MIAOW - An Open Source RTL Implementation of a GPGPU*

Raghuraman Balasubramanian Vinay Gangadhar Ziliang Guo Chen-Han Ho Cherin Joseph
Jaikrishnan Menon Mario Paulo Drumond Robin Paul Sharath Prasad Pradip Valathol
Karthikeyan Sankaralingam
University of Wisconsin-Madison

Abstract: *Graphic Processing Unit (GPU) based general purpose computing is developing as a viable alternative to CPU based computing in many domains. In this paper, we introduce MIAOW (Many-core Integrated Accelerator Of Wisconsin), an open source RTL implementation of the AMD Southern Islands GPGPU ISA, capable of running unmodified OpenCL-based applications. We present our design motivated by our goals to create a realistic, flexible, OpenCL compatible GPGPU, capable of emulating a full system. We demonstrate that MIAOW enables disruptive and transformative research and has the potential to bring all of the benefits of open source development to GPUs in real products in the long term.*

The trend for the last several years in computer architecture has been the effort to extract more performance from the available silicon. One such approach has been to exploit the massive parallelism of graphic cards and use their execution units for general purpose computation instead of simply graphics operations. To further extend this work and unleash the benefits of open source development of GPU hardware, we have developed and released an open source GPU called MIAOW which is implemented in RTL and prototyped on an FPGA. This paper reports on the design, development, characterization, and research utility of MIAOW (*acronymized as **M**any-core **I**ntegrated **A**ccelerator **O**f **W**isconsin*). MIAOW's RTL source code, simulation infrastructure, and benchmarks are available for download on github – `https://github.com/VerticalResearchGroup/miaow`. The source code release also includes White Paper with many details. MIAOW is currently primarily a research vehicle, but we envision that with community development it becoming a main-stream commercial GPU available in silicon, competitive with commercial designs.

Goals The primary driving goals for MIAOW are: i) *Realism:* it should be a *realistic* implementation of a GPU resembling principles and implementation tradeoffs in industry GPUs; ii) *Flexible:* it should be flexible to accommodate research studies of various types, the exploration of forward-looking ideas, and form an end-to-end open source tool; iii) *Software-compatible:* It should use standard and widely available software stacks like OpenCL or CUDA compilers to enable executing various applications and not be tied to in-house compiler technologies and languages. iv) *Open source:* The RTL source is released open source along with verification tool chain, synthesis scripts etc.

Design Approach Driven by these goals, we have developed MIAOW as an implementation of a subset of AMD's Southern Islands(SI) ISA [1]. While we pick one ISA and design style, we feel it is representative of a GPGPU design [4] — AMD and NVIDIA's approaches have some commonalities [5]. This delivers on all three primary goals. It is a real ISA (machine's internal ISA compared to PTX or AMD-IL which are external ISAs found in products launched in 2012), is a clean-slate design so likely to remain relevant for a few years, and has a complete ecosystem of OpenCL compilers and applications. In concrete terms, MIAOW focuses on microarchitecture of the Compute Unit (CU) and implements them in synthesizable Verilog RTL, and leaves the memory hierarchy and memory controllers as behavioral (emulated) models. MIAOW takes a hybrid strategy with some components, namely L1 cache, OCN, and memory controller implemented as behavioral C/C++ modules. This strikes a good balance between realism, flexibility and a framework that can be released. A modified design called Neko has been synthesized on an FPGA.

Hardware architecture MIAOW implements a subset of the Southern Islands ISA which we summarize below. The *architectural state and registers* defined by MIAOW's ISA includes the program counter, execution mask, status registers, mode register, general purpose registers (scalar s0-s103 and vector v0-v255),

*Some authors have current affiliations at Google, Qualcomm, NVIDIA, and EPFL. Work done at UW-Madison.

2015 IEEE COOL Chips XVIII

Local Data Share (LDS), 32-bit memory descriptor, scalar condition codes and vector condition codes. *Program control* is defined using predication and branch instructions. The *instruction encoding* is of variable length having both 32-bit and 64-bit instructions. Scalar instructions are organized in 5 formats [SOPC, SOPK, SOP1, SOP2, SOPP]. Vector instructions come in 4 formats of which three [VOP1, VOP2, VOPC] use 32-bit instructions and one [VOP3] uses 64-bit instructions to address 3 operands. Scalar memory reads (SMRD) are 32-bit instructions involved only in memory read operations and use 2 formats [LOAD, BUFFER_LOAD]. Vector memory instructions use 2 formats [MUBUF, MTBUF], both being 64-bits wide. Data share operations are involved in reading and writing to LDS and global data share (GDS). Four commonly used instruction encodings are shown in Table 1. Two memory *addressing modes* are supported - base+offset and base+register.

