Reconfigurable Buffer Structures for Coarse-Grained Reconfigurable Arrays

Éricles Sousa, Frank Hannig, Jürgen Teich

▶ **To cite this version:**

Éricles Sousa, Frank Hannig, Jürgen Teich. Reconfigurable Buffer Structures for Coarse-Grained Reconfigurable Arrays. 5th International Embedded Systems Symposium (IESS), Nov 2015, Foz do Iguaçu, Brazil. pp.218-229, 10.1007/978-3-319-90023-0_18 . hal-01854155

HAL Id: hal-01854155
https://inria.hal.science/hal-01854155

Submitted on 6 Aug 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

Reconfigurable Buffer Structures for Coarse-Grained Reconfigurable Arrays

Éricles Sousa, Frank Hannig, and Jürgen Teich

Hardware/Software Co-design
Department of Computer Science
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)
{sousa, hannig, teich}@cs.fau.de

Abstract. Coarse-Grained Reconfigurable Arrays (CGRAs) have emerged as a powerful solution to speedup computationally intensive applications. Heterogeneous MPSoC architectures containing such reconfigurable accelerators have the advantage of providing high flexibility, power-efficiency, and high performance. However, CGRAs may suffer from a data access bottleneck. To mitigate this problem, we present a reconfigurable buffer architecture for CGRAs. Here, the buffers can be configured at runtime to select between different schemes for memory access, i. e., addressable RAMs or pixel buffers. We showcase the benefits of our approach by prototyping a heterogeneous MPSoC architecture containing a RISC processor and a class of CGRA called Tightly Coupled Processor Arrays (TCPAs). The architecture is prototyped in FPGA technology. For basic image processing algorithms, we demonstrate that our proposed buffer structures for system integration allow to increase the memory bandwidth utilization and allow for a performance improvement of up to 7% in comparison to state-of-the-art solutions for image processing.

1 Introduction

Semiconductor technology has already hit the *power wall* and is not far away from hitting the *utilization wall* [1]. These effects are caused by shrinking technology, which continuously leads to higher energy densities. However, chips can only handle a limited power budget. As a consequence, the potentially available chip area might not be fully utilized or at least not simultaneously. Thus, these days, energy efficiency has become more important than pure computing power. This means, that in order to scale computing performance in the future, systems' energy efficiency has to be significantly improved. The design of embedded systems containing heterogeneous hardware and customized resources, such as accelerators dedicated for one application domain, is a promising solution to address this challenge.

In this realm, CGRAs are appealing by providing programmability with the potential for high computational throughput and at the same time high energy efficiency [2]. There are many possible ways of integrating such reconfigurable accelerators into System-on-Chip (SoC) designs. For instance, they can be tightly coupled with a processor and the communication can be realized by a specialized interface or via a shared register file. An alternative is to share the last level cache of a CPU. In this case, a dedicated controller for

the shared cache (e. g., shared L2 cache) is needed for connecting CPU and accelerator. However, this approach requires cache coherency models and protocols that need to be adapted according to the targeted application. Apart from these system integration options, it is also possible to connect a hardware accelerator to a shared bus or NoC. Here, the communication with the rest of the system could be realized by message passing, for example, using DMA transfers to/from a local accelerator memory. Although this solution scales very well, the overhead for accessing shared resources may compromise the performance of such accelerators. As a solution for this challenge, we propose to use a very flexible buffer structure that can be configured at runtime either as addressable RAM or pixel buffer as first presented in [3].

In this paper, we propose to use such buffers for coupling a RISC processor directly to the border processing elements of a class of CGRAs called TCPAs (tightly coupled processor arrays [3]). We are using an edge detection algorithm as a case study to demonstrate how the image processing throughput can thereby be increased up to the memory bandwidth available in the system. In the remainder of this paper, we first compare our solution with the state-of-the-art solutions in literature. Then, the target class of TCPA accelerators is presented in Section 3. Section 4 describes in detail our proposed system integration of a TCPA to a RISC processor and its implementation as an FPGA prototype. Experimental results on memory bandwith and performance improvements are provided in Section 5. Finally, we conclude our work in Section 6.

