx
mov    0x10(%ebp),%esi
```
Acceleration during translated code

- **Code cache**
  - Store every translated code in cache and reuse it.

- **Chain**
  - Link one TB to next TB reduce trips to the manger.
CMP$im

A trace-driven timing simulator
**Feature**
- Use PIN to conduct exploratory memory performance studies.

**Pros**
- Native execution, faster speed
- Flexible memory module

**Cons**
- Limit on host ISA.
- Only can use on application level-based simulation.
- Only used on memory studies
- Low-scalability
Multi2Sim
A user-level timing simulator
What is Multi2Sim

- A simulation framework for CPU-GPU heterogeneous computing
  - CPU: X86
  - GPU (OpenCL): Southern-Island
- Application-only emulation
- It provides models for
  - Superscalar, multithreaded, and multicore CPUs, and GPU architectures
  - Memory hierarchies
Multi2Sim simulation flow

- Three main components
  - Functional simulator (emulator)
  - Detailed simulator (pipeline simulation)
  - Performance model (Cache hierarchy, prefetch)
- Functional simulator
  - Map the guest execution instruction to the implementation function which can run on the host
- Detailed simulator
  - Calculate timing (data hazard, control hazard)
- Performance model
  - Calculate timing (structural hazard)
Instruction emulation in simulator

- Emulation of x86 instructions
  - Update x86 registers.
  - Update memory

- Emulation of Linux system calls
  - Instruction “int 0x80” is used to perform a system call
  - Emulate the system call according to “system call code” and its arguments.

1. Read instr.
2. Decode
3. Read int 0x80
   - no: Simulate x86 instr.
   - yes: Emulate system call
4. Next IP
The memory hierarchy

- **Flexible hierarchies**
  - Any number of caches organized in any number of levels.
  - Each architecture with a timing simulation specifies its own entry point (cache memory) in the memory hierarchy, for data or instructions.
  - Cache coherence is guaranteed with an implementation of the 5-state MOESI protocol.

- **Event queue support**
  - Only simulate when the simulation time arrive the event trigger time
• mod_access (return: event_id)
  • for creating an event on Memory hierarchy
  • ACCESS_MODULE, ACCESS_TYPE, ADDRESS

• mod_in_flight_access
  • Check event finished or not
  • ACCESS_MODULE, EVENT_ID, ADDRESS
Gem5
A full-system timing simulator
What is Gem5

- Integrate two simulators
  - M5: CPU models, ISAs, I/O devices, infrastructure
  - Gems (Ruby): cache coherence protocols, interconnect model
Features

- Execution modes: System-call Emulation (SE) & Full-System (FS)
- ISAs: Alpha, ARM, MIPS, Power, SPARC, x86
- CPU models: AtomicSimple, TimingSimple, InOrder, and O3
- Cache coherence protocols: broadcast-based, directories, etc.
- Interconnection networks: Simple & Garnet (Princeton, MIT)
- Devices: NICs, IDE controller, etc.
- Multiple systems: communicate over TCP/IP
mov_rm8_imm8

```c
void x86_isa_mov_rm8_imm8_impl(X86Context *ctx)
{
    unsigned char value = ctx->inst.imm.b;
    X86ContextStoreRm8(ctx, value);
    x86_uinst_new(ctx, x86_uinst_move, 0, 0, 0, x86_depp_rm8, 0, 0, 0);
}
```
Speed and Accuracy of ISS

- **Architectural Simulators** (e.g., SimpleScalar, SMITsim)
  - Minimal functionality
  - Accurate timing

- **Interpreted Emulators** (e.g., Bochs, SIMICS)
  - Full timing and (µ)arch details

- **Fast Emulators** (e.g., QEMU, SimNow™)
  - Full functional, memory and system details; simple timing
  - No system details, no memory paths

- **Virtual Machines** (e.g., VMware, Virtual PC)
  - Native virtualization, direct execution
  - Accurate functionality
  - Minimal timing

---

[A. Falcon et al, ISPASS, 2007]
Sampling simulation

[DAC 14’]DAPs: Dynamic Adjustment and Partial Sampling for Multithreaded/Multicore Simulation

Chien-Chih Chen, Yin-Chi Peng, Cheng-Fen Chen, Wei-Shan Wu, Qinghao Min, Pen-Chung Yew, Weihua Zhang, Tien-Fu Chen
Timing model dominates simulation time

- Simulation can be decomposed into two models
  - Functional model (FM)
  - Detailed timing model (TM)
- Simulation time increases non-linearly as core no. increases
- Detailed TM dominates simulation time
- Statistical sampling can improve simulation time
Sampling Mechanism

- Simulate a **small representative** subset of program execution
- TM only executes on sampling point

**Sampling techniques**

- **Static sampling**
  - Select regular sampling points before run-time

- **Dynamic sampling**
  - Decide sampling points at run-time
Challenges in Multithreaded/Multicore simulation

- Synchronization events would cause simulation time to vary:
  - Static sampling could not capture program variations.

- In multithreaded application, threads don’t run simultaneously:
  - Synchronized core sampling cannot capture synchronization events.