Of a total of over 400 instructions in SI, MIAOW's instruction set is a carefully chosen subset of 95 instructions and the generic instruction set is summarized in Table 1. This subset was chosen based on benchmark profiling, the type of operations in the data path that could be practically implemented in RTL by a small design team, and elimination of graphics-related instructions. MIAOW does not support 64 bit integer and floating operations as of now, and this is one of the limitation of MIAOW. But the AMD APP kernels [2] do not use 64 bit operations aggressively and all of them could be run on MIAOW, thus making MIAOW 32-bit software compatible to OpenCL applications. In short, the ISA defines a processor which is a tightly integrated hybrid of an in-order core and a vector core all fed by a single instruction supply and memory supply with massive multi-threading capability. The complete SI ISA judiciously merges decades of research and advancements within each of those designs.

The MIAOW GPGPU adheres to the canonical design of a GPU and consists of a simple dispatcher, a configurable number of compute units, memory controller, OCN, and a cached memory hierarchy. MIAOW allows scheduling up to 40 wavefronts on each CU, which may belong to different work-groups. Figure 1 shows the high-level organization of MIAOW and corresponding area and power breakdown of the main sub-modules. Figure 2 shows the pipeline organization. Overall the trends for MIAOW are similar to industry products as analyzed in more detail elsewhere [3].

Results and Impact We have shown what it takes to build a GPU and shows a small academic team can build a modern GPU. MIAOW provides transformative capability in advancing GPU research and ultimately into an open source product. MIAOW allows physical design exploration of "traditional" research topics, new types of research explorations, and validation/calibration of simulators. Overall MIAOW is a unique and potentially disruptive architecture for GPUs.

References

[1] Reference guide: Southern islands series instruction set architecture, `http://developer.amd.com/wordpress/media/2012/10/AMD_Southern_Islands_Instruction_Set_Architecture.pdf`, 2012.

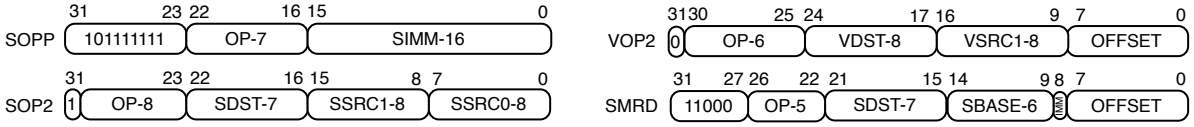
[2] Amd app 3.0 sdk, kernels and documentation, 2013.

[3] R. Balasubramanian, V. Gangadhar, Z. Guo, C.-H. Ho, C. Joseph, J. Menon, M. P. Drumond, R. Paul, S. Prasad, P. Valathol, and K. Sankaralingam. Enabling gpgpu low-level hardware explorations with miaow - an open source rtl implementation of a gpgpu. (to appear). *IEEE Transcations on Architecture and Code Optimization*, 2015.

[4] M. Fried. Gpgpu architecture comparison of ati and nvidia gpus, 2012.

[5] Y. Zhang, L. Peng, B. Li, J.-K. Peir, and J. Chen. Architecture comparisons between nvidia and ati gpus: Computation parallelism and data communications. In *IISWC '11*, 2011.

Type	Instructions
Vector	ALU: {U32, I32, F32} - add, addc, sub, mad, madmk, mac, mul, max, max3, min, subrev Bitwise: {B32} - and, or, xor, not, mov, lshrrev, lshlrev, ashlrev, ashrrev, bfe, bfi, cndmask Compare: {U32, I32, F32} - cmp_{ lt, eq, le, gt, lg, ge, ne, ngt, neq }
Scalar	ALU: {U32, I32} - add, addk, sub, max, min, mul, mulk Bitwise: {B32} - and, andn2, or, xor, not, mov, movk, lshl, lshr, ashr, saveexec Compare: {U32, I32} - cmp_{ eq, lt, gt, ge, lt, le, eq, lg, gt, ge, lt, le } Conditional: - barrier, branch, cbranch, endpgm, waitcnt
Memory	Scalar_Mem – SMRD DWORD Format: {x, x2, x4} - load, buffer_load Vector_Mem – Buffer Format: {x, xyzw} - tbuffer_load, tbuffer_store Date Share (LDS, GDS) : {B32} - ds_read, ds_write