2 Related Work

Often, there is only a fine line between on-chip processor arrays and coarse-grained reconfigurable architectures (CGRA) since the provided functionality is similar. CGRA examples include architectures such as PACT XPP [4] and ADRES [5], both of which are arrays that can switch between multiple contexts by runtime reconfiguration. Whereas ADRES is a CGRA that is tightly coupled with a VLIW processor, PACT's XPP architecture provides a column of RAMs at two borders of the array, which can be configured to two different modes: addressable or streaming mode. However, except one simple counter per buffer, the architecture does not provide any sophisticated address generators. Thus, if complex buffer addressing schemes are required, the PEs of the array have to be involved in the task of address computations. In [6], the authors propose a generic VHDL template based on a full buffering approach which allows a fast and efficient parallel and pipelined processing of 2-D stencil code applications. This approach is limited to regular window-based applications and the stencil mask has to move always in the same scanning order. In order to cover different applications, Liang et al. [7] describe different kinds of buffering schemes such as full buffering, partial buffering, packing, and buffering with packing. However, at runtime, there is no possibility to switch between these configurations and the selection of the buffer operation has to be defined as a parameter at synthesis time. In the area of high-level synthesis, there exist two tools, ROCCC [8], which provide so-called *smart buffers*, and PARO [9], which allows to generate dedicated pixel buffers automatically. Unlike all aforementioned research works, the reconfigurable buffer structure introduced in [3] is able to adapt according to different application requirements.

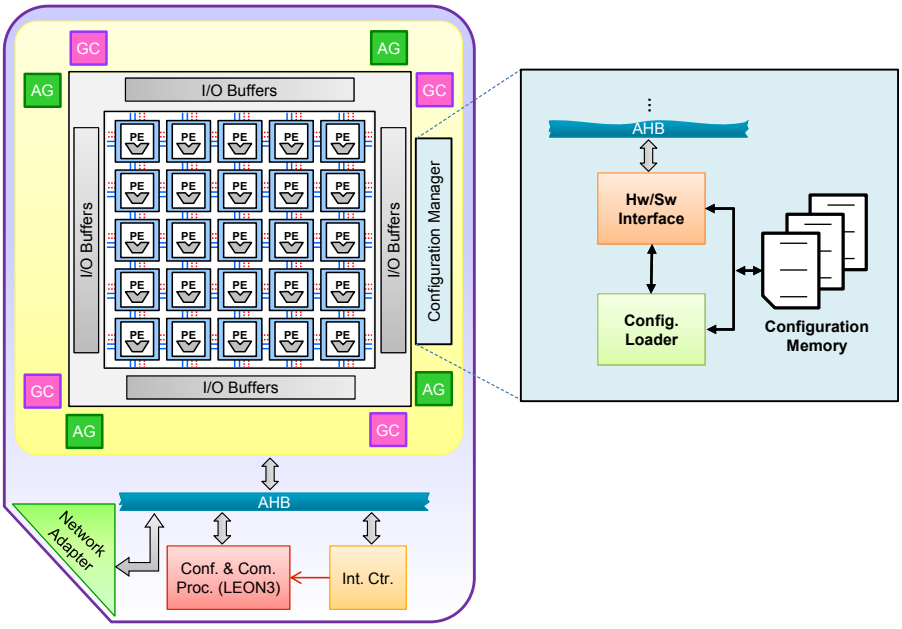

Fig. 1. An abstract architectural view of a TCPA is shown on the left. The abbreviations AG and GC stand for address generator and global controller, respectively. On the right side is the configuration manager (CM) shown that provides the interface to reconfigure the entire TCPA architecture.

3 Accelerator Architecture

A TCPA [10] as shown in Fig. 1 denotes a class of CGRAs being highly parameterizable. The heart of this accelerator consists of a massively parallel array of tightly coupled Very Long Instruction Word (VLIW) Processing Elements (PEs) complemented by peripheral components such as I/O buffers as well as several control, configuration, and communication companions. Some parameters, such as number of PEs, interconnect topology, number of functional units as well as the register organization within the PEs, are defined at synthesis time, whereas other parameters such as programmable delays between neighbor processors and inter-PE interconnect can be reconfigured at runtime. Each PE at the boundary can read/write data directly from/to a local buffer (denoted I/O buffer in Fig. 1) connected to it and each PE can exchange data with its neighbor PE in a single clock cycle. A TCPA can exploit a parallel and direct PE-to-PE communication, as long as input data is available as well as space is available for accepting processed output data at the surrounding I/O buffers of the array. Through the VLIW nature of each PE and the parallel and synchronous execution of mainly loop nest iterations, a TCPA nicely exploits both instruction- and loop-level parallelism while achieving a much higher energy efficiency compared to general purpose Commercial Off-The-Shelf (COTS) embedded processors [11]. TCPAs can be integrated into SoC designs, e. g., using a bus-based interconnect architecture, shared registers, or a shared data cache. Thus, they can be used as accelerators in different platforms in order to speedup computationally intensive applications. The building blocks of a TCPA are briefly described in the following.

