Preliminary integration of vortex within the OpenPiton platform

Davy Million, César Fuguet Tortolero, Adrian Evans, Jonathan Balkind, Frédéric Petrot

Preliminary Integration of Vortex within the OpenPiton Platform

Davy Million
davy.million@cea.fr
Univ. Grenoble Alpes, CEA, List
F-38000 Grenoble, France

César Fuguet
cesar.fuguettortolero@cea.fr
Univ. Grenoble Alpes, CEA, List
F-38000 Grenoble, France

Adrian Evans
adrian.evans@cea.fr
Univ. Grenoble Alpes, CEA, List
F-38000 Grenoble, France

Jonathan Balkind
jbalkind@ucsb.edu
University of California, Santa Barbara
Santa Barbara, USA

Frédéric Petrot
frederic.petrot@univ-grenoble-alpes.fr
Univ. Grenoble Alpes, TIMA
F-38000 Grenoble, France

CCS Concepts

• **Computer systems organization → Processors and memory architectures**.

Keywords

Cache Memory, GPU, NoC, RISC-V, Open-Source Hardware

ACM Reference Format:
Davy Million, César Fuguet, Adrian Evans, Jonathan Balkind, and Frédéric Petrot. 2024. Preliminary Integration of Vortex within the OpenPiton Platform. In *2024 Vortex Workshop co-located with MICRO 2024, November 3rd, 2024, Austin, US.* ACM, New York, NY, USA, 2 pages.

1 Introduction

One approach to scaling performance is the adoption of heterogeneous architectures which tightly integrate CPUs, GPUs and specialized accelerators. These systems can be divided into two categories, those with *Coherent Interconnects* and those without. In the former, all compute units have a "single view" of the shared memory, simplifying the programming model. In the latter, the programmer must ensure each compute unit accesses the most recent version of data, using mechanisms such as fences and explicit flushing of caches. Due to the high data bandwidth required by GPUs, traditional hardware-based cache coherency mechanisms do not scale, thus there is active research on relaxed coherency strategies [4]. This research requires access to multi-core, heterogeneous compute platforms able to execute meaningful benchmarks and where the RTL can be modified to evaluate new solutions.

OpenPiton [5] is a widely-used many-core, open-source research platform which can boot SMP Linux. It contains *tiles* which consist of a CPU core, a private L1 cache, an additional private L1.5 cache, a slice of a distributed L2 cache and Network-on-Chip (NoC) routers. Coherency between the L1.5 and L2 caches is maintained using a directory-based MESI protocol. It initially used an OpenSPARC core [5], but has been extended to other CPU including RISC-V [6]. Accelerators such as the NVIDIA Deep Learning Accelerator [6] and eFPGAs [2] have been integrated with OpenPiton.

Vortex-2024, November 3rd, 2024, Austin, US

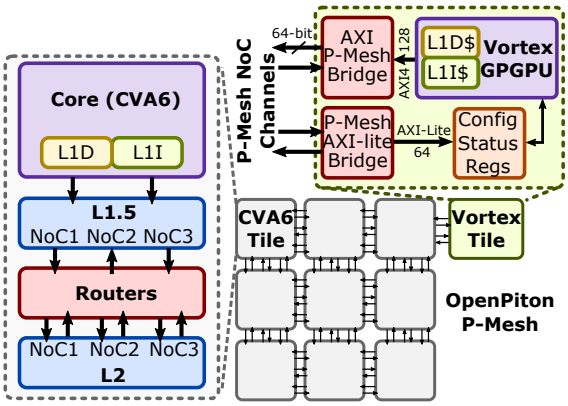

Figure 1: High-Level Integration of Vortex into OpenPiton

Vortex [3] is an open-source (GP-)GPU which is based on RISC-V ISA extensions. It has many configuration parameters, including the number of warps per core, number of threads per warp, size of the caches, etc. Vortex has a complete software stack, relying on the PoCL [7] framework and LLVM to produce a RISC-V binary from OpenCL sources.

