An Open-Hardware Coarse-Grained Reconfigurable Array for Edge Computing

Rubén Rodríguez Álvarez, Benoît Denkinger, Juan Sapriza, José Miranda Calero, Giovanni Ansaloni, and David Atienza Alonso
EPFL
Lausanne, Switzerland
ruben.rodriguezalvarez@epfl.ch

ABSTRACT

In this work, we propose an open-hardware low-power coarse-grained reconfigurable array connected to a lightweight microcontroller and enclosed in an application mapping framework. The latter provides complete support to configure kernels in the reconfigurable array, execute applications, and measure performance.

CCS CONCEPTS

• **Computer systems organization → Embedded systems**.

KEYWORDS

CGRA, edge computing, open source, open hardware

ACM Reference Format:
Rubén Rodríguez Álvarez, Benoît Denkinger, Juan Sapriza, José Miranda Calero, Giovanni Ansaloni, and David Atienza Alonso. 2023. An Open-Hardware Coarse-Grained Reconfigurable Array for Edge Computing. In *20th ACM International Conference on Computing Frontiers (CF '23), May 9–11, 2023, Bologna, Italy*. ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3587135.3591437

1 INTRODUCTION

Coarse-grained reconfigurable arrays (CGRAs) are composed of a mesh of processing elements, each typically embedding an arithmetic-logic unit (ALU) and a small register file. They can efficiently host computational kernels derived from the Data Flow Graph (DFG) of applications. Since they are reconfigurable at the operation level, they present a much lower area for control logic and require little time for reconfiguration. Herein, we introduce an open-hardware and low-power CGRA design and its companion framework for application mapping,[1] developed at the Embedded Systems Laboratory (ESL) of EPFL. The CGRA can be integrated into a system as a memory-mapped accelerator connected to the system bus. Indeed, we provide an example of its integration with the X-HEEP microcontroller,[2] which creates a platform able to execute complete applications where some specific kernels can be accelerated by the CGRA.

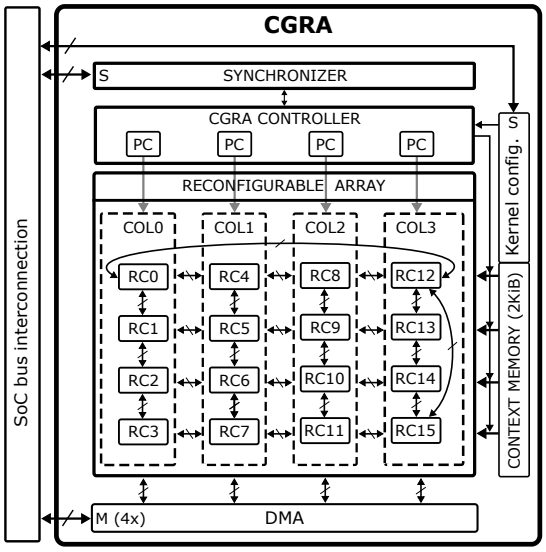

Figure 1: CGRA architecture (M: master port, S: slave port). All RCs have a nearest neighbours interconnection torus, but for simplicity, it is only shown for a few cells (RC0-RC12 and RC12-RC15).

Completing the CGRA framework is a firmware library, written in C, enabling to interface applications with the accelerator. The full system can be simulated for testing and validation using Verilator, Questasim, or VCS. The reconfigurable array size (i.e., the number of reconfigurable cells and their layout) can be changed for design exploration depending on the specific application domain. This reconfigurability will be further simplified in the future and the testing environment extended to also support an FPGA-based emulation platform, which will enable even faster prototyping and validation in a physical device.

2 CGRA ARCHITECTURE

A block scheme of the CGRA architecture is shown in Figure 1. Its default configuration features a four-by-four array of identical reconfigurable cells (RCs), which are interconnected with the four nearest neighbours in a torus configuration to enable data movement within the reconfigurable mesh. Each RC is composed of one

[1]The RTL code and its integration into an open-source microcontroller can be downloaded at https://github.com/esl-epfl/cgra_x_heep.
[2]X-HEEP is an open-source RISC-V based microcontroller for building edge computing platforms https://github.com/esl-epfl/x-heep.

CF '23, May 9–11, 2023, Bologna, Italy

ACM ISBN 979-8-4007-0140-5/23/05.
https://doi.org/10.1145/3587135.3591437

This work has been partially supported by the Swiss NSF ML-Edge Project (GA No. 200020 182009), in part by the Innosuisse WaSTeLeSS Innovation project (Ga No. 51864.1 IP-ENG) and in part by the RESoRT Project (GA No. REG-19-019) from Fondation Botnar.

Table 1: RC 32-bit word instruction format

Field	muxAsel	muxBsel	aluOp	rfSel	rfWe	muxFsel	imm
Bits	31:28	27:24	23:18	17:16	15	14:12	11:0

ALU, two multiplexed inputs, one output register, a four-element register file, and a 32 words private program memory. It is time-multiplexed —each RC executes at each clock cycle the instruction indexed by the column program counter (PC)— enabling the execution of modulo-scheduled loops [2]. Moreover, each RC can use its own internally stored values or the output value of any of its four closest neighbours (top, left, bottom, or right).

