Integration of a CGRA Accelerator with a CVA6 RISC-V Core for the Cloud-edge

Juan Granja, Daniel Vázquez, Alfonso Rodríguez and Andrés Otero

Centro de Electrónica Industrial, Universidad Politécnica de Madrid (UPM), Madrid, Spain

Abstract

In the context of the development of adaptable nodes for the cloud-edge continuum, this work integrates a Coarse-Grain Reconfigurable Array (CGRA) accelerator with an application-class RISC-V processor on a System on Chip. To do so, a DMA interface is devised to provide the CGRA with its configuration data and to manage the transfer of input and output data between the CGRA and memory, all under the control of the RISC-V CPU via memory-mapped registers. The resulting platform is deployed on an FPGA, and its performance is evaluated when accelerating a set of relevant tasks, both in a bare-metal environment and under a Linux operating system. A kernel module is written for the latter, allowing the use of the CGRA accelerator from a Linux user process.

Introduction

The cloud-edge continuum views cloud, edge, and fog as part of a single space where tasks can be scheduled on various devices. Edge devices in this context benefit from application-class processors and reconfigurable accelerators to provide the demanded interoperability, adaptability, and local processing capability. This work is the first step in developing an edge node for the cloud-edge continuum.

Platform Selection

The CVA6 was selected as a host processor for the edge node among other open-source, application-class RISC-V processors for the following reasons: (1) it is written in industry-standard SystemVerilog, (2) it has an ISA extension interface (CV-X-IF) to add custom instructions, and (3) it has an active user community.

The CVA6-based edge node has been developed within a minimal platform maintained by OpenHW Group [1], including a boot ROM and a reduced set of peripherals, all interconnected via an AXI crossbar.

This platform was selected for use in this work due to simulation support for the Verilator open-source simulator and existing configuration files for the KC705 FPGA development board used in this work.

The STRELA CGRA

STRELA [2] is a Coarse Grain Reconfigurable Array (CGRA) targeted at the low-power embedded domain. It comprises a four-by-four grid of Processing Elements (PEs), with four input and four output ports on the top and bottom borders. The PEs can execute integer arithmetic, logical, and comparison operations on 32-bit words, plus loops and branches. A given code fragment to be accelerated is mapped to a CGRA configuration, by hand or architecture-specific compilers.

The STRELA CGRA was initially integrated into an open-source, RISC-V microcontroller platform named X-HEEP [3]. This implementation uses an on-chip 8-way interleaved memory to allow simultaneous access by independent memory modules in the input and output ports of the CGRA. This approach is only practical for limited on-chip memory. A bigger, external memory is needed to run Linux.

CGRA Operation

Figure 1 shows the intended operation of the STRELA CGRA as an accelerator in bare-metal and Linux.

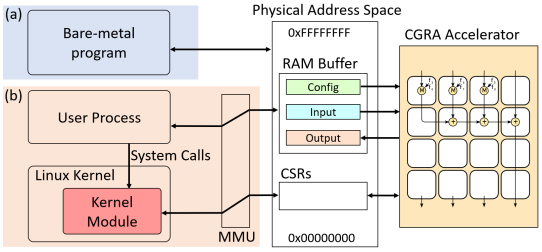

Figure 1: *CGRA operation in (a) bare-metal, (b) Linux.*

CGRA configuration and input data are placed at given addresses in system memory, and space is allocated for the output data. The RISC-V CPU specifies these memory regions and sets a stride for input data through control-status registers (CSRs). Once a valid configuration is loaded into the CGRA, it can process input data and write the results back into memory.

In a bare-metal context, the CPU has direct access to physical memory, including the CSRs. Conversely, under Linux, it is necessary to setup a mapping between a Linux user process and a known region of physical memory. This is done in a kernel module, which is also responsible for accessing the CSRs while

This work has been supported by the MYRTUS project, funded by the European Union with grant number 101135183.
Corresponding author: `juan.granja@upm.es`

Table 1: *CGRA speedup.*

Task	Bare-metal	Linux	X-HEEP
ReLU	8.7x	2.2x	15.4x
2D Convolution	1.1x	4.3x	18.6x

presenting a suitable interface for the user process.

Both cases maintain cache coherency by flushing the data cache before and after the CGRA execution.

