LOST IN ABSTRACTION: PITFALLS OF ANALYZING GPUS AT THE INTERMEDIATE LANGUAGE LEVEL

TONY GUTIERREZ, BRADFORD M. BECKMANN, ALEXANDRU DUTU, JOSEPH GROSS, JOHN KALAMATIANOS, ONUR KAYIRAN, MICHAEL LEBEANE, MATTHEW POREMBA, BRANDON POTTER, SOORAJ PUTHOOR, MATTHEW D. SINCLAIR, MARK WYSE, JIEMING YIN, XIANWEI ZHANG, AKSHAY JAIN†, TIMOTHY G. ROGERS†

AMD RESEARCH, †PURDUE UNIVERSITY
EXECUTIVE SUMMARY
HIGH-LEVEL DIFFERENCES: IL VS. MACHINE ISA

- Intermediate Language (IL)
  - ISA for virtual machine
  - Represents data parallel execution well
  - Primarily designed for compiler optimizers

- Lots of details abstracted
  - GPU pipeline is SW managed
  - Machine ISA manages lots state for various HW/SW interfaces
AGENDA

- Executive summary
- Motivation and background
- Pitfalls of analyzing GPUs using IL
- HW runtime correlation and error
- Conclusion
CYCLE-LEVEL SIMULATION IS IMPORTANT

SW ABSTRACTIONS

- Rapid prototyping of research ideas
- Open-source – inexpensive!
  - Functional call convention
  - Special value location
  - System calls
  - And more...

Application binary interface
• Binary format
• Functional call convention
• Special value location
• System calls
• And more…
Old View: GPU is an accelerator for offloading data parallel functions from the CPU.

New View: GPU as primary HPC and datacenter compute device. CPU used for I/O, system services, etc.

We must understand how to properly model the HW/SW interfaces in light of this new view.
HIGH-LEVEL SOFTWARE INTERACTIONS
GPU IS A HW/SW CO-DESIGNED MACHINE

1) Two-phase compilation flow
2) Rich runtime layer
3) Co-designed HW tightly coupled to runtime

As GPU/SW stack becomes more complex, emulation becomes more difficult
Runtime API used to trigger dispatch
Finalizer or JIT generates machine ISA from IL
Instruction schedules are HW dependent

Kernel driver is much easier to maintain: only need ioctl

By coupling simulation infrastructure to OS layer only, we have the flexibility to easily support other programming models.
# GPU SIMULATORS

## OVERVIEW OF SOFTWARE ABSTRACTIONS IN STATE-OF-THE-ART SIMULATORS

<table>
<thead>
<tr>
<th></th>
<th>Runtime Support</th>
<th>ISA</th>
<th>ABI Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPGPU-Sim</td>
<td>Emulated</td>
<td>IL</td>
<td>Simplified by simulator</td>
</tr>
<tr>
<td>Multi2Sim</td>
<td>Emulated</td>
<td>IL/Machine ISA</td>
<td>Simplified by simulator</td>
</tr>
<tr>
<td>gem5</td>
<td>Emulated</td>
<td>IL</td>
<td>Simplified by simulator</td>
</tr>
</tbody>
</table>
GPU SIMULATORS

OVERVIEW OF SOFTWARE ABSTRACTIONS IN STATE-OF-THE ART SIMULATORS

<table>
<thead>
<tr>
<th></th>
<th>Runtime Support</th>
<th>ISA</th>
<th>ABI Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPGPU-Sim</td>
<td>Emulated</td>
<td>IL</td>
<td>Simplified by simulator</td>
</tr>
<tr>
<td>Multi2Sim</td>
<td>Emulated</td>
<td>IL/Machine ISA</td>
<td>Simplified by simulator</td>
</tr>
<tr>
<td>gem5</td>
<td>Off-the-shelf</td>
<td>Machine ISA</td>
<td>Models real ABI</td>
</tr>
</tbody>
</table>

- This work adds
  - GCN3 support – AMD’s GPU machine ISA
  - HSA ABI
  - Support for off-the-shelf ROCm stack (user space)
  - Emulated ROCk

- We evaluate the effects on simulation for both HSAIL and GCN3
- We demonstrate that HSAIL introduces significant additional error

Radeon Open Compute Platform (ROCc)
- The open-source implementation of HSA principles for AMD Devices
  - ROCr – runtime
  - ROCt – thunk (user driver)
  - ROCk – kernel driver
  - HCC – heterogenous compute compiler
AGENDA

- Executive summary
- Motivation and background
- Pitfalls of analyzing GPUs using IL
  - Methodology
  - Quantitative analysis
    - Instruction scheduling
    - Kernel argument access
    - Control flow
    - Instruction expansion
- HW runtime correlation and error
- Conclusion
gem5’s GPU model
- With GCN3 support added
- Support for HSA standard and off-the-shelf ROCm

ROCM version 1.1
- HCC-hsail clang compiler version 3.5
  - Same binary used on hardware/gem5
  - For HSAIL extract the kernel code from binary before finalizing to GCN3

HW runs on AMD Pro A12-8800B APU
- Radeon open compute profiler (RCP) used to capture hardware data

