MIAOW: An Open Source GPGPU

www.miaowgpu.org

Vinay Gangadhar, Raghu Balasubramanian, Mario Drumond, Ziliang Guo, Jai Menon, Cherin Joseph, Robin Prakash, Sharath Prasad, Pradip Vallathol, Karu Sankaralingam

Vertical Research Group
University of Wisconsin - Madison
Executive Summary

• MIAOW is a credible GPGPU implementation
  – Compatible with AMD Southern Islands ISA
  – Runs OpenCL programs and prototyped on FPGA
  – Similar design to industry state-of-art
  – Similar performance to industry state-of-art*
  – Flexible and Extendable

• MIOAW’s hardware design is Open Source

• Contributes to changing hardware landscape

* Frequency, Physical Design, Area goals relaxed
Applications that drive computing are changing
Need innovative new hardware
Open Source Hardware is gaining momentum
Some Open Source Hardware Microprocessors
ZERO Open Source GPUs
Lessons from Open Source S/W

- PHP, Linux, ruby, mysql, sqlite, apache, gcc _late 80s, early 90s_ $0
- Facebook, Twitter, Whatsapp, Instagram _Web 2.0_ >$10 bill.
- MIAOW, OpenCores, RISC-V etc.. $0
- ? $0
MIAOW Technical Overview

Demonstrate MIAOW is credible GPGPU

Implications and Possibility of Open Source Hardware
## ISA Summary

<table>
<thead>
<tr>
<th>Type</th>
<th>Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Vector</strong></td>
<td>add, addc, sub, mad, madmk, mac, mul, max, max3, min, subrev</td>
</tr>
<tr>
<td></td>
<td>and, or, xor, not, mov, lshrev, lshrev, ashrev, ashrrev, bfe, bfi, cndmask</td>
</tr>
<tr>
<td></td>
<td>cmp.{ lt, eq, le, gt, lg, ge, ne, ngt, neq }</td>
</tr>
<tr>
<td><strong>Scalar</strong></td>
<td>add, addk, sub, max, min, mul, mulk</td>
</tr>
<tr>
<td></td>
<td>and, andn2, or, xor, not, mov, movk, lshl, lshr, ashr, saveexec</td>
</tr>
<tr>
<td></td>
<td>cmp.{ eq, lt, gt, ge, lt, le, eq, lg, gt, ge, lt, le }</td>
</tr>
<tr>
<td></td>
<td>barrier, branch, cbranch, endpgm, waitcnt</td>
</tr>
<tr>
<td><strong>Memory</strong></td>
<td>load, buffer_load</td>
</tr>
<tr>
<td><strong>Vector_Mem:</strong></td>
<td>tbuffer_load, tbuffer_store</td>
</tr>
<tr>
<td><strong>Scalar_Mem:</strong></td>
<td>ds_read, ds_write</td>
</tr>
</tbody>
</table>

- 95 instructions
- Single-precision support only
- No graphics support (yet)
MIAOW has **32** Compute Units (CUs)
Hardware Organization

- Single Issue
- 40 Wavefronts
- 16-wide vector ALUs
- LSU – Memory operations
MIAOW Implementations

(a) Full ASIC Design
Ultra-threaded Dispatcher
CU
CU
CU
CU
CU
CU
CU
CU
L2 Cache
M0 M1 M2

Area, Power from Floorplan and simulation

(b) Mapped to FPGA
Ultra-threaded Dispatcher
CU
CU
CU
CU
CU
CU
CU
CU
L2 Cache
M0 M1 M2

Long running apps S/W development Prototyping

(c) Hybrid Design
Virtex7 FPGA
1 CU @ 50 MHz
133K LUTs, 100K Registers

compute_unit_fpga_i

simf0
163050
issue0
45376
alu
53702
103050
valu
75912
scoreboard

Legend
CU : Compute Unit; M0, M1, M2 : Memory Controllers
RTL Hard Macro BRAM FPGA IP C/C++ Models
Design Team