We present the first results of an integration of a single Vortex core (intermediate version between 2.1 and 2.2) as a peripheral tile on the edge of OpenPiton, as shown in Figure 1.

2 Hardware Architecture

OpenPiton's NoC and coherence system are what constitutes the P-Mesh. The NoC consists of three 64-bit full-duplex channels for data and cache coherency messages. However, Vortex has an AXI-4 *manager* interface (128-bit bus) to communicate with main memory. We used a simple AXI-4 to P-Mesh bridge, which is limited to one outstanding transaction on the P-Mesh network. The Vortex core was configured with 4 warps of 4 threads, a L1 instruction cache and a L1 data cache of 16 KB with 128-bit cache lines. Vortex has a AXI-4 Lite *subordinate* interface which is used to configure the device by providing the starting address for the kernel and data plus a control bit to start the processing.

Normally, Vortex has a dedicated private memory and the data and kernel are copied here by a general purpose (GP) processor. In our use model, Vortex and the GP processor (CVA6) use the same shared memory, removing the need to copy data back and forth. In this proof-of-concept, the CVA6 MMU is disabled and there is no operating system, making it possible for the GP cores and Vortex to share pointers.

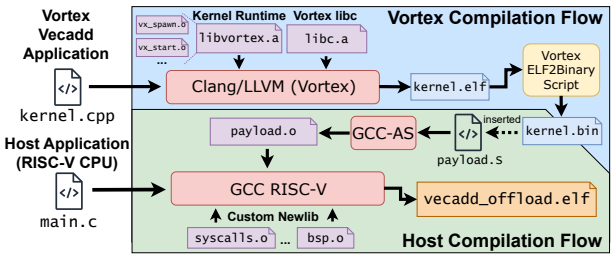

Figure 2: Compilation Flow for Vortex with RISC-V GP Core

The version of Vortex being used includes a write-through L1 data cache. Vortex also supports L2 and L3 caches, but they were disabled for this single core configuration. The AXI to P-Mesh bridge provides IO-cache coherency, meaning that if Vortex modifies data, the copies in the GP processor private caches are invalidated. The reverse is not true, if the GP processor modifies data, there is currently no way to invalidate Vortex's cache due to limitations of both Vortex's caches and the AXI to P-Mesh bridge.

3 Software Flow

For this prototype, a RISC-V bare-metal runtime was used for both GP cores and Vortex core to provide a light *libc* library. Both the device kernel and application are compiled separately (Figure 2). The device binary (`kernel.bin`) is embedded as a *fat binary* in the final executable (`vecadd_offload.elf`). This executable is loaded into shared memory, and then a single GP executes the *main()* function which configures and starts the Vortex core. Vortex then starts fetching code and data directly from shared memory. Currently, the GP core polls a Vortex status register to identify the end of the execution, although this could be managed by interrupts.

For benchmarking, we used the existing `VecAdd` C++ kernel which takes two vectors, computes the sum and writes the result to memory. We vary the number of vector entries to observe how performance scales. To reduce the overhead, we manually inlined the kernel operation in the runtime processing loop (Listing 1), removing the cost of a function call for each kernel evaluation. This change is ad-hoc and prevents the invocation of other kernels. When compiled, this loop contains 23 instructions due to the complex index calculations of *blockIdx.x/y/z* which can handle multi-dimensional tensors, despite this problem being one dimensional.