![Graph showing IPC vs. M cycles for CPU0 and CPU1.](graph.png)
Hybrid sampling mechanisms for Multithreaded/Multicore simulation

- Regular sampling
  - As the conventional sampling

- Aggressive sampling
  - Capture thread synchronization primitives
  - Frequently simulate to resist the variation
  - Simulation accuracy ↑

- Lazy sampling
  - Rarely simulate the repeated phases
  - Simulation speed ↑
The DAPs framework

- Centralized sampling controller
  - Manage the sampling parameters and control the TM
- Synchronization monitor: capture synchronization primitives
- Partial selection: let each core has its own sampling switcher
- Locality-phase detection: distinguish the executed phase
Results: simulation speedup

- Mark the lazy sampling for a high priority
  - Recover the simulation speed for the aggressive sampling
- More than 16x faster than fully-timing simulation.
- The simulation speed of DAPs is 1.5x~1.6x faster than TBS.
- In fluidanimate, streamcluster, swaptions
  - DAPs has a gain of simulation speed via lazy sampling.
Event-driven timing simulation w/ QEMU
Challenges of trace-driven simulation

- Execution flow isn’t affected by structural hazard.

![Diagram showing execution flow in QEMU with trace driven simulation.]
Design Challenges

- **Benefit of Qemu**
  - Support full-system
  - Enough Simulation speed
  - Stability

- **Difficulty of Qemu**
  - Complex code translation
  - No support for performance model trace
  - Need to affect the progress of application

- **Difficulty of Large-scale architecture simulation**
  - Communicate between concurrent Qemu simulations
  - Synchronization overhead

---

**Performance model requirement**

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction count</td>
<td>Instruction address (memory access)</td>
</tr>
<tr>
<td>Data address (memory access)</td>
<td>Program counter</td>
</tr>
<tr>
<td>Multi-core simulation sequence</td>
<td>Performance model parallelism</td>
</tr>
<tr>
<td>Retrieve trace more effectively</td>
<td>Performance model flexibility</td>
</tr>
</tbody>
</table>
Skip the stalled core

QEMU

LD A
LD B
LD C
LD D

Trace
Trace
Trace
Trace

L1 $
L1 $
L1 $
L1 $

L2 $
L2 $
L2 $
L2 $

LLC

Mem
Cycle-Accurate HSA Simulator

Connect to gpgpusim
CUDA/OpenCL API library interface
- PTX instruction set emulator
- Timing model
- Power model
Interface to CUDA/OpenCL (OCL) API

- Implement OpenCL / CUDA interface calls in a new DLL.
- Adjust LD_LIBRARY_PATH and CUDA/OpenCL application runs on simulator rather than GPU hardware.
Building full-system memory exploration environment

CUDA/ OpenCL Benchmarks
- libCUDA/ libOpenCL

Linux/Android

Full-system Sim by QEMU
- X86/ARM

Process Info.

Detailed CPU simulator
- (OOO pipeline model)

GPGPUSim
- CudaGPU
- PTX Code, Vars
- Copy Engine
- CudaCores

Performance model (Cache hierarchy)
## Comparison of Simulator Features

<table>
<thead>
<tr>
<th>Name</th>
<th>X86 ISA</th>
<th>ARM ISA</th>
<th>Linux</th>
<th>Android</th>
<th>CPU model</th>
<th>GPU model</th>
<th>CUDA/OpenCL</th>
<th>Shared address space</th>
</tr>
</thead>
<tbody>
<tr>
<td>QEMU</td>
<td>√</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MARSSx86</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multi2Sim</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>OpenCL+(CUDA)</td>
</tr>
<tr>
<td>GPGPU-Sim</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>OpenCL+CUDA</td>
</tr>
<tr>
<td>HSAemu</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td>OpenCL</td>
</tr>
<tr>
<td>gem5-gpu</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>CUDA</td>
</tr>
<tr>
<td><strong>Ours</strong></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>OpenCL+CUDA</td>
</tr>
</tbody>
</table>

Our goals: Full system + Qemu + GPU
Proposed Full-System Simulation Framework for HSA and memory

- Integration of QEMU and GPGPU-Sim
- Simulation framework with
  - Detailed CPU (fast), GPU simulation
  - Shared address space via QEMU Mem
  - Support CUDA and OpenCL
  - Detailed memory systems (from M2S)
  - Full-system supporting
  - Vertical profiling from application and OS kernel
Handle multiple address domains

- Our CUDA/Our OpenCL/NTHU OpenCL Library
- Guest Applications
- Guest OS Kernel
- QEMU
- GPGPUSim

- Help_Func_RW
  - 1-to-1 CUDA Runtime
    - cudaGetDeviceProperties
    - cudaLaunch
    - cudaSetupArgument
    ...

- 1-to-1 OpenCL Runtime
  - clCreateBuffer
  - clGetDeviceInfo
  - clCreateProgramWithSource
  - clEnqueueNDRangeKernel
  ...

- NCTU GPGPU driver
  - program_create
  - kernel_create
  - kernel_set_arg
  - gpgpusim_launch_kernel

- GPGPUSim address

- Guest virtual address
- Guest phy address

- I/O device
- MMIO

- Guest OS Kernel
- Guest Applications
- Our CUDA/Our OpenCL/NTHU OpenCL Library
- QEMU
A Qemu-based on platform
- Full-system simulator for manycore and OS behavior
- Trace mechanism in Qemu
- Include instruction/data trace
- Exchanging info with performance model
- Flexible cycle accurate timing model
The End

Thank you!

Keep going …