Ventus: an RVV-based General Purpose GPU Design and Implementation

Kexiang Yang[1,2], Hualin Wu[3], Jingzhou Li[1,2], Chufeng Jin[1,2], Yujie Shi[1,2], Xudong Liu[1,2], Zexia Yang[1,2], Fangfei Yu[1,2], Mingyuan Ma[1,2], Sipeng Hu[4], Tianwei Gong[4], Hu He[1,2]*

[1]Tsinghua University
[2]International Innovation Center of Tsinghua University, Shanghai
[3]Terapines Ltd
[4]Beijing Information Science and Technology University

Abstract

Graphics Processing Units (GPUs) have become the most popular platform for accelerating modern applications such as Machine Learning, Signal Processing, and Graph workloads. Modern GPUs use Single Instruction, Multiple Thread (SIMT) structures to schedule several Single Instruction, Multiple Data (SIMD) pipelines, thus maximizing Data Level Parallelism. In this work, we propose Ventus, a General Purpose GPU (GPGPU) implementation based on the RISC-V Vector Extension (RVV). Lanes in the warps of Ventus are organized as a vector-thread architecture. We add self-defined instructions, such as branch, register index extension and Tensor Core related instructions to fulfill the functional requirements of a GPGPU. We accomplish a complete OpenCL to RVV compiler and driver that fit our hardware design. Ventus is successfully deployed on an FPGA-based platform and scales up to 16 SMs with a total of 256 warps. This work is developed in Chisel HDL and is now open-sourced on Github.

Introduction

GPGPU is a field of computing that utilizes the processing power of GPUs for parallelism and high-throughput computing. GPGPU has become increasingly popular in recent years, especially in fields such as machine learning, scientific computing, and computer vision. Open-source GPGPU helps researchers dive into the inner implementation of the hardware and benefit from software-hardware co-design. There are several open-source GPGPU implementations, MIAOW, Nyuzi and Vortex [1].

RVV is a set of optional instructions defining a series of vector element operations and vector registers, supporting different vector widths and lengths at different situation.

We propose our open-source GPGPU, Ventus. In this work, we extend the meaning of RVV to serve as the ISA for GPGPU. Taking into account the programming model and practical requirements, we supplement ISA with our custom extensions. We use Chisel HDL to design an RTL, develop a driver based on PoCL, and create a compiler from OpenCL to RVV based on LLVM. Finally, we evaluate our work on FPGA.

Motivation

Both CUDA and OpenCL provide similar programming models for GPGPU with SIMT architecture, where threads are organized into groups, called thread blocks, or Correlative Thread Array (CTA). Programmers describe the behavior of individual threads and organize them into thread blocks at an expected size, which are then scheduled on the hardware. In NVIDIA's GPU, such hardware units are call Streaming Multiprocessors (SMs) which work as multithreaded SIMD processors. Threads at the same thread block are grouped into warps and scheduled for execution on SMs, switching between warps to hide fetch and execution latency [2].

Vector processors extend the capability of multi-data processing with SIMD lanes, which is similar to SMs in GPUs. RVV and GPGPU also share similarities in terms of compilers. Compilers only focus on the behavior of a single vector element, and process the corresponding data based on the index of the vector element itself. The hardware then packs and schedules the operations on the warps.

Considering the typical data usage requirements of GPGPU and the high configurability that RVV exhibits for vectors of different lengths and element widths, we have chosen the Zve32f version of RVV as our implementation. Combined with RV32IMA_zfinx, RVV programs are executed in the form of a hardware vector-thread architecture. Our hardware can be considered as an RVV processor with elen=32 and vlen=32, and allows multiple RVV programs to be executed in a time-sharing manner, thus supporting SIMT programming model.

Using only the instructions in RVV is not sufficient to cover GPU functionality. Firstly, we add branch control and thread synchronization instructions to control the behavior of threads as [1] does. Secondly, considering that RVV limits the number of registers, we import register index extension instructions, extending the number of architectural registers to 256, which is compatible with current RISC-V programs. Thirdly, we provide immediate-offset memory access instructions to extend RVV's addressing modes. Finally, we add support for tensor convolutions and exponent function required by Transformer.

Ventus GPGPU design

Hardware design

Our hardware architecture is shown in Figure 1. As for task allocation, the driver sends tasks to the CTA-scheduler in thread block units, which are then divided into warps and sent to the SM for execution. As for storage, L1Cache and shared memory are private to the SM, and all SMs are connected to a common L2Cache with an NoC.