Processor Array: Before synthesis, the rows and columns defining the total number of PEs of an array need to be specified. The array may be even configured to have regions of heterogeneous PEs. For instance, some of the processors at the borders might include extra functionality for the purpose of address generation. However, in the rest of the paper, we consider only homogeneous arrays.

Array Interconnect: The PEs in the array are interconnected by a circuit-switched mesh-like interconnect, which allows data produced in one PE to be used already in the next cycle by a neighboring PE. An interconnect wrapper encapsulates each PE and is used to describe and parameterize the inter-PE network topology. The wrappers are arranged in a grid fashion and may be customized at compile time to have multiple input/output ports in the four directions, i. e., north, east, south, and west. Using these wrappers, indeed different topologies like a 2-D mesh, but also other topologies such as torus or 4-D hypercube can be implemented and changed dynamically. Thus, the array interconnect can be reconfigured to support different applications. To define all possible interconnect topologies, an adjacency matrix is defined for each interconnect wrapper in the array at compile time. Each matrix explains how the input ports of its corresponding wrapper and the output ports of the encapsulated PE are connected to the wrapper output ports and the PE input ports, respectively. If multiple source ports are allowed to drive a single destination port, then a multiplexer with an appropriate number of input signals is generated. The select signals for such generated multiplexers are stored in configuration registers and can therefore be changed at runtime [12]. Two different networks, one for data and one for control signals, can be defined by their data width and number of dedicated channels in each direction. For instance, two 16-bit channels and one 1-bit channel might be chosen as data and control network, respectively.

Processor Element: A PE itself is again a highly parameterizable component with a VLIW (very long instruction word) structure. Different types and numbers of functional units (e. g., adders, multipliers, shifters, logical operations) can be instantiated as separate functional units, which can work in parallel. We call the processing elements *weakly-programmable* [12] since the functional units have only a reduced, domain-specific instruction set, which is tailored for a specific field of applications. Additionally, the control path is kept very simple (no interrupt handling, multi-threading, instruction caching, etc.).

Buffers/Address Generators: As the PEs are tightly coupled, they do not have direct access to a global memory. Data transfers to and from the array must be performed through the border PEs. Instead of using FIFOs, the border PEs are connected to highly adaptable surrounding I/O buffers that are explained in more detail in Section 4.

Global Controller: Due to the regularity of the considered loop programs, and since most of the static control flow information is needed in all PEs that are involved in the parallel computation of a given loop nest, we can move as much as possible of the *common* control flow out of the PEs to a global controller (GC) per application. The GC generates branch control signals, which are propagated in a delayed fashion over the control network to the PEs where they are combined with the local control flow (program execution). Moreover, this orchestration enables the execution of nested loop programs at *zero-overhead loop*.

Configuration and Communication Processor: The admission of an application on the processor array, as well as the communication with a network via a network adapter (NA), and TCPA programming is managed by a companion RISC processor (LEON3 in Fig. 1) that is named configuration & communication processor (CCP). In consequence, the companion handles resource requests and initiates appropriate DMA transfers via the NA to fill and flush the I/O buffers around the array.

Configuration Manager: The Configuration Manager (CM) holds configuration streams for the TCPA. This includes both the assembly codes to be loaded into the PEs as well as interconnect reconfiguration. Since TCPAs are coarse-grained reconfigurable architectures, the size of their configuration streams normally amounts to a few hundred bytes, which enables ultra fast context switches in the system. The configuration loader transfers a configuration stream to the PEs via a configuration bus. It is possible to group a set of PEs in a rectangular region to be configured simultaneously if they receive the same configuration, thereby reducing significantly the configuration time. As also depicted in Fig. 1, the CM is mainly composed of three parts, a hardware/software interface, configuration loader, and configuration memory. The interface decodes the commands sent from a CCP, which can read or write into a configuration memory that stores the interconnection configuration as well as the binary code for all PEs. Once the configuration memory is populated, the configuration loader starts to configure the interconnection topology between the PEs. Afterwards, each PE is loaded with its assembly (binary) code, and finally, the CM issues a reset to trigger the start of parallel computation on the configured array.

4 Reconfigurable Buffer Structures

TCPAs are envisioned to be used as programmable accelerators in MPSoCs and are very suited for domain-specific computing from the areas of signal, image, and video processing, as well as other streaming processing applications. Based on the inherent algorithmic nature of an application and the chosen parallelization strategy (e. g., pipelining, loop partitioning), different I/O and buffering approaches might be appropriate. For example, consider an one-dimensional digital signal processing application for a continuous audio signal where input data (audio samples) are streamed into a filter, are processed and after some initial latency filtered data are streamed out. For such 1-D applications, streaming buffers (e. g., a FIFO) at the input and the output would be ideally suited in order to decouple the filtering from the rest of the system. Especially in case of systems that are comprised of buses or NoCs, which do not offer any guaranteed service, asynchronous streaming buffers are vital in order to increase performance and quality.

