EXTRA: An Open Platform for Reconfigurable Architectures

Cătălin Bogdan Ciobanu
Giulio Stramondo
Ana Lucia Varbanescu
University of Amsterdam

Lorenzo Di Tucci
Marco Rabozzi
Luca Stornaiuolo
Marco Santambrogio
Politecnico di Milano

Andreas Brokalakis
Antonis Nikitakis
Synelixis Solutions

Grigorios Chrysos
Charalampos Vatsolakis
Charitopoulos Georgios
Dionisios Pnevmatikatos
Telecommunication Systems Institute

ABSTRACT

Reconfigurable hardware is becoming increasingly mainstream, evolving to a valid alternative to Graphics Processing Units-based hardware accelerators. However, several major challenges remain for migrating existing software to heterogeneous reconfigurable architectures. The EXTRA project aims to develop an integrated environment for developing and programming reconfigurable architectures. The EXTRA platform enables the joint optimization of architecture, tools, and reconfiguration technology, and targets the future High Performance Computing hardware nodes. In this paper, we present four innovative EXTRA technologies: (1) a hardware-software co-design framework; (2) a parallel memory system; (3) a decoupled access execute framework for reconfigurable technology; and (4) transparent access and virtualization of reconfigurable hardware accelerators. Moreover, we describe how the EXTRA technologies targeting the Amazon F1 cloud compute instances can be used in medical applications such as the retinal image segmentation.

CCS CONCEPTS

• **Computer systems organization** → **Reconfigurable computing**; **Data flow architectures**; • **Hardware** → **Hardware accelerators**; **Hardware-software codesign**;

KEYWORDS

FPGA, Exascale, Polymorphic, PolyMem, CAOS, DAER, RACOS

ACM Reference Format:

Cătălin Bogdan Ciobanu, Giulio Stramondo, Ana Lucia Varbanescu, Andreas Brokalakis, Antonis Nikitakis, Lorenzo Di Tucci, Marco Rabozzi, Luca Stornaiuolo, Marco Santambrogio, Grigorios Chrysos, Charalampos Vatsolakis, Charitopoulos Georgios, and Dionisios Pnevmatikatos. 2018. EXTRA: An Open Platform for Reconfigurable Architectures. In *2018 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XVIII)*, July 15–19, 2018, Pythagorion, Samos Island, Greece. ACM, New York, NY, USA, 10 pages. <https://doi.org/10.1145/3229631.3236092>

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

SAMOS XVIII, July 15–19, 2018, Pythagorion, Samos Island, Greece

© 2018 Copyright held by the owner/author(s).

ACM ISBN 978-1-4503-6494-2/18/07.

<https://doi.org/10.1145/3229631.3236092>

1 INTRODUCTION

Reconfigurable hardware gains popularity in a wide range of domains, ranging from smartphones [1] and computer monitors [2] to game consoles [3]. Moreover, reconfigurable hardware is also used in cloud computing [4] and compute instances in the cloud such as the Amazon EC2 F1 Instances [5]. Reconfigurable devices like Field Programmable Gate Arrays (FPGAs) are becoming a valid High Performance Computing (HPC) alternative to Graphics Processing Units (GPUs), as they provide very high computational performance with superior energy efficiency by employing customized datapaths.

However, several major challenges remain for the programmers who try to migrate from traditional CPU-based work-flows to heterogeneous systems using reconfigurable hardware accelerators. In this paper, we introduce an open platform that aims to address several of these challenges, facilitating easier development for programmers and an opportunity for the research community to work together on further bridging the gap between traditional software development and employing reconfigurable accelerators.

The open platform is described in Section 3. Developed in the context of the Exploiting eXascale Technology with Reconfigurable Architectures (EXTRA) project [6–9], the platform provides an integrated environment for application design, development, and deployment on reconfigurable architectures. The idea of this new and flexible exploration platform is to enable the joint optimization of architecture, tools, applications, and reconfiguration technology in order to prepare for the HPC machines of the future.

Our open research platform integrates several innovative technologies: CAD as an Adaptive Open-platform Service (CAOS), a Polymorphic Parallel Memory (PolyMem), Just-in-time Synthesis, Virtual Coarse-Grained Reconfigurable Array (VCGRA), Decoupled Access-Execute architecture for Reconfigurable accelerators (DAER), Reconfigurable ACcelerator OS (RACOS), performance modeling, and the Artisan Meta-Programming framework. In this paper, we present four of these main EXTRA technologies: CAOS, PolyMem, DAER, and RACOS, and one target application in the medical domain - Retinal Image Segmentation.

Despite the efforts in High Level Synthesis (HLS) tools, the acceleration of existing algorithms on FPGA-based High Performance Reconfigurable Computers (HPRCs) is still challenging and requires

highly experienced developers with a deep knowledge of hardware-software co-design. To alleviate these challenges, EXTRA proposes CAOS, a toolchain that targets the full stack of the application optimization process, starting from the identification of the kernel functions amenable to hardware acceleration, to the optimization of such kernels and the generation of the runtime management for the target system. CAOS is an open project, aiming to foster the adoption, development and improvement of HPRC systems in the community. We present CAOS in Section 4.

FPGAs are tightly linked to parallel processing applications where memory bandwidth is a hot commodity. For many applications, ranging from graph processing to machine learning, and from scientific simulations to financial analysis, more bandwidth is likely to amount to better performance. One approach to address this demand for increased bandwidth is to re-think existing memory systems. Newly-emerging technologies [10, 11] hold promise, but their large-scale integration depends on the processor vendors and is, therefore, rather slow. A more viable solution is to design and develop parallel memories, which could provide an immediate memory bandwidth increase as large as the number of parallel lanes. While this proposal sounds straightforward in theory, many challenges emerge when designing and/or implementing such memories in practice [12]. Writing the data efficiently, reading the data with a minimum number of accesses and maximum parallelism, and actually using such memories in real applications are only three of these challenges which we address in Section 5.