The proposed accelerator is connected directly to the main memory through four master ports (one per column), which are controlled by direct memory access (DMA). Additionally, these can access the data in parallel if the system bus allows multiple master-slave transactions simultaneously (this is the default bus configuration for the X-HEEP instantiation provided as a proof-of-concept). Each column stores the addresses for reading and writing, which can be configured from the CPU.

The CGRA instruction set allows computing a wide variety of kernels using 32-bit basic arithmetic and logical operations. Additionally, conditional and unconditional jumps are supported, allowing the mapping of kernels with `if` statements and `for` loops. Note also that jumps are managed by the hardware on a column-by-column basis, as several kernels can be mapped or executed concurrently, and each can require one or multiple columns.

Input/output transfers are managed by direct and indirect loads and stores. Direct loads/stores access to the position in memory configured with the read and write addresses, and it is auto-incremental. Conversely, indirect loads/stores encode the address to be used as part of the instruction, which is then stored in a dedicated register.

The CGRA instruction word format has static bit fields for all the instructions (see Table 1). Fields are linked with the multiplexers that control the operation executed in the ALU and the operands sources. This arrangement waives the need for a decoder and allows the hardware to execute one instruction per cycle for simple operations. The cycle latency of loads and stores depends on the system bus, while multiplications are performed in 3 cycles in order to relax the critical path.

The CGRA comprises a context memory of 2 KiB that stores the instructions describing kernels that can be loaded at run-time in the RCs according to the application requirements. This memory is connected to the system bus, and it is programmed by the CPU. A CGRA synchronizer schedules the acceleration request from the CPU on the available columns of RCs by copying and executing the instructions in the RCs, as detailed in [1]. If a kernel request can be executed in the columns available at run-time, the instructions from the kernel are copied from the context memory to the private memory of the corresponding RCs and the execution is initiated. Otherwise, the synchronizer stalls the request until enough resources are present. Hence, kernels are mapped into specific columns inside the RCs mesh at run-time instead of compile time, resulting in a flexible use of resources. The synchronizer incorporates a set of 32 control slave registers to configure and launch the kernel's execution. A set of these registers also provide performance counters that allow measuring the performance and utilization of kernel execution.

Table 2: Post-synthesis area breakdown of a 4×4 CGRA instance. Total area is $\sim 0.4\,\mathrm{mm}^2$ in 65nm technology.

Block	Ctx & kernel	Sync.	Ctrl.	DMA	Program mem.	ALU & Muxes	Data regs.
Area (%)	7	4	1	2	39	43	4

Finally, the kernel configuration memory (with 15 elements) is used to describe the kernels stored in the context memory. It specifies for each kernel the number of columns required, the start position of the instructions in the context memory, and the number of instructions that the kernel encodes and need to be copied into the RCs private memory.

This CGRA was included in the first ASIC implementation of X-HEEP using the 65 nm low-power LVT TSMC technology node. The area occupied by the CGRA after synthesis is 402 985 $\mu\mathrm{m}^2$. The area breakdown in Table 2 shows that most of the area is occupied by the datapath and the program memories, while the control logic only represents a minor portion of the total area, as opposed to other reconfigurable arrays with greater granularity such as FPGAs. The context memory is built with SRAM macros while the program memories use registers implemented with standard cells, which explains the difference in area size, although both have the same memory size. The clock speed achieved for the complete system is 250 MHz (same clock used by the CGRA).

3 CGRA ASSEMBLER AND FIRMWARE

An assembler is provided to generate binary configuration words from a description of mapped/scheduled operations. The framework eases the programming effort for governing the CGRA execution, as operations can be described in human-readable form. Implementations of benchmark kernels, described as mapped operations in assembly language, are provided within the framework. These kernels are divided into two sets: manually mapped kernels and kernels obtained from the SAT-MapIt [3] modulo scheduling compiler[3].

Also part of the CGRA companion software suite are utilities for the generation of test C code functions, which allow launching kernels from an application to validate the CGRA functionality and to compare performance with respect to software run-time on the host processor, for example. Several kernels can be configured within the same context memory as far as the instructions generated fit in the context memory. In the proof-of-context design we provide, the number of instructions per RC times the number of columns of the kernel for all kernels has to be lower than 128.

REFERENCES

[1] Loris Duch, Soumya Basu, Rubén Braojos, David Atienza, Giovanni Ansaloni, and Laura Pozzi. 2016. A multi-core reconfigurable architecture for ultra-low power bio-signal analysis. In *2016 IEEE Biomedical Circuits and Systems Conference (BioCAS)*. IEEE, 416–419.

[2] Bingfeng Mei, Serge Vernalde, Diederik Verkest, Hugo De Man, and Rudy Lauwereins. 2003. Exploiting loop-level parallelism on coarse-grained reconfigurable architectures using modulo scheduling. *IEE Proceedings-Computers and Digital Techniques* 150, 5 (2003), 255–261.

[3] Cristian Tirelli, Lorenzo Ferretti, and Laura Pozzi. 2023. SAT-MapIt: A SAT-based Modulo Scheduling Mapper for Coarse Grain Reconfigurable Architectures. In *To appear in DATE'23*.

[3] SAT-MapIt is an open-source, SAT-based compiler for CGRAs, available at https://github.com/CristianTirelli/SAT-MapIt.