- Small initial design team (12 mo)
  - 5-person HDL team
  - 1-person software team
  - 1-person physical design team
- Added FPGA expert
- 3 undergrads extended the design
- Total duration: 36 months
- Area, Frequency, Performance, Power – NON GOALs
Software Compatibility

- Runs unmodified OpenCL programs
- All OpenCL benchmarks
- Many Rodinia benchmarks
- Easily extendable to add additional instructions
MIAOW Technical Overview

Demonstrate MIAOW is a credible GPGPU

Implications and Possibility of Open Source Hardware
MIAOW vs. AMD Tahiti

AMD’s latest GPU CU architectures build upon Tahiti. Tahiti released in 2011.
Performance Comparison

MIAOW | AMD Tahiti | Tahiti Speedup over AMD A10-7850K CPU
--- | --- | ---
5X | 107X | 121X
46X | 30X | 40X

Relative Performance (Cycles)

- BinarySearch
- BitonicSort
- MatrixTranspose
- PrefixSum
- Reduction
- ScanLargeArrays
Area Comparison

Tahiti CU area*: 5.02 mm² @ 28nm
MIAOW CU area: 9.1 mm² @ 32nm

*Estimate from die-photo analysis and block diagrams from wccftech.com
Area Comparison

Tahiti CU area*: 5.02 mm² @ 28nm
MIAOW CU area: 9.1 mm² @ 32nm

*Estimate from die-photo analysis and block diagrams from wccftech.com
Power

Tahiti CU power*: 0.52 W
MIAOW CU Power: 1.1 W

* Ballpark estimate from TDP and occupancy
Power

Tahiti CU power*: 0.52 W
MIAOW CU Power: 1.1 W

* Ballpark estimate from TDP and occupancy
MIAOW is comparable to industry designs
MIAOW Technical Overview

Demonstrate MIAOW is a credible GPGPU

Implications and Possibility of Open Source Hardware
Lessons Learned

• It was surprising this was doable!
  – Microarchitecture design, HDL implementation, verification was not tedious
• Software toolchain being available was great
• We punted on physical design
• FPGA tools are still quite tedious to use
Implications for Industry

• Open Source Hardware GPU
  – Relevance to OpenCompute & Maker movement

• How can a HW startup benefit from MIAOW
  – Start with MIAOW and focus on innovative pieces from day one

• IP and Compiler
  – License under BSD, ISA is OK, compiler usable
  – How to avoid IP infringement?
Lessons from Open Source S/W

PHP, Linux, ruby, mysql, sqlite, apache, gcc
*late 80s, early 90s*

Facebook, Twitter, Whatsapp, Instagram
Web 2.0

$0

>$10 bill.

MIAOW, OpenCores, RISC-V etc..

$0
What drives Open Source Software?

Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects, Lakhani K and Wolf R. In Perspectives on Free and Open Source Software

• It’s fun!
  – “Enjoyment-based intrinsic motivation, namely how creative a person feels when working on the project, is the strongest and most pervasive driver.”

• It’s valuable
  – “user need, intellectual stimulation derived from writing code, and improving programming skills are top motivators”
Conclusion

• MIAOW is transformative for GPU research

• Its role in open source hardware movement?

• Are open source hardware chips feasible?

• More community support → First Open Source Silicon GPU Chip
www.miaowgpu.org

Journal article *(ACM TACO 2015)*: Enabling GPGPU Low-Level Hardware Explorations with MIAOW: An Open-Source RTL Implementation of a GPGPU

3.9 FPS on FPGA @ 50 MHz

23 FPS in simulation @ 222 MHz
Back Up Slides
Fetch & Wavepool

Fetch Controller → Round Robin Scheduler → WavePool Controller → Scoreboard Feeder