Offloading computationally intensive kernels from a CPU to a reconfigurable accelerator is not trivial. In Section 6, we describe DAER and how it can be applied for high-performance reconfigurable acceleration of different classes of applications. Moreover, we present some initial performance results of combining this framework with the EXTRA Parallel memory system in Section 7. Lastly, our Reconfigurable ACcelerator OS (RACOS) provides a simple interface between software and reconfigurable hardware. We introduce RACOS and evaluate its scheduling capabilities in Section 8.

Section 9 describes our Retinal Image Segmentation case study.

2 RELATED WORK

Recent technological advances allow the integration of reconfigurable hardware, alongside with general-purpose processors (CPUs) to form an efficient HPC system. In recent years, several efforts have been undertaken to improve HPC systems in order to reach exascale performance. The European High-Performance Computing Joint Undertaking (EuroHPC JU) [13] is a new organization that focuses on developing a world-class exascale supercomputing infrastructure based on competitive technology from European projects. Some of the European projects that focus on exascale computing are described in this section.

ECOSCALE [14] introduces a novel heterogeneous energy efficient hierarchical architecture and a hybrid programming environment for automatic execution of exascale applications onto an HPC platform. The framework supports thousands of reconfigurable hardware blocks, while taking into account the projected trends and characteristics of HPC applications. The ECOSCALE approach is hierarchical and scales well by partitioning the physical system into multiple independent compute nodes.

Figure 1: The EXTRA Open Research Platform

ExaNest [15] supports a ground-breaking computing architecture for exascale-class systems built upon power-efficient 64-bit ARM processors. The project proposes alternative storage and interconnect options on actual hardware, using real world HPC applications.

ExaNoDe [16] focuses on the design of a highly energy efficient and highly integrated heterogeneous compute node targeting exascale level computing. The design of the target node mixes low-power processors, heterogeneous co-processors and includes advanced hardware integration technologies with a novel memory system.

AllScale [17] provides a parallel programming model for the effective development of highly scalable, resilient and performance-portable parallel applications for exascale systems. Their approach is based on task-based nested recursive parallelism and it provides mechanisms for automatic load-balancing in the simulations.

Another project that focuses on developing a programming environment that will enable the productive deployment of highly parallel applications in exascale computing systems is EXA2PRO [18]. EXA2PRO will integrate tools that will address significant exascale challenges, i.e., source code quality improvement, efficient exploitation of exascale systems' heterogeneity and data and memory management optimization.

LEGATO [19] focuses on addressing the problem of power- and energy-efficient computing for heterogeneous platforms. This work started with a mature software stack implementation, and will optimize this stack to support energy-efficient computing on a commercial cutting-edge CPU-GPU-FPGA heterogeneous hardware substrate and FPGA-based Dataflow Engines (DFEs). Their performance results show that their direction will increase energy efficiency up to an order of magnitude.

The state-of-the-art research studies described above address different exascale computing aspects. In this context, EXTRA is unique because it attempts to bring together different technological aspects from the lowest level, i.e., computing IPs and custom hardware architectures, all the way up to the highest levels, i.e., a flexible platform and tools, for efficient HPC applications deployment. All the novel EXTRA frameworks and tools are tested on a set of diverse and substantial applications, relevant to exascale computing.

Figure 2: Overview of the CAOS flow and its components.

3 THE EXTRA OPEN RESEARCH PLATFORM

Illustrated in Figure 1, the EXTRA Open Research Platform supports the optimization of applications for the next-generation reconfigurable HPC systems. For this purpose, this platform provides an integrated system that combines models for various reconfigurable architectures, tools and applications, allowing researchers to focus on these aspects individually or as a whole.

The platform **Input** includes the definition of the workload (application source-code and code annotations, profiling data, performance requirements) as well as hardware platform requirements (size and type of resources). The **Hardware Platform Model** is a high-level specification of the target platform used for execution, which guides the design and optimization processes. The **CAOS Tools Platform** produces a reconfigurable design through a set of analyses and transformations by combining knowledge of application characteristics, application requirements, profiling data, and the hardware platform model. The **Performance Modelling** component checks whether performance goals are feasible. In addition, it provides scalability estimates to determine the feasibility of exascale computing by performing advanced performance modeling techniques that forecast exascale workload requirements and performance. The most efficient **reconfigurable design** is selected based on the performance and hardware models used, and consists of a collection of hardware kernels derived with different optimization techniques, and an application driver that realizes the application logic through the platform runtime system. The **Reconfigurable Platform** corresponds to the actual hardware machines and simulators used for design execution, and its runtime system.

4 HARDWARE SOFTWARE CO-DESIGN FRAMEWORK

This section presents the design flow and the main features of the CAOS framework [20], our fully integrated platform for assisting and automating High Level Language (HLL) applications acceleration on reconfigurable HPC systems. CAOS also provides practical interfaces to enable plugging in extensions and enhancements, thus encouraging contributions from the entire community, further facilitating the adoption of reconfigurable hardware in HPC.

Figure 3: CAOS framework the modules interactions

4.1 Introduction to the CAOS flow

CAOS expects the application designer to provide the following: application code written in a HLL such as C/C++, one or multiple datasets to be used for code profiling, and a description of the target reconfigurable system. In order to simplify the set of possible optimizations and analyses that can be performed on a specific algorithm, CAOS allows the user to accelerate its application using one of the available *architectural templates*. An architectural template is a characterization of the accelerator in terms of both its computational model and the communication with the off-chip memory. As a consequence, an architectural template constrains the architecture to be implemented on the reconfigurable hardware and poses restrictions on the application code that can be accelerated, so that the number and types of optimizations can be tailored for a specific type of implementation. Furthermore, CAOS is meant to be orthogonal and build on top of tools that perform HLS, place and route, and bitstream generation. Code transformations and optimizations are performed at the source code level, while each architectural template has its specific requirements in terms of HLS and hardware synthesis tools.

