

Work-In-Progress: Accelerating Numpy With OpenBLAS For Open-Source RISC-V Chips

Cyril Koenig^{1*}, Enrico Zelioli¹, Frank K. Gürkaynak¹ and Luca Benini^{1,2}¹ETH Zurich²Università di Bologna

Abstract

RISC-V allows for building general-purpose computing platforms with programmable accelerators around a single open-source ISA. However, leveraging heterogeneous SoCs within high-level applications is a tedious task. In this preliminary work, we modify the OpenBLAS library to offload selected linear kernels to a programmable manycore accelerator (PMCA) using OpenMP. By linking the Python package Numpy against this library, we enable acceleration of high-level applications. We target an open-source heterogeneous System-on-Chip with a `rv64g` Linux capable host and a `rv32ima` PMCA. Using this platform emulated on FPGA, and the presented software stack, we can accelerate Python applications with linear algebra operators like matrix multiplication.

Introduction

With the democratization of artificial intelligence and machine learning, the demand for embedded and high-performance hardware optimized for linear calculus is continuously growing. In this context, RISC-V will allow for building general-purpose platforms with linear computing accelerators from different vendors around the same open ISA.

Nevertheless, accelerating applications written in high-level languages can be a tedious task. The BLAS API addresses this issue by identifying a list of basic linear algebra operations. Multiple implementations of the API have been proposed, and many high-level applications can leverage their platform-specific optimizations by binding linear algebra operators (e.g., matrix multiplication).

These implementations can target different purposes. OpenBLAS, for instance, features hand-crafted kernels for various architectures and ISAs. BLIS, offers high performance for symmetric multi-threading [1]. IRIS-BLAS [2] targets heterogeneous architectures with general purpose GPUs. However, there is no open source implementation combining hand-crafted RISC-V host kernels and heterogeneous computation with RISC-V programmable manycore accelerators (PMCA).

In this work, we use the Hero software development kit (SDK) presented in [3] to extend OpenBLAS for an open-source RISC-V based heterogeneous system-on-chip (heSoC). We add a heterogeneous implementation of GEneral Matrix Multiply (GEMM) for a `rv32imafd` PMCA. We benchmark our accelerated OpenBLAS implementation from a Python application running on the `rv64g` host-core. Our preliminary result show $2.71\times$ execution speedup when offloading a Numpy matrix multiplication on the device rather than executing it on the host.

Open-Source Platform

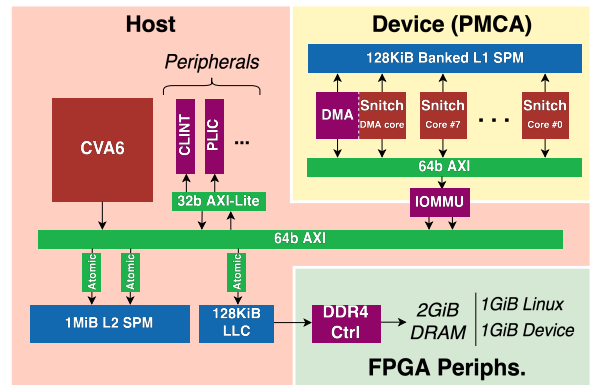


Figure 1: *Open-Source heterogeneous platform with CVA6 and Snitch. The L1 SPM contains the device local data, the dual-port L2 SPM contains constants and device instructions, and the device DRAM contains physically contiguous buffers for shared data structures.*

The platform¹ used in this work is based on Cheshire, a SoC built around the `rv64g` application-class core CVA6. This host is coupled to a PMCA that features eight Snitch cores with double precision FPUs. The accelerator cluster contains 128 KiB of local scratch-pad memory (SPM) that is refilled from the external shared DRAM using a DMA engine. The DRAM is partitioned into two regions, one used by the operating system, and one manually managed to avoid fragmentation. When the IOMMU is not used, shared data structures must be copied to the device DRAM before use. We emulate the platform on a Xilinx VCU128.

Open-Source Software Stack

For this study, we introduce the software stack shown in Figure 2. This stack is built upon previous work for heterogeneous programming.

*Corresponding author: cykoenig@iis.ee.ethz.ch

¹ https://github.com/pulp-platform/carfield/tree/date_iommu_evaluation

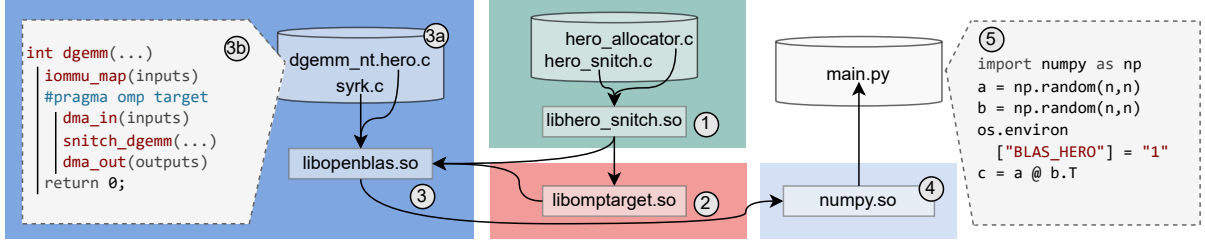


Figure 2: The proposed software architecture. The Hero library ① contains device managements functions. The OpenMP target library ② contains the callbacks for the OpenMP API. The OpenBLAS library ③ contains computing kernels for host and/or device. The Numpy ④ package is linked against OpenBLAS. Finally, the user application ⑤ imports Numpy.