Instruction Cache

PC0, PC1, ..., PC39

WF_ID Gen

WF tag, Done

Instr Valid

Base0, Base1, ..., Base39

LDS, VGPR, SGR Base

exec_mask

mask_gen
Vector ALU/FPU

Vector ALU/Vector FPU
Issue: Wavefront Scheduling and Arbiter
Load/Store Unit

- Memory
- Stage Pipeline
- FSM
- Arithmetic

**LSU Controller**
- buf_wr_en
- opcode
- exec_mask
- WF_ID, PC
- destination
- exec_mask

**Address Calculation**

**LD/ST Buffer**

**Data Memory**

**To Write Back**

**Issued Instr Table**

**Load Store Queue**
HOTCHIPS 2015

Many technical details in this publication: TACO 2015: Enabling GPGPU Low-Level Hardware Explorations with MIAOW: An Open-Source RTL Implementation of a GPGPU

www.miaowgpu.org
Virtex7-based FPGA – Neko

- 1 CU design
- 50 MHz
- Utilization:
  - LUTs 133K
  - Registers: 100K
# Flexibility

<table>
<thead>
<tr>
<th>Design choice</th>
<th>Realistic</th>
<th>Flexibility</th>
<th>Area/Power impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fetch bandwidth (1)</td>
<td>Balanced†</td>
<td>Easy to change</td>
<td>Low</td>
</tr>
<tr>
<td>Wavepool slots (6)</td>
<td>Balanced†</td>
<td>Parametrized</td>
<td>Low</td>
</tr>
<tr>
<td>Issue bandwidth (1)</td>
<td>Balanced†</td>
<td>Hard to change</td>
<td>Medium</td>
</tr>
<tr>
<td># int FU (4)</td>
<td>Realistic</td>
<td>Easy to change</td>
<td>High</td>
</tr>
<tr>
<td># FP FU (4)</td>
<td>Realistic</td>
<td>Easy to change</td>
<td>High</td>
</tr>
<tr>
<td>Writeback queue (1)</td>
<td>Simplified</td>
<td>Parametrized</td>
<td>Low</td>
</tr>
<tr>
<td>RF ports (5,4)</td>
<td>Simplified</td>
<td>Hard to change</td>
<td>High</td>
</tr>
<tr>
<td>RF ports (SRAM) (1)</td>
<td>Realistic</td>
<td>Hard to change</td>
<td>Low</td>
</tr>
<tr>
<td>Types of FU</td>
<td>Simplified</td>
<td>Easy to change</td>
<td>High</td>
</tr>
</tbody>
</table>

† Fetch optimized for cache-hit, rest sized for balanced machine.
Numbers in parenthesis indicate the design parameters.
Verification

The diagram illustrates the process of verifying GPU systems. It starts with Random Unit Tests / Unit Tests / Benchmarks leading to OpenCL Compiler and AMD GPU Drivers. These components generate a GPU Kernel Binary, which is then fed into the Emulator. The Emulator, in turn, interacts with components such as Thread Dispatcher, GPU Memory, and CU (0) to CU (N), which include Instr Buffer, LDS, and Trace Mon. The output of this process includes Pass / Fail indicators.
## As a Research Tool

<table>
<thead>
<tr>
<th>Direction</th>
<th>Research Idea</th>
<th>MIAOW enabled findings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traditional µarch</td>
<td>Thread-block compaction (TBC)</td>
<td>• Implemented TBC in RTL&lt;br&gt;• Significant design complexity&lt;br&gt;• Increase in Critical Path length</td>
</tr>
<tr>
<td>New Directions</td>
<td>Circuit-Failure Prediction (Aged SDMR)</td>
<td>• Implemented entirely in µarch&lt;br&gt;• Works elegantly in GPUs&lt;br&gt;• Small area, power overheads</td>
</tr>
<tr>
<td></td>
<td>Timing Speculation (TS)</td>
<td>• Quantifies error-rate on GPU&lt;br&gt;• TS framework for future studies</td>
</tr>
<tr>
<td>Validation of Simulator studies</td>
<td>Transient Fault Injection</td>
<td>• RTL Level Fault Injection&lt;br&gt;• More Gray area than CPUs&lt;br&gt;• Silent data corruption seen</td>
</tr>
</tbody>
</table>
Virtex7-based FPGA – Neko

Virtex7 FPGA
1 CU @ 50 MHz
133K LUTs, 100K Registers