CAOS currently supports three different architectural templates: SST (Single Stencil Time-Step), Master-Slave, and Dataflow. SST provides an architecture targeted for stencil codes written in C [21]. Within this context, CAOS offers a design exploration algorithm [22] that jointly maximizes the number of SST processors that can be instantiated on the target FPGA, and identifies a floorplan of the design that minimizes the inter-component wire-length in order to allow implementing the system at a higher frequency. The Master-Slave architectural template [23] poses less restrictions on the final accelerator and source code. It requires a C/C++ application kernel that can be efficiently tiled, so that the resulting accelerator operates on a subset of the application data that is block-transferred to and from the FPGA local memory. Finally, the Dataflow architectural template, powered by the OXiGen tool [24], targets dataflow applications and currently exploits the MaxCompiler within the backend to efficiently implement the application as a Maxeler DFE.

As shown in Figure 2, the overall CAOS design flow is subdivided into the frontend flow, the function optimization flow, and the backend flow. The main goals of the frontend are to analyze the application provided by the user, match the application against one or more architectural templates, profile the user application against the user specified datasets and, finally, guide the user through the hardware/software partitioning of the application to define the kernels to be implemented on the reconfigurable hardware. The function optimization flow performs static analysis and hardware

Figure 4: Using PolyMem as a parallel cache

resource estimation of the kernels to be accelerated on the FPGA. These analyses depend on the considered architectural template. The results of the analysis are used to estimate the performance of the hardware functions and to derive the optimizations to apply (such as loop pipelining, loop tiling, and loop unrolling).

After one or more iterations of the function optimization flow, the resulting kernels are given to the backend flow, in which the desired architectural template for implementing the system is selected and the required HLS and hardware synthesis tools are leveraged to generate the final design. Depending on the CAOS project settings, the backend will either generate the final bitstream or a hardware design ready to be synthesized. The last option allows the user to perform extra manual tuning. Within the backend, CAOS also takes care of generating the host code for running the FPGA accelerators, and optionally guides the place and route tools by floorplanning the system components.

4.2 The CAOS infrastructure

Each of the frontend, functions optimization, and backend flows consists of several modules. From an infrastructure perspective, as shown in Figure 3, the CAOS framework is designed as a microservices architecture in which each module is deployed in a separate Docker application container [25]. The modules interact with a centralized CAOS flow manager, responsible for storing the state of the design flow and orchestrating the execution of the modules. The application designer can interact with the framework via a web-based user interface that communicates with the CAOS flow manager. A module only needs to implement a set of well defined Representational State Transfer APIs in order to interact with the flow manager. The data exchange is performed using JSON files, whose structure depends on the specific module being considered, and raw archives, such as the datasets used for application profiling and the application source code.

Thanks to its web-oriented architecture, the CAOS framework can be easily deployed as-a-service on one or more cloud instances. This is particularly interesting considering the availability of the Amazon Web Services (AWS) F1 instances which currently feature Xilinx Virtex UltraScale+ FPGAs. The CAOS framework takes full advantage of such instances and allows the user to develop a hardware FPGA-accelerated application directly in the cloud.

To use AWS F1, the user has to provide a hardware description matching a F1 instance, and leverage the Master-Slave architectural template, for which CAOS provides an integration with SDAccel. After having performed the CAOS frontend and functions optimization flows, the backend performs two steps. First, it generates the bitstream necessary to run the FPGA program with SDAccel. Secondly, CAOS automatically generates the host code for the application by properly instantiating all the memory objects to host the input and output buffers, and the logic for the invocation of

Figure 5: PolyMem supported access patterns

Table 1: The PRF memory access schemes

PRF Access Scheme	Available Access Patterns
ReO (Rectangle Only)	Rectangle
ReRo (Rectangle, Row)	Rectangle, Row, Main and secondary Diagonals
ReCo (Rectangle, Column)	Rectangle, Column, Main and secondary Diagonals
RoCo (Row, Column)	Row, Column, Rectangle
ReTr (Rectangle, Transposed Rectangle)	Rectangle, Transposed Rectangle

the accelerated kernel. The user does not need to modify the original application to integrate the accelerated kernel. The generated system is ready to be used on the F1 instances.

5 THE EXTRA PARALLEL MEMORY SYSTEM

To address the challenges related to the design and practical use of parallel memory systems, we propose PolyMem [26], a Polymorphic Parallel Memory. We envision PolyMem as a high-bandwidth, two-dimensional (2D) memory which is *used to cache performance-critical data right on the FPGA chip*, making use of the existing distributed memory banks (the BRAMs). We chose a 2D address space for PolyMem to allow the programmers to easily place data structures such as vectors and matrices in this smart buffer, thus decreasing the need for complex index computation typically needed for a traditional, linear access memory. Furthermore, using polymorphism, PolyMem not only delivers high performance for the most common two-dimensional access patterns (such as rows, columns, rectangles, or diagonals), but it also enables combining several such patterns in the same application. Finally, by supporting customization of capacity, bandwidth, number of read/write ports, and different parallel access patterns, PolyMem allows users to configure the parallel memory to fit their application.

Figure 4 depicts the envisioned system architecture. The FPGA board, featuring its own high-capacity DRAM memory, is connected to the host CPU through a PCI Express link. PolyMem acts like a high-bandwidth, 2D parallel software cache, able to feed an on-chip application kernel with multiple data elements in every clock cycle.

PolyMem is inspired by existing research on the Polymorphic Register File (PRF) [27–29]. While the PRF was designed as a runtime customizable register file for Single Instruction, Multiple Data (SIMD) co-processors, PolyMem is tailored for FPGA accelerators for HPC, which require high bandwidth but do not necessarily implement full-blown SIMD co-processors and their corresponding instruction sets on the reconfigurable fabric. When using FPGAs, PolyMem uses a small fraction of the FPGA resources to configure itself to match the workload, and makes use of Block RAM (BRAM) to implement the parallel memory.