HeroSDK: HeroSDK [3] contains a heterogeneous LLVM (15) compiler, host runtimes, and Linux kernel (6.1.22) modules to build heterogeneous applications with OpenMP. The Hero library ① contains device management functions and calls to the kernel module (for instance, to map the accelerator’s IO space). This library contains device-agnostic files such as *hero_allocator.c*, which manages L2 SPM and device DRAM, and device-specific files like *hero_snitch.c* which boots the device. This allows for quick porting to new open-source accelerators. Then, the OpenMP target library ② implements callbacks for the device offload API via the LibHero. As this stack is implemented in C/C++, it may not be easily usable in high-level applications. Thus, in this work, we extend HeroSDK with OpenBLAS support.

Compiling OpenBLAS with HeroSDK: We included into OpenBLAS (0.3.29) an heterogeneous rv64/rv32 implementation of GEMM. With minor changes to the Makefiles in OpenBLAS, we select kernels to be compiled only for the host like *syrk.c* ③a and kernels to be compiled for the host and accelerator using the HeroSDK LLVM compiler. The pseudo-code of the heterogeneous kernel is provided in ③b. The execution starts on the host and the region within the `#pragma` is compiled and offloaded to the accelerator. The resulting *libopenblas.so* ③b contains the device functions to be copied to L2 before the first offload.

Python test application: We can link Numpy to OpenBLAS ④, and write Python applications ⑤.

Results

We execute the application ⑤ with and without Hero device offloading and measure the execution time using the Python function `os.time()`. In Figure 3, we show the runtime divided into three regions for different problem sizes. During the “data copy” region, the host copies inputs and outputs between the Linux portion and the device portion of the DRAM. During “fork/join” the host enters and exits OpenBLAS and OpenMP. In “compute” the device DMA copies local data and processes them in SPM. One of the key challenges of heterogeneous computing is to keep the “fork/join” and “data copy” overheads as low as possible to reach interesting speedups. Our solution

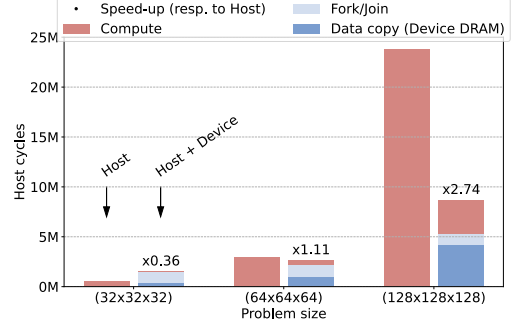


Figure 3: Execution time (measured from Python) for a float64 matrix multiplication with and without offloading.

is 2.71× faster than host-only computation for matrices of size 128. The “data copy” remains the major overhead with 47% of the total runtime. Since the platform features an open-source RISC-V IOMMU [4], future work will focus on removing this overhead via zero-copy offloading. From a previous study on the same platform, we expect creating IO page table entries for this input size to be 7.5× faster than copying, bringing the total speedup to 4.7×. Further improvements can be expected from highly optimized kernels and SIMD operations on lower precision data types.

Discussion

In this preliminary work, we compile OpenBLAS for an open-source RISC-V heSoC. With minor modifications to the codebase, we extend the existing RISC-V host implementation with heterogeneous kernels for RISC-V PMCAs. We verify the benefits of the accelerator on a simple Python application with Numpy. This allows for easily leveraging heterogeneous RISC-V SoCs in high-level applications such as ML frameworks.

References

- [1] *BLIS performance*. <https://github.com/flame/blis/blob/master/docs/Performance.md>. Accessed: 2025-02.
- [2] N. Rao Miniskar et al. “IRIS-BLAS: Towards a Performance Portable and Heterogeneous BLAS Library”. In: *IEEE 29th HiPC*. 2022.
- [3] C. Koenig et al. “HeroSDK: Streamlining Heterogeneous RISC-V Accelerated Computing from Embedded to High-Performance Systems”. In: *IEEE 42nd ICCD*. 2024.
- [4] Manuel Rodríguez et al. “Open-source RISC-V Input/Output Memory Management Unit (IOMMU) IP”. In: *RISC-V Summit Europe, Barcelona*. 2023.