For two-dimensional image processing (e. g., edge detection, Gaussian filtering) or linear algebra algorithms (matrix-matrix multiplication, LU decomposition, etc.), the requirements are quite different. In this case, the data often already resides somewhere in the system—e. g., in the main memory—and has to be transported to the accelerator before it can be computed. If large problem instances have to be computed, partitioning techniques [13,14] are used to break down the data into several smaller chunks, which have to be transported and processed one after the other in the accelerator. Data locality is a key concept for efficient execution (performance, energy consumption) in such cases.

Thus, the amount of reads and writes to the main memory has to be minimized as much as possible, and redundant data copies should be avoided in order to increase energy efficiency. For instance, when blocking is applied to map stencil computations to multiple processors that can process the input data independently in parallel, border problems may occur, i. e., input data on the border area is needed in two partitions. The size of this overlap region varies according to the window size of the local operator. For a window with $w \times w$ pixels, the total of data that overlaps into neighboring regions is equal to the kernel radius, $r = \left\lfloor \frac{w}{2} \right\rfloor$. But, because the pixels are shared in two directions, the overlap area is twice the radius r of the window, i. e., $2r$. Thus, when an input image of size $W \times H$ is partitioned in M horizontal tiles, the total of data shared between all partitions is given by Eq. (1)

$$T_{overlap} = 2r \cdot W \cdot (M-1) \tag{1}$$

In the case where only N vertical tiles are computed in parallel, Eq. (1) can be rewritten by replacing the terms W and M by H and N, respectively.

Moreover, the additional overhead for transferring all the border elements from the local memory to the input buffers is defined by Eq. (2)

$$Overhead = T_{overlap} \cdot L, \tag{2}$$

where L is the latency to copy these data from the local memory to the input buffers. To avoid this additional overhead, a hardware mechanism is desirable that does not affect the performance and does not require any data copies.

In order to fulfill the aforementioned demands, we propose a highly adaptable architecture, which can be configured to either work as addressable memory banks (RAM), provide data in a streaming manner, or function as buffers customized for stencil operations. Figure 2(a) presents an overview of the proposed I/O buffer architecture, which uses dual-ported RAMs (DPRAMs) as interface for data transfer and clock domain transition between the local bus (AHB[1]) on the top and the processor array on the bottom. To reduce the amount of connectors to the AHB, several DPRAMs may be wrapped as a single buffer, where the individual RAMs are associated to the most significant bits of the target address, presented by the AHB. The connection between the DPRAMs and the processor array can be established in several ways, as each of the RAM components has a discrete data channel to the TCPA. To increase the storage capacity, the address space of DPRAMs can be combined, as it is shown for combinations of two and four DPRAMS in (b) and (c), respectively. Although Fig. 2(a) shows the read and the write direction between the buffer and the TCPA, the two subsequent figures (b) and (c) omit the write direction from the array to the buffer for better visibility. Data reads and writes between the DPRAMs and the TCPA are generated by a single address generator (AG) for the buffer, which can be configured to follow arbitrary addressing schemes, for instance, to facilitate dense or sparse stencil operations. Depending on the configuration, the most significant bits of the address, generated by the AG are used to select between concatenated memories. Data partitioning onto neighboring processing elements and loop-carried data dependencies between iterations may introduce offsets

[1] *Advanced High-performance Bus* (AHB) is a bus protocol introduced by ARM Ltd. as part of the *Advanced Microcontroller Bus Architecture* (AMBA) for SoC designs.