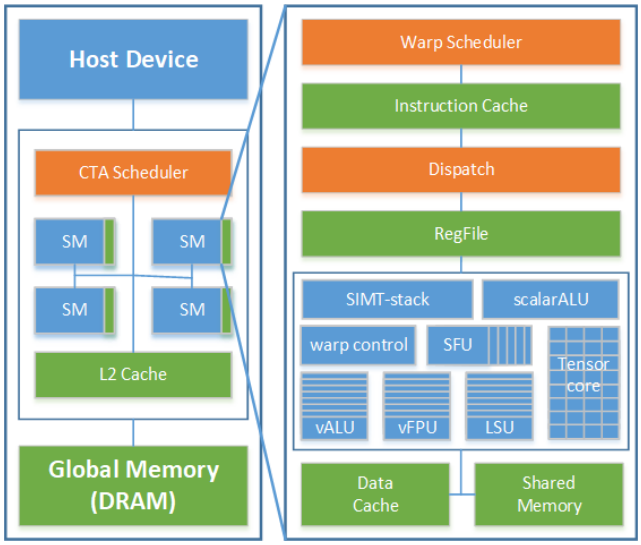

Figure 1: Ventus GPGPU Architecture

Regarding the internal design of the SM, it's described as a SIMD processing pipeline that supports multithreaded scheduling. The front-end includes instruction fetching, decoding, I-buffer, banked vector and scalar register files, scoreboard, and warp scheduler. The back-end includes scalar data-path and vector data-path. Since we use vector instructions to distinguish vector operations, functions such as common data and address calculations, as well as overall branch jumps, can be completed using scalar instructions.

To distinguish thread blocks and thread IDs, we allocate private CSRs for each warp. By utilizing vid.v instruction in RVV, combined with the warp ID information provided by CSR, we can obtain the thread ID of each thread.

Software stack

Ventus is designed to support full profile of OpenCL 2.0. Users write OpenCL kernels which are executed on each SIMT with a portion of workload explicitly set via OpenCL API such as clEnqueueNDRangeKernel. The OpenCL kernel is written in a way that is agnostic to the total workload size, as users can dynamically set the total work size, work dimension, and group size at runtime via the OpenCL API. The OpenCL kernel is explicitly compiled and built by the OpenCL API, using functions such as clCompileProgram, clLinkProgram, or clBuildProgram.

We store the compiled and linked kernel and data in the Executable and Linkable Format (ELF), using a customized linker script to map sections such as .text, .rodata, .data, and .bss into the 4G physical addressing space. As there are scalar ALUs and vector ALUs in each SM, the compiler generates scalar and vector instructions based on analyzing whether the execution path in the kernel is diverged, non-diverged code path will be translated into RV32IMA_zfinx instructions which will be executed on scalar ALU, while diverged code path will be translated into RVV instructions which will be executed on vALU, vFPU etc.

In addition to the ELF loadable sections mentioned above, we have designed different stacks for scalar path and vector path. Local memory is mapped to SRAM for fast data sharing within SM, while the result of scalar ALU is share by the entire SM, we allocate stack for scalar ALU on local memory as well, the RISC-V SP register is used to store the stack base pointer and stack grows upwards. For each SIMT thread, stack spaces are allocated in the private memory of each thread, RISC-V TP register stores the base address of the stack, Ventus hardware will add per-thread offset to TP register translation private memory into physical memory addressing space.

We developed OpenCL driver based on PoCL, which bridges the OpenCL program with the hardware. We also ported open-source ISA simulator spike as Ventus GPGPU ISS. The OpenCL compiler and libraries for Ventus were developed based on LLVM compiler infrastructure.

Evaluation

We use Chisel HDL to develop hardware codes. A 16SM-16warp-16lane version with Tensor Core occupies 65% of the area of 4 VU19P FPGAs. Due to the inter-chip communication latency being limited to 10MHz on FPGAs, the theoretical peak performance is 5GFlops FP32 on normal lanes and 5GFlops FP32 on Tensor Core. After adding FPGA driver and AXI bus support, the design has been deployed on a Xilinx VCU128 board with 2SM-4warp-8lane, achieving a clock frequency of 100MHz and a peak performance of 3.2GFlops with 11% of the area.

OpenCL benchmarks are currently being tested and are expected to be completed soon.

Conclusion

Our proposed Ventus based on RVV includes custom instructions and implements a processor with multithreaded scheduling capabilities, as well as multi-level task allocation and memory access components, fulfilling the requirement of a GPGPU. We have also developed an OpenCL driver and compiler. All hardware designs are developed using Chisel HDL, and we test our designs on FPGA.

All software and hardware codes are open-sourced at https://github.com/THU-DSP-LAB/ventus-gpgpu.

References

[1] Blaise Tine, et al. Vortex: Extending the RISC-V ISA for GPGPU and 3D-Graphics. In MICRO-54, 2021.

[2] Tor M. Aamodt, et al. General-purpose Graphics Processor Architectures. Morgan & Claypool Publishers, 2018.