CGRA Integration in the SoC

The CGRA was integrated into the selected platform by wrapping it in a module that includes the necessary hardware for data provisioning via DMA and for configuration and control of the accelerator. This module connects to the AXI-4 crossbar interconnect via a master port for DMA and a slave port for access to the CSRs, as seen on Figure 2.

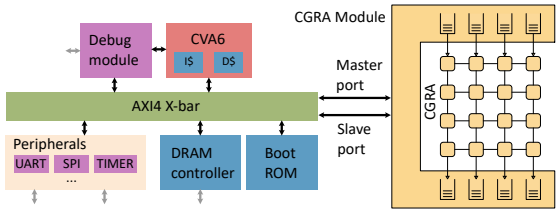

Figure 2: *CGRA integration in the CVA6 platform.*

Due to the need to perform strided memory accesses to support specific workloads, the DMA interface does not use consecutive burst transfers, but single beats. Multiple outstanding transactions allow close to one transaction per cycle, even with high access latency. FIFOs on the CGRA's borders are the destinations and sources for input and output data, respectively. These buffers reduce the CGRA's stall cycles and are needed to support multiple outstanding transactions.

Results and Discussion

The implemented design was evaluated with two benchmarks selected in [2], a ReLU operation and a 2D convolution, both in bare-metal and under Linux. The ReLU task is performed on 32 KB of 32-bit words and the 2D convolution on 64x64 images with a 3x3 kernel. Table 1 shows the obtained speedup with the aid of the CGRA with respect to a pure software solution.

Execution time measurements are detailed in Figure 3 for a ReLU task under Linux. The CGRA time is divided into execution, configuration loading, and setup of the CSRs. Under Linux, overheads not present in bare-metal are included, such as system calls to access the kernel module and memory copy operations to and from the shared RAM buffer. In the case of Figure 3, the latter dominate the total execution time.

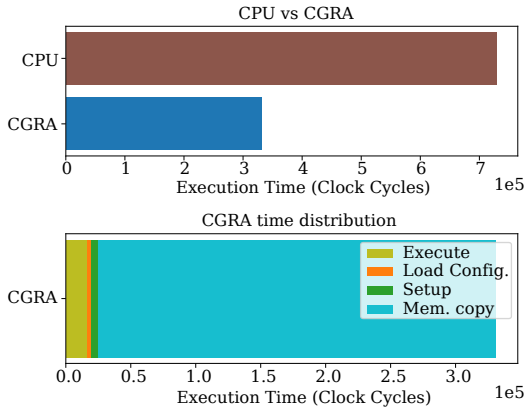

Figure 3: *Execution time of the ReLU task under Linux.*

Notably, the *software* implementations of the ReLU and 2D convolutions are up to five times slower under Linux than in bare-metal. This overhead can be attributed to the Memory Management Unit (MMU) being enabled for address translation, though its mismatch with the 15-30 % overhead mentioned in the literature [4] remains to be investigated.

In bare-metal, the CVA6 STRELA implementation provides 1.7x and 17x less speedup than the X-HEEP implementation for the ReLU and 2D convolution tasks. Although the two implementations are not directly comparable due to the different nature of the host processors (application class and microcontroller class), it is clear that the memory access bandwidth, using a single master port for DMA, is limiting the performance of the CGRA accelerator in the CVA6 implementation. This results in not taking full advantage of the parallel processing capability of the CGRA.

Further work may involve an interleaved accelerator cache to speed up memory access, and configuring the CGRA with custom instructions instead of CSRs.

The reader is referred to [5] for more details on this work.

References

[1] *CVA6 GitHub Repository.* `https://github.com/openhwgroup/cva6`. (Visited on 05/10/2024).

[2] Daniel Vazquez et al. "STRELA: STReaming ELAstic CGRA Accelerator for Embedded Systems". In: *arXiv preprint arXiv:2404.12503* (2024).

[3] Simone Machetti et al. "X-HEEP: An open-source, configurable and extendible RISC-V microcontroller for the exploration of ultra-low-power edge accelerators". In: *arXiv preprint arXiv:2401.05548* (2024).

[4] Guilherme Cox and Abhishek Bhattacharjee. "Efficient address translation for architectures with multiple page sizes". In: *ACM SIGPLAN Notices* 52.4 (2017).

[5] Juan Granja. "Integration of a CGRA Accelerator with a CVA6 RISC-V Core for the Cloud-edge Continuum". Master Thesis. Universidad Politécnica de Madrid, 2024.