<table>
<thead>
<tr>
<th>Workload</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array BW</td>
<td>Memory streaming</td>
</tr>
<tr>
<td>Bitonic Sort</td>
<td>Parallel merge sort</td>
</tr>
<tr>
<td>CoMD</td>
<td>DOE Molecular-dynamics algorithms</td>
</tr>
<tr>
<td>FFT</td>
<td>Digital signal processing</td>
</tr>
<tr>
<td>HPGMG</td>
<td>Ranks HPC systems</td>
</tr>
<tr>
<td>LULESH</td>
<td>Hydrodynamic simulation</td>
</tr>
<tr>
<td>MD</td>
<td>Generic molecular-dynamics algorithms</td>
</tr>
<tr>
<td>SNAP</td>
<td>Discrete ordinates neutral particle transport application</td>
</tr>
<tr>
<td>SpMV</td>
<td>Sparse matrix-vector multiplication</td>
</tr>
<tr>
<td>XSBench</td>
<td>Monte Carlo particle transport simulation</td>
</tr>
</tbody>
</table>
**KNOWLEDGE OF UNDERLYING HW RESOURCES**

**GCN3 VIEW OF COMPUTE UNIT PIPELINE**

Not utilized by HSAIL instructions

- s_load s[0]
- s_waitcnt(0)
- v_add v6, s0, v0

Load count = 0
ld $v0, [%__arg_p1]
add $v2, $v0, $v1
Much better instruction scheduling from GCN3 compiler

Higher reuse distance
- Less probability of accesses same banks/registers

Better register allocation
**KERNEL ARGUMENT ACCESS**

**MEMORY SEGMENTS**

- **HSAIL**
  - `# load kernarg 0 ptr
   ld_kernarg $d0, [%__arg_p1]`
  - `# load kernarg 0
   ld_global $d1, [$d0]`
  - HSAIL ld specifies `segment + arg num offset` only

- **GCN3**
  - `# s[6:7] = Kernarg ptr
   s_load_dword s[0:1], s[6:7], 0x08`
  - GCN3 `s_load_dword` uses `real address + byte offset`
  - ABI specifies Kernarg pointer stored in s[6:7]
VRF VALUE UNIQUENESS

SCALAR UNIT DOES NOT IMPROVE VALUE UNIQUENESS

Many VRF R/W are redundant
- Typically GCN3 codes experience more value uniqueness
- ABI abstraction in HSAIL hides some value redundancy
  - Base address storage

Uniqueness definition: ratio of unique lane values to active lanes.

EXEC = 1110  %Unique = 66%
CONTROL FLOW DIVERGENCE
SIMT VS. VECTOR EXECUTION MODEL

HSAIL

Source code:

```c
if (i > 31) {
    *x = 84;
} else if (i < 16) {
    *x = 90;
}
```

Execute taken path first & flush IB

Branch over BB2 & BB3, flush IB

Fall through to BB3

Reconvergence point reached, HW initiated jump to divergent path

Instruction buffer

Flushed!
CONTROL FLOW DIVERGENCE
SIMT VS. VECTOR EXECUTION MODEL

HSAIL

Source code:
if (i > 31) {
    *x = 84;
} else if (i < 16) {
    *x = 90;
}

Branches are optimizations for case when EXEC = 0 for a BB
GCN3 relies on predication more frequently
- Requires fewer hardware “jumps”
- Requires fewer IB flushes
HSAIL instructions are semantically powerful
- Single HSAIL inst => several GCN3 insts

Declarative: what operation to perform

Imperative: how to perform operation (Newton-Raphson Method)
GCN3 executes far more dynamic instructions
- Code expansion
- Extra instructions due to ABI
- Dependency handling instructions
- Intermixed scalar instructions
- Varies across applications
AGENDA

- Executive summary
- Motivation and background
- Pitfalls of analyzing GPUs using IL
- HW runtime correlation and error
- Conclusion
HSAIL adds significant, and unpredictable error
- Inherent to using HSAIL and emulated runtime
- With only publicly available information, GCN3 still improves error by > 30%

Results correlate well
- May indicate preservation of performance trends
- Microarchitectural events, and absolute performance still left with significant error
GPU Compute workloads are becoming more complex
- Utilize many components of the system simultaneously
- Lots of complex HW/SW interactions

Modeling the full stack correctly is important
- Challenging, as HW changes frequently
- Abstracting at OS only provides nice balance

Machine ISA instructions accurately capture application behavior
- Microarchitecture characteristics skewed by IL
- Machine ISA captures real HW events/state

GPU simulators must capture full-system behavior and machine ISA/microarchitecture interaction
**INTERESTED IN LEARNING MORE?**

**MODEL ENHANCEMENTS AND PUBLIC RELEASE**

- Public release of GCN3 ISA and ROCm support coming soon
- ISCA 2018 tutorial
  - Will cover:
    - Model updates
    - ROCm simulation in detail
    - Toolchain and benchmarks
      - *HSAIL has been deprecated
      - Toolchain uses LLVM IL and compilers directly produces ISA binary
DISCLAIMER & ATTRIBUTION

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions and typographical errors.

The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION.

AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

ATTRIBUTION

© 2018 Advanced Micro Devices, Inc. All rights reserved. AMD, the AMD Arrow logo, and combinations thereof are trademarks of Advanced Micro Devices, Inc. OpenCL is a trademark of Apple Inc. used by permission by Khronos. Linux is a registered trademark of Linus Torvalds. Other names are for informational purposes only and may be trademarks of their respective owners.