Figure 6: Customizing PolyMem for the Application

5.1 The Polymorphic Register File

A PRF is a parameterizable register file, which can be logically reorganized by the programmer or a runtime system to support multiple register dimensions and sizes simultaneously [30]. The simultaneous support for multiple conflict-free access patterns, called *multiview*, is crucial, providing flexibility and improved performance for target applications. The polymorphism aspect refers to the support for adjusting the sizes and shapes of the registers at runtime. In Table 1, each *multiview* scheme (ReRo, ReCo, RoCo and ReTr) supports a combination of at least two conflict-free access patterns. In this work, we reuse the PRF conflict-free parallel storage techniques and patterns, as well as the polymorphism idea to design PolyMem. Figure 5 illustrates the set of access patterns supported by the PRF and, ultimately, by PolyMem. In this example, a 2D logical address space of 8×9 elements contains 10 memory Regions (R), each with different size and location: matrix, transposed matrix, row, column, main and secondary diagonals. Assuming a hardware implementation with eight memory banks, each of these regions can be read using one (R1-R9) or several (R0) parallel accesses.

By design, the PRF optimizes the memory throughput for a set of predefined memory access patterns. For PolyMem, we consider $p \times q$ memory modules and the five parallel access schemes presented in Table 1. Each scheme supports dense, conflict-free access to $p \cdot q$ elements¹. When implemented in reconfigurable technology, PolyMem allows application-driven customization: its capacity, number of read/write ports, and the number of lanes can be set pre-runtime (or even at runtime using partial reconfiguration), to best support the application needs.

In summary, PolyMem uses the technology developed for the PRF to build a parallel memory (Figure 5) for three reasons: (1) it provides a generic, out-of-the-box solution to implement a parallel memory, thus avoiding error-prone, time-consuming custom memory design; (2) it can be customized for the application at hand; (3) its multi-view property allows 2D arrays to be distributed across several BRAMs, enabling runtime parallel data access using *multiple, different "shapes"* without the need for hardware reconfiguration (see Table 1). Effectively, with the PRF-based PolyMem, programmers can assume a parallel memory and focus on algorithm optimizations rather than complex data transformations or low-level details.

5.2 Customizing PolyMem for the Application

To customize PolyMem for a given application [31, 32], we start from the application memory access pattern, for which we find the *optimal parallel access schedule* - i.e., the best sequence of parallel

¹In this work, we will use "x" to refer to a 2D matrix, and "." to denote multiplication.

Figure 7: MAX-PolyMem Implementation Block-diagram

accesses to the application data - for each potential configuration (scheme, capacity, lanes), as depicted in Figure 6. To determine the optimal schedule we formulate the problem as a set covering problem, using Integer Linear Programming (ILP) or a heuristic for the search itself. We finally select the best configuration based on two metrics: speedup and efficiency. The hardware implementation can then be customized accordingly. This generic methodology allows application-driven customization of non-redundant parallel memories with predefined parallel access patterns.

The Gauss-Seidel kernel was used as a case study [31]. The experimental results suggest that a speedup of up to 9.85X can be obtained when compared to a system employing a sequential memory, showing that such memories can be effective for improving the memory bandwidth.

5.3 MAX-PolyMem: PolyMem for DFEs

To enable a quick design and benefit from a high-level programming abstraction, our first prototype of PolyMem, called MAX-PolyMem, was implemented using Maxeler's platform and their MaxJ programming model. MAX-PolyMem is open-source [33].

Figure 7 shows a diagram describing MAX-PolyMem, our MaxJ PolyMem implementation. 2D parallel application accesses are made using two coordinates, (i, j) , and the shape of a parallel access, $AccType$. $DataIn$ and $DataOut$ represent the data which is written to and read from MAX-PolyMem. The core of MAX-PolyMem's design consists of a 2D array of memories ($p \times q$ BRAMs). These are used to store the data in a distributed manner. In Figure 7, eight such memories are illustrated (M0-M7); these are the **Memory Banks**. The number of banks defines the number of data elements which are read/written in parallel per data port.

Based on the (i, j) coordinates and the requested access type $AccType$, the Address Generation Unit (AGU) expands the parallel

Listing 1: Matrix products

```

for (int i = 0; i < DIM; i++)
  for (int j = 0; j < DIM; j++) {
    C[i][j] = 0;
    D[i][j] = 0;
    for (int k = 0; k < DIM; k++) {
      C[i][j] += A[i][k] * B[k][j];
      D[i][j] += B[i][k] * A[k][j];
    }
  }

```

access in its individual components by computing the coordinates of all the accessed elements ($p \times q$ addresses in total). The Module Assignment Function (MAF) is a mathematical function that maps each element in the 2D address space to one of the **Memory Banks**. The MAF guarantees conflict-free access to the supported access patterns. In this work, we use the five MAFs listed in Table 1 and described in detail in [28]. **M** implements all the MAFs supported by our design and outputs the select signal in the **Shuffles**. The Addressing Function **A** computes, for each accessed element, the intra-memory bank address.

In [26], a multi-dimensional Design Space Exploration (DSE) approach was used, where the capacity, number of lanes, and number of read ports for each PolyMem scheme were empirically evaluated. Our results show that (1) MAX-PolyMem can utilize the entire capacity of on-chip BRAMs, allowing the instantiation of a 4MB parallel memory on the Maxeler Vectis DFE; (2) the MaxJ design delivers up to 22GB/s write bandwidth and up to 32GB/s aggregated read bandwidth using up to 4 read ports, at a clock frequency of up to 202MHz, and (3) we are able to utilize all the available BRAMs with reasonable logic utilization.

Finally, to determine whether any unexpected bandwidth limitations occur when using MAX-PolyMem in practice, we have designed and implemented the STREAM benchmark [34], which measures the bandwidth of different in-memory array operations. Using the COPY component of STREAM, we measured the bandwidth of a polymorphic memory with 1 read and 1 write port, and found that we achieve over 99% of the calculated peak performance.

5.4 PolyMem Integration into CAOS

One of the important goals of the CAOS architecture is to allow external researchers to easily integrate their own modules within the CAOS design flow, either for benchmarking purposes, or to enhance the capability of the system. Furthermore, the CAOS modules do not need to reside on the same physical system: the CAOS flow manager only needs to know the IP address and port of the modules in order to interact with them. Thanks to the flexibility of the CAOS architecture and to the adaptability of the PolyMem technology, we were able to integrate the EXTRA Parallel Memory System within the CAOS optimization flow for the master-slave architectural template.