OP	Operands for instruction. Each format has its own operands.
VDST	Vector destination register, can address only vector registers.
SRC0	Source 2, can address vector, scalar and special registers, also can indicate constants.
SSRC1	Scalar source 1, can address only scalar registers.
IMM	Flag that marks whether OFFSET is an immediate value or the address of a scalar register
SIMM	16 bit immediate value
SDST	Scalar destination register, can address only scalar registers.
VSRC1	Vector source 1, can address only vector registers.
SBASE	Scalar register that contains the size and base address.
OFFSET	Offset to the base address specified in SBASE

Table 1: Supported ISA

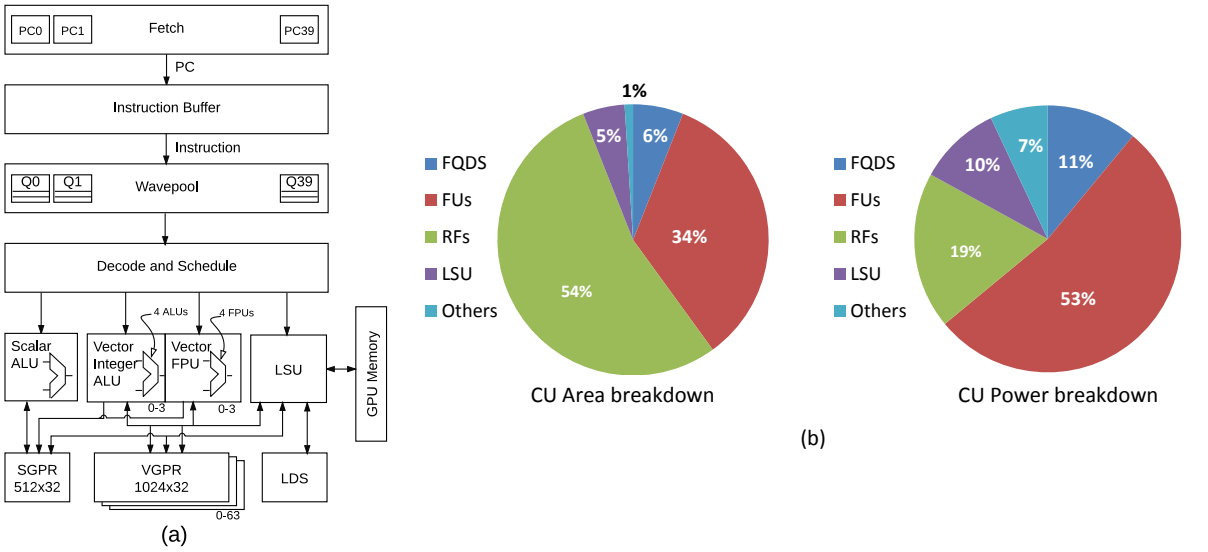


Figure 1: a) Compute Unit (CU) microarchitecture; b) Compute Unit (CU) Area and Power breakdown
[FQDS: Fetch/Queue/Decode/Schedule; FUs: Functional Units; RFs: Register Files; LSU: Load Store Unit]
(Area metric – μm^2 and Power metric – mW)

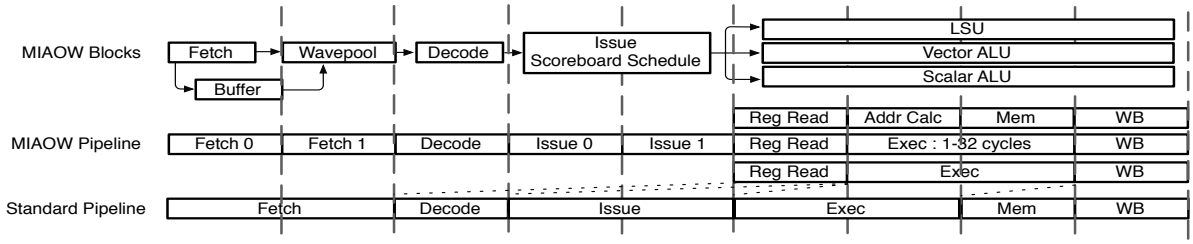


Figure 2: MIAOW Compute Unit Pipeline stages

2015 IEEE COOL Chips XVIII 978-1-4673-7325-8/15/$31.00 ©2015 IEEE