Listing 1: Vortex Runtime Processing Loop

```
// vx_spawn.c/process_threads() - runtime processing loop
for (task_id = ... ; task_id += threads_per_warp) {
  blockIdx.x = task_id % gridDim.x; // blockIdx is global
  blockIdx.y = (task_id / gridDim.x) % gridDim.y;
  blockIdx.z = task_id / (gridDim.x * gridDim.y);
  // instead of callback(arg) with callback = vecadd
  dst_ptr[blockIdx.x] = src0_ptr[blockIdx.x] + \
      src1_ptr[blockIdx.x];
}
```

4 Initial Performance Results

We compare the performance (vector elements / cycle) obtained for the `VecAdd` kernel executing on the standalone Vortex platform and on our new multicore platform, for different input vector sizes, expressed in bytes (Figure 3). In the standalone platform, the external memory is modeled with Ramulator[1], which gives an average

[1]https://github.com/CMU-SAFARI/ramulator2

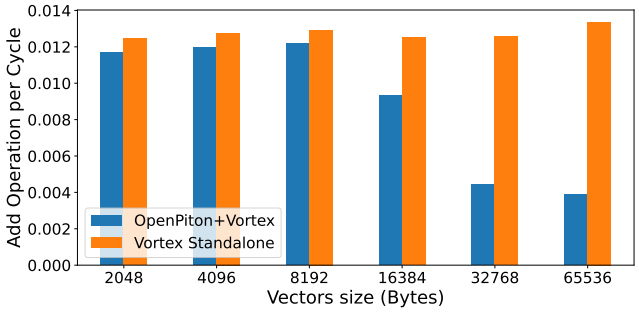

Figure 3: Performance of Inlined `VecAdd` versus Vector Size - Standalone and Heterogeneous Platforms

latency of approximately 20 clocks, whereas in OpenPiton+Vortex we observed the typical memory latency to be 20 or 220 cycles in the case of a L2 hit or miss, respectively. The memory hierarchy of the systems is different, but the comparison still highlights the bottlenecks. In all runs, the cache was "warm" ; that is the kernel was executed a first time, before the performance measurement.

The AXI to P-Mesh bridge requires an aknowledgement from the L2 for any serialized write request, explaining the performance difference below 16KB. The performance of the standalone system is roughly constant with respect to the vector size. This suggests that Vortex is able to mask the latency of the memory through a large number of outstanding requests and its ability to overlap operations in different warps. In our system, we see that the performance drops for vector sizes above 16KB, which corresponds to the size of the L1 data cache. When there are read misses, they are serialized by the AXI to P-Mesh bridge which limits the throughput to one outstanding read miss at a time.

5 Future Work and Conclusions

With our initial prototype, performance is limited for datasets exceeding the L1 cache size, due to the non-pipelined implementation of the AXI to P-Mesh bridge. With a pipelined bridge, the performance should match that of the standalone implementation.

Beyond this performance bottleneck, further work is required to put in place a memory coherency scheme. At a minimum, Vortex caches need to support a software invalidation mechanism. A step further would consist of hardware cache coherency where Vortex caches are invalidated by explicit messages from OpenPiton. With the above enhancements, this prototype should provide a first step towards an open-source heterogeneous CPU/GPU chiplet system, as outlined in [1].

References

[1] Adrian Evans et al. 2024. Invited Paper : Open Source Heterogeneous Chiplet-based Computing Architectures (to appear). In *ICCAD 2024*.
[2] Ang Li et al. 2023. Duet: Creating Harmony between Processors and Embedded FPGAs. https://arxiv.org/abs/2301.02785
[3] Blaise Tine et al. 2021. Vortex: Extending the RISC-V ISA for GPGPU and 3D-GraphicsResearch. (2021). https://arxiv.org/abs/2110.10857
[4] Johnathan Alsop et al. 2018. Spandex: A Flexible Interface for Efficient Heterogeneous Coherence. In *International Symposium on Computer Architecture (ISCA)*.
[5] Jonathan Balkind et al. 2016. OpenPiton: An Open Source Manycore Research Framework. In *ASPLOS 2016*.
[6] Jonathan Balkind et al. 2020. BYOC: A "Bring Your Own Core" Framework for Heterogeneous-ISA Research. In *ASPLOS 2020*.
[7] Pekka Jääskeläinen et al. 2016. pocl: A Performance-Portable OpenCL Implementation. (2016). https://arxiv.org/abs/1611.07083