In order to perform the integration and leverage the support of Vivado HLS (used as part of the master-slave architectural template), we developed a C/C++ version of PolyMem. This allows the EXTRA Parallel Memory System to be available for different platforms, such as the ones from Xilinx. We further enhanced the CAOS performance estimation module to evaluate PolyMem’s utilization of the on-chip memory and estimate the potential performance gains.

Figure 8: Decoupled Access-Execute architecture for Reconfigurable accelerators (DAER)

Finally, the CAOS code optimization module has been extended to generate the necessary code for integrating PolyMem into the application kernel.

As an example, the application described in Listing 1 performs two matrix multiplication operations ($C = A \times B$ and $D = B \times A$).

The CAOS performance estimation module detects that the accesses to the four matrices depends on the loops indexes i , j , and k . In particular, A and B are accessed both row-wise and column-wise within the innermost k -loop. For this reason, with a fixed on-chip memory partitioning (e.g., row-wise and column-wise) for both these matrices, we have different parallelism at the hardware level, when the innermost loop is executed. With the PolyMem RoCo access scheme, we allow *both* rows and columns to be accessed in parallel. Indeed, implementing the two matrices with the PolyMem technology allowed parallel accesses for both A and B , at the cost of a moderate increase of the hardware resources utilization, and avoids on-chip data duplication and additional delays due to unnecessary data transfers. The implementation of this simple example for matrices of dimension $DIM = 96$ implemented with a 16-bank PolyMem ($p = 4$ and $q = 4$), on a Xilinx Virtex-7 VC707 with a clock frequency of 100MHz, occupied 1.04% additional LUTs and 1.60% of additional Registers, and achieved a speedup of 5X, compared to the implementation based on default BRAM row-wise and column-wise block partitionings with the same partitioning factor ($p \cdot q = 16$).

6 DECOUPLED-ACCESS EXECUTE FRAMEWORK

The efficient mapping of computational intensive applications on reconfigurable technology focuses on two directions: (a) the process-plane, i.e., efficient interconnected units that accelerate the actual data processing, and (b) the data access-plane, i.e., efficient ways to access data in memory and transfer them to/from the accelerator. The process-plane implementation is fairly well understood and there are mature (HLS) tools which produce efficient reconfigurable architectures. The data access-plane, however, is much more challenging. The data fetch for Big Data and HPC applications has proven to be even more complex and time consuming than the respective processing.

Figure 9: DAER Architecture for SMVM

Towards these directions, we present a framework that produces **Decoupled Access-Execute** architectures for **Reconfigurable accelerators** (DAER) and it is motivated by the idea of **Decoupled Access-Execute** (DAE) architectures [35]. DAE split code regions into two distinct phases: a memory-bound access phase and a compute-bound execute phase. The memory access phase performed all the necessary operations for transferring data to and from memory, while the execute phase addressed the algorithm's computations. The DAER framework splits the target application into parallel kernels. Following the DAE paradigm, each kernel consists of two parts: the data processing part, which is used solely for performing calculations, and the data fetching part, which is used for memory transactions of input and output data. The proposed framework offers a structured and well-defined way of executing applications on reconfigurable logic with really good performance results.

DAER has several advantages. First, it can be applied for high-performance reconfigurable acceleration of different classes of applications. Also, the DAER-based architectures can be implemented on platforms with different characteristics. Last, the DAER framework is very flexible, as it supports both stream-based and arbitrary memory accesses, it can target applications with high data access dependencies, and works with either single or multiple kernels.

6.1 DAER Framework

This section presents the DAER framework and its basic components. Next, we describe the code transformations that the user has to make in the original application code in order to map an application to the target architecture.

DAER components. The DAER framework architecture, depicted in Figure 8, is generic and it can be used for utilizing a single or several accelerators, with or without inter-accelerator communication. Each mapped accelerator is based on the combination of two reconfigurable units, i.e., the **fetch unit** and the **processing unit**.

Each **fetch unit** is connected directly to the CPU, the external memory, and the neighboring fetch units. The CPU connection is used for passing parameters to the fetch module regarding the application's memory traces, i.e., starting memory addresses, array sizes, etc. The memory connection is used for fetching input data and sending back the processed results. Finally, the fetch unit(s) can

directly access data produced by a neighboring accelerator residing on the reconfigurable platform.

The second component of the DAER architecture is the **processing unit**, which implements the logic and/or arithmetic operations. The processing unit works as a simple data-flow engine. FIFO-based links enable the communication between the processing and fetch units, and are used for passing data and sending results.

Both units are amenable to code-specific acceleration through HLS directives. They can be instantiated multiple times according to the needs of the mapped application and the available resources.

DAER Architecture. In order to "pass" an application through DAER, the application has to be first split in two parts: the hardware-based units that perform the memory accesses, i.e., fetch units, and the processing units that implement the main algorithmic workload.

The memory access dependencies are resolved by distributing memory accesses to separate pipeline units, i.e., fetch units, which send read requests and receive data concurrently. Thus, the proposed architecture can disaggregate memory accesses by dividing each fetch unit internally into smaller ones that work in a pipelined parallel way.

The DAE-based code will be annotated with specific HLS directives for creating the desired hardware architecture and the proper I/O interfaces:

- **Pipeline, Dataflow, Loop Unroll and Loop Merge** directives are used for creating fast pipelined modules for both the fetch and the process units.
- **FIFO** directives are used for building low-latency FIFO-based interfaces between the fetch and the process units.

6.2 Case Study and performance evaluation

Our case study application is Sparse Matrix Vector Multiplication (SMVM), for which the sparse matrices are stored using the Compressed Sparse Row (CSR) format. SMVM on modern heterogeneous architectures is a challenging task, and its implementation on reconfigurable hardware could lead to many drawbacks. First, the sparsity of the input matrices cannot be predefined, as it is completely different for various application domains. Second, the sparse matrix alignment in memory is of high importance, as far as the memory access patterns and the overall performance of the system are concerned. CSR storage also leads to pointer chasing and data dependencies, which are detrimental to performance. Finally, the matrix representation format, together with the sparsity variation, often introduce irregularity in memory access patterns, making it difficult to achieve high performance when using a regular execution model.