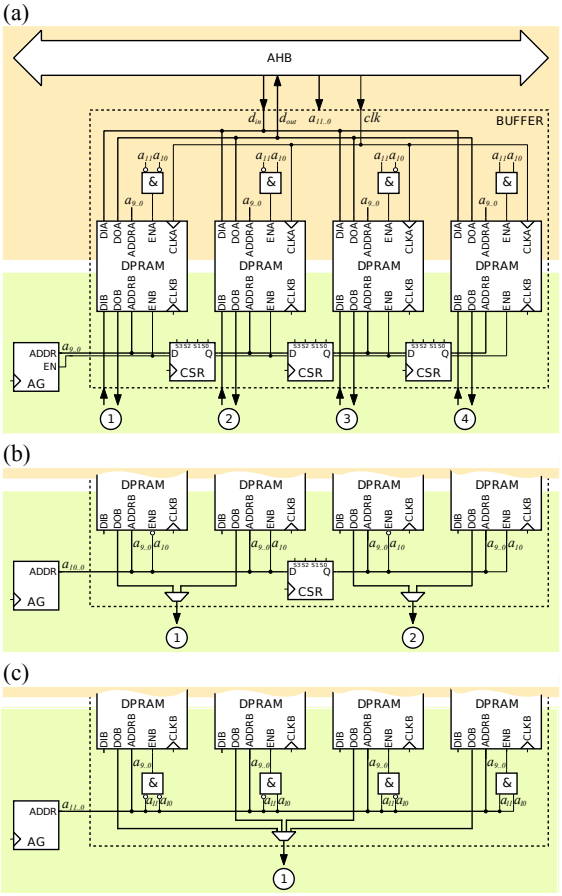

Fig. 2. In (a), (b), and (c), a reconfigurable I/O buffer architecture is shown which may be configured into different modes and trade-off the buffer sizes with the number of available independent ports to the processor array [3].

between computations [15], however, the addressing scheme for data access remains the same, thus it is sufficient to delay the values generated by the AG through the use of configurable shift registers (CSRs). Instead of enabling every part of the CSR individually, the amount of necessary control logic can be greatly reduced by following a logarithmic scheme, which also reduces the energy consumption.

In addition to the so far introduced capabilities, it is very often necessary to propagate entire image lines between neighboring processing elements for computation. To reduce memory transfers into the buffers over the local bus, also a chained buffer is supported, referred to as *pixel buffer*. The key idea is to initially fill the buffers from the local bus and to pass data from one buffer to the next as it is read out. Operating the buffers in this way only requires a single port to maintain data streaming to the buffer. However, the contents can be fed to the array via individual ports in parallel. For instance, three

Table 1. Resource utilization of our reconfigurable buffer structure on a Kintex-7 FPGA.

Component	Slice Register	LUT	DPRAM
Reconfigurable Buffer	825	1,658	5

interlinked buffers provide also three output ports to the TCPA, whereas new data will only be written to the first buffer in the sequence. Since the two ports of the DPRAM also provide clock-transition between the bus and the TCPA, the bus-side port of the memory must provide a means to switch between the bus and TCPA clock domain. Despite of the individual representations shown above, the buffer architecture is a combined implementation, which allows to select one of the introduced modes.

5 Experimental Results

In order to demonstrate the concepts and benefits of our proposed I/O buffer architecture, the TCPA shown in Fig. 1 has been synthesized on a Kintex-7 FPGA prototyping platform. The architecture consists of a RISC processor and a TCPA, both connected through an AMBA bus. An SRAM of 256 MB is also available and used to store the input image frames arriving over the network adapter (NA) as shown in Fig. 1. The RISC processor is a LEON3, which is a synthesizable VHDL model of a 32-bit processor compliant with a SPARC V8 architecture and can run up to 120 MHz. This processor is highly configurable at synthesis time and is particularly suitable for SoC designs. In our prototype, the TCPA is composed of a 5×5 array of VLIW processing elements operating at 60 MHz. The considered target application is a 5×5 Laplace operator used to detect the vertical and horizontal edges in an image. This algorithm is widely used as a pre-processing step in many image processing and computer vision applications. In the prototyped architecture, only five read ports are necessary to deliver data to the TCPA. Each PE computes one convolution coefficient and the last PE in the last row also performs the addition of all convolved values and outputs the final result of the computation.

We assume two DMA engines to read and write data from/to buffers. Using the addressable RAM, the DMA has to provide data to all individual channels of the input buffer. Instead, by using the pixel buffer, only the first channel receives data and automatically delivers the pixel values to adjacent channels.

To achieve similar granularity and access pattern as succeeded in the pixel buffer, the addressable RAM would require a double buffer scheme to hide the additional access for copying data of the consecutive image lines that are swept by the scanning window. This option is not considered, because the implementation of a double buffer scheme would demand more hardware resources. Furthermore, an additional controller would be required to transfer data from the main memory. Therefore, we use a blocking operation for the addressable RAM. Hence, the input image is divided into 5 horizontal tiles and each tile is individually processed by a group of 5 PEs. On the other hand, using the pixel buffer the input data is a steady stream. In both buffer schemes, the number of hardware resources is the same. Table 1 presents the resource utilization of one reconfigurable buffer structure that has 5 I/O channels. Since the accelerator is connected to a shared bus, it is possible to minimize the communication latency by transferring the input data simultaneously along with computation in an overlapping fashion. Thus, the input