Figure 9 presents the proposed DAER-based architecture for SMVM. Modules communicate in a streaming scheme using FIFOs. Memory access dependencies are resolved by the pipeline architecture that maps independent fetch units, which access the global memory in parallel. It is important to emphasize the way that the dependencies of the mapped application are resolved. Specifically, the architecture of the implemented fetch unit consists of four sub-units (Figure 9). The first reads the number of non-zero elements per line of the sparse matrix before starting the main processing. The second fetches the indices of the sparse matrix non-zero elements, while the third one fetches the values of the CSR matrix and

Figure 10: DAER and PolyMem combination for SMVM

vector, and streams them to the process unit. The process results are sent back to memory via the fourth fetch subunit.

The proposed architecture was mapped on two HPC platforms, i.e., Micron Hybrid Memory Cube (HMC)[36] and Convey HC-2ex, with high data I/O rate capabilities. The mapped HPC systems achieved up to 1.5X - 2.5X performance acceleration when compared to the best optimized software solution.

Based on the DAER mapping for the case study algorithm, we make the following observations. First, the DAER-based architectures achieve high parallelization level due to parallel read requests and data access ports. Second, the DAER framework solves the problem of data dependencies by pipelining read requests using parallel fetch sub-units. Last, the non-streaming read requests are grouped, so that they are streamed by different mapped fetch subunits and the results are pipelined through the network of the fetch modules until the processing module, where they are processed.

7 DAER AND POLYMEM INTEGRATION

As described in the previous Section, DAER is a generic framework which can be used for accelerating algorithmic workloads with various parameters on reconfigurable hardware. Its main advantages is the high level of parallelism and the different memory access schemes that it can support using either independent parallel fetch units or pipelined sub-fetch units. On the other hand, PolyMem is based on parallel memories that support parallel data accesses based on specific memory access patterns.

We integrated DAER and PolyMem in order to show the performance advantages that their combination can offer for accelerating various algorithmic workloads on reconfigurable hardware. We focused on the data access parallelism and ways for overcoming internal algorithmic data dependencies of the target applications.

7.1 Case Study and performance evaluation

The DAER-based architecture for the SMVM algorithm was presented in Figure 9. As described in Section 6, the proposed architecture is highly complex due to the high data dependencies of the mapped algorithm. These dependencies were resolved with the proposed architecture of the pipelined fetch sub-units. The architecture that combines DAER and PolyMem frameworks, Figure 10, consists of independent parallel fetch units, which serialize the data requests to the PolyMem module. In more details, the

Figure 11: RACOS Architecture

mapped architecture consists of parallel FIFOs, muxes, demuxes and control-based modules that convert the single-data memory access requests from the pipelined sub-fetch units into PolyMem parallel-data access requests. In addition, the processing step of the proposed architecture is parallelized using parallel processing units for the multiple data that arrive at each read request.

Our performance results suggest that augmenting DAER with PolyMem results in up to 6.0X memory accesses reduction, mainly due to parallel memory accesses. Also, the DAER and PolyMem combination can use internal parallel memory accesses with a cache-based scheme, which can lead to an even higher reduction of the total number of data fetches.

8 RACOS: RECONFIGURABLE ACCELERATOR OS

The EXTRA project is designed upon simple and intuitive interfaces between software and reconfigurable hardware. RACOS, our Reconfigurable ACcelerator OS[37], schedules reconfigurable hardware accelerators and performs I/O transparently to the user. It orchestrates and enables multiple applications to use and share accelerators, effectively virtualizing the reconfigurable resources. RACOS supports Multiple Partially Reconfigurable Regions (PRRs); each can host either single- or dual-threaded accelerators, and is responsible for scheduling the accelerators for execution.

8.1 Architecture

The main components of RACOS, as presented on Figure 11, involve the FPGA board, the Kernel Level Driver, and the user level library. Library calls are translated to system calls, modifying elements in the list of accelerated kernels. Hardware events trigger handler calls, served by the I/O scheduler. The reconfigurable hardware consists of a static region implementing the fixed functionality and streaming interfaces, and the reconfigurable regions.

The kernel-level software provides the interface between applications and PRRs in the FPGA. The accelerators submitted to RACOS are kept on the accelerator list until they are scheduled for execution. The scheduler, triggered by hardware events, performs a policy-based selection. We implement and evaluate four scheduling policies: (1) simple; (2) in order, which respects the order of requests; (3) out of order; and (4) forced, which aims to reduce the number of reconfigurations. A user level library is also implemented as a

Figure 12: Retinal Image Segmentation

straightforward component which translates library functions to system calls.

8.2 Evaluation

We evaluated two accelerated image processing applications, which we combine to produce three execution scenarios: (a) multiple instances of a single application share all the accelerators, (b) multiple instances of both applications share the common accelerators, and (c) with multiple threads that do not share any of the hardware accelerators. Our results show that (1) despite its generality, RACOS can achieve high throughput rates, close to the maximum reported in bibliography[38], and (much) better reconfiguration throughput reaching 177 partial reconfigurations per second for our platform and benchmark accelerators, and (2) that RACOS' flexibility comes at a very small resource cost (about 3% of a medium sized FPGA), comparable or better than the current state of the art[39].

9 RETINAL IMAGE SEGMENTATION: A CASE STUDY

In computer vision, image segmentation is the process of partitioning a digital image into multiple segments. The goal of segmentation is to simplify and/or change the representation of an image into something that is more meaningful and easier to analyze, either by a human observer or by an automated computer function.

One of the applications of interest for EXTRA - retinal vessel segmentation [40] - refers to the extraction of the vessel structure from the background in fundus images. The fundus corresponds to the interior surface of the eye and includes the retina, optic disc, macula, fovea, and posterior pole. Vessel segmentation enables the extraction of morphological attributes of retinal blood vessels, such as length, width and branching pattern. These assist the diagnosis, screening, treatment and evaluation of various cardiovascular and ophthalmologic diseases such as diabetes, hypertension, arteriosclerosis and choroidal neovascularization.

In such images, the problem of inaccurate segmentation comes mainly as a result of non-uniform illumination of the background. Moreover, the variable distance of different retinal areas from the camera causes further degradation because of the expected loss of focus in some areas. These effects cause color level variations within the same image and color level differences between different images, resulting in significant complications in detecting the blood vessel tree accurately.

9.1 Application Description

The Retinal Image Segmentation is illustrated in Figure 12. The application deployed in EXTRA project is based on the concept of

matched filters [41]. The input images used are either grayscale or 2D color retinal images (RGB). In both cases the actual input is similar, since for color images only the green channel is retained as it contains most of the information. The size of the images is not fixed and depends on the instruments that have been used to capture the images.

The main computational kernel of the application is comprised of two processes. The first one includes a number of denoising functions in the form of Gaussian filters aiming to reduce the effect of high frequency noise. The presence of such noise can lead to false detections in the forthcoming computations.

The main vessel detection function is a set of steerable matched filters. Since the cross-section of a vessel can be modeled as a Gaussian function, a series of Gaussian-shaped filters can be used to mark the vessels for detection. Steerable filters are used (in the current implementation, seven directions - 0, 30, 60, 90, 120, 150 and 180 degree angles - are considered) to separate extracted features with the strongest responses.

The segmentation process per 5MP image on a high-end workstation (Xeon E5-1650v3 @3.5GHz) requires approximately 105 sec to complete. However, the process of completing the segmentation for a single patient requires the processing of a series of images that depend on the required accuracy of results, as well as on the quality of the image capture equipment (less precise tools require the capture of more images for better results). Typically, several retinal images captured from different angles are used, while it is also common to inject the patient with indo-cyanine green dye to produce reversed images as well. As a result, the overall number of images per patient increases to tens or hundreds. Furthermore, this application is indented to be used in central processing hubs which process retinal images of multiple patients, a scenario that further increases the overall computational task severely.

It should also be noticed, that advances in image acquisition sensors will eventually require the processing of larger images. In such cases, the computational costs quadruples with the doubling in image size. On top of that, higher resolution images require the use of more filters in the processing pipeline and therefore the computational costs increases even further.

9.2 Implementation with CAOS

The CAOS framework can transform a pure C application version into a working hardware-accelerated solution for the Xilinx Zynq (Zedboard), Zynq UltraScale (ZCU102 development board) and the Amazon F1 instance (using Xilinx Virtex UltraScale+ FPGA and a Xeon host processor). Compared to a completely manual process, CAOS consolidates all design steps required to perform such a task (profiling, testing, design space exploration, hardware/software partitioning, and implementation), producing a solution with practically equivalent results, both in terms of performance and hardware resources used. Some performance tuning may be required because CAOS operates with certain default values (e.g., clock frequency), but this is trivial to perform since the final outcome is a complete Vivado project that the designer may tune in order to change such attributes.

The default solution produced by CAOS outperforms the original C-version by more than 4 times when using the F1 instance. The design was clocked at 200MHz, and the accelerated kernel is 5.5X

faster than the software implementation. The overall application is 4.2X faster, executing in 25s compared to the original 105s. Manual optimizations could provide additional benefits, however it should be stressed that these optimizations would require a significant code rework in order to adopt Xilinx-specific libraries and design methodologies that would break portability to other devices and platforms, requiring at the same time significant effort and multiplying the test and verification time at every possible deployment.

These experiments demonstrate the biggest advantage of CAOS: for application developers with no significant expertise in hardware design, it is possible to produce working hardware accelerated solutions that can be readily deployed (especially for the F1 instance, since the framework produces everything that is required). On the other hand, for experienced hardware developers, CAOS offers an easy way to produce multiple iterations of the same application, testing different optimizations and targeting multiple implementation platforms, which increases productivity.

10 CONCLUSIONS

Reconfigurable computing is a very promising technology in the race towards exascale computing, due to its high performance and energy efficiency. However, more research needs to be invested in improving the support for application development on heterogeneous, reconfigurable HPC platforms. The EXTRA Open Research platform, presented in this paper, is an ecosystem of tools and technologies that provides such support.

This paper provided a detailed analysis of four EXTRA core technologies: (1) We described the principles of our CAOS toolchain, highlighting that it was designed as a tool open for contributions from the community. CAOS targets the full stack of the application optimization process. (2) We presented our parallel memory, PolyMem, addressing the need for high bandwidth in data-intensive applications. Our results indicate that PolyMem, acting as a parallel caching mechanism, delivers high-bandwidth and a high degree of flexibility for application designers. Furthermore, we discuss the PolyMem-CAOS integration. (3) Our DAER framework targets both stream-based applications and workloads with arbitrary memory accesses, aiming at mitigating the problem of efficiently mapping them to reconfigurable hardware. PolyMem has also been integrated with the DAER framework. (4) Finally, we introduced RACOS, our Reconfigurable ACcelerator OS which enables transparent access and virtualization of reconfigurable hardware accelerators.

As a case study for the capabilities of the EXTRA platform, we presented the implementation of a retinal image segmentation application. Specifically, we demonstrated that CAOS can lower the effort barrier to use reconfigurable hardware accelerators.

Overall, this paper shows evidence that the EXTRA platform is a feasible solution for enabling developers of HPC applications access to heterogeneous reconfigurable computing, without high penalties in terms of performance or energy efficiency. Therefore, we believe the EXTRA Open Research Platform is a step forward towards using reconfigurable hardware in exascale computing.

ACKNOWLEDGMENTS

This project has received funding from the EU Horizon 2020 research and innovation programme under grant No 671653.

REFERENCES

- [1] "Apple iPhone 7 Teardown," techinsights.com/about-techinsights/overview/blog/apple-iphone-7-teardown/.
- [2] "NVIDIA G-Sync HDR Module Adds \$500 to Monitor Pricing," techpowerup.com/245463/nvidia-g-sync-hdr-module-adds-usd-500-to-monitor-pricing/.
- [3] "Super Nintendo Games Shall Live Eternal On The Analogue Super Nt," forbes.com/sites/curtissilver/2017/10/16/analogue-super-nt-snes-classic/.
- [4] "Project Catapult," microsoft.com/en-us/research/project/project-catapult/.
- [5] "Amazon EC2 F1 Instances," aws.amazon.com/ec2/instance-types/f1/.
- [6] "The EXTRA Project." [Online]. Available: www.extrahpc.eu/
- [7] C. B. Ciobanu et al., "EXTRA: Towards an Efficient Open Platform for Reconfigurable High Performance Computing," in *CSE 2015*, Oct 2015, pp. 339–342.
- [8] D. Stroobandt et al., "EXTRA: Towards the Exploitation of eXascale Technology for Reconfigurable Architectures," in *ReCoSoC 2016*, 2016, pp. 1–7.
- [9] —, "An Open Reconfigurable Research Platform as Stepping Stone to Exascale High-Performance Computing," in *DATE 2017*, March 2017, pp. 416–421.
- [10] J. Jeddellouh and B. Keeth, "Hybrid memory cube new DRAM architecture increases density and performance," in *VLSIT 2012*, 2012, pp. 87–88.
- [11] H. Jun et al., "HBM (High Bandwidth Memory) DRAM Technology and Architecture," in *2017 International Memory Workshop (IMW)*, 2017, pp. 1–4.
- [12] S. Yin et al., "Multibank memory optimization for parallel data access in multiple data arrays," in *Proceedings of ICCAD*. IEEE, 2016, pp. 1–8.
- [13] "EuroHPC: European HPC Initiative." [Online]. Available: eurohpc.eu/
- [14] I. Mavroidis et al., "Ecoscale: Reconfigurable computing and runtime system for future exascale systems," in *DATE 2016*, 2016, pp. 696–701.
- [15] M. Katevenis et al., "The ExaNeSt Project: Interconnects, Storage, and Packaging for Exascale Systems," in *DSD 2016*, 2016, pp. 60–67.
- [16] A. Rigo et al., "Paving the way towards a highly energy-efficient and highly integrated compute node for the Exascale revolution: the ExaNoDe approach," in *DSD 2017*, 2017, pp. 486–493.
- [17] R. Iakymchuk et al., "A particle-in-cell method for automatic load-balancing with the allscale environment," in *EASC 2016*, 2016.
- [18] "The EXA2PRO project." [Online]. Available: exa2pro.eu/
- [19] "The LEGaTO project." [Online]. Available: legato-project.eu/
- [20] M. Rabozzi et al., "A CAD Open Platform for High Performance Reconfigurable Systems in the EXTRA Project," in *ISVLSI 2017*, 2017, pp. 368–373.
- [21] R. Cattaneo et al., "On how to accelerate iterative stencil loops: A scalable streaming-based approach," *ACM TACO*, vol. 12, no. 4, p. 53, 2016.
- [22] M. Rabozzi et al., "Optimizing streaming stencil time-step designs via FPGA floorplanning," in *FPL 2017*, 2017, pp. 1–4.
- [23] L. Di Tucci et al., "The Role of CAD Frameworks in Heterogeneous FPGA-Based Cloud Systems," in *ICCD 2017*, 2017, pp. 423–426.
- [24] F. Peverelli et al., "OXiGen: A tool for automatic acceleration of C functions into dataflow FPGA-based kernels," in *IPDPSW 2018*, 2018.
- [25] "Docker," <https://www.docker.com>.
- [26] C. B. Ciobanu et al., "MAX-PolyMem: High-Bandwidth Polymorphic Parallel Memories for DFEs," in *IPDPSW RAW 2018*, 2018, pp. 1–8 (to appear).
- [27] A. Ramirez et al., "The SARC Architecture," *IEEE Micro*, vol. 30, no. 5, pp. 16–29, 2010.
- [28] C. Ciobanu, "Customizable Register Files for Multidimensional SIMD Architectures," Ph.D. dissertation, Delft University of Technology, The Netherlands, 2013.
- [29] C. B. Ciobanu et al., "The Case for Polymorphic Registers in Dataflow Computing," *IJPP*, May 2017. [Online]. Available: <https://doi.org/10.1007/s10766-017-0494-1>
- [30] C. Ciobanu et al., "A Polymorphic Register File for Matrix Operations," in *Proceedings of SAMOS 2010*, 2010, pp. 241–249.
- [31] G. Stramondo et al., "From Application Access Traces to Custom Parallel Memories," in *Proceedings of CompSys 2017*, 2017, pp. 1–7.
- [32] —, "Towards Application-Centric Parallel Memories," in *Euro-Par PPW HeteroPar 2018*, 2018, pp. 1–12 (to appear).
- [33] "MAX-PolyMem Source Code," github.com/giuliostramondo/RAW2018.
- [34] "The STREAM benchmark website." <https://www.cs.virginia.edu/stream/>
- [35] J. E. Smith, "Decoupled access/execute computer architectures," in *ISCA '82*, 1982, pp. 112–119.
- [36] "Hybrid memory cube specification 2.1," hybridmemorycube.org/files/SiteDownloads/HMC-30G-VSR_HMCC_Specification_Rev2.1_20151105.pdf.
- [37] C. Vatsolakis and D. Pnevmatikatos, "RACOS: Transparent access and virtualization of reconfigurable hardware accelerators," in *SAMOS 2017*, 2017, pp. 11–19.
- [38] M. Jacobsen and R. Kastner, "Riffa 2.0: A reusable integration framework for fpga accelerators," in *FPL 2013*, Sept 2013, pp. 1–8.
- [39] D. V. Vu et al., "Enabling partial reconfiguration for coprocessors in mixed criticality multicore systems using PCI express single-root I/O virtualization," in *ReConFig 2014*, 2014, pp. 1–6.
- [40] P. Echevarria et al., "Blood Vessel Segmentation in Retinal Images," *Archives of Ophthalmology*, vol. 122, no. 4, pp. 552–563, 2004.
- [41] S. Chaudhuri et al., "Detection of Blood Vessels in Retinal Images using Two-dimensional Matched Filters," *IEEE Transactions on Medical Imaging*, vol. 8, no. 3, pp. 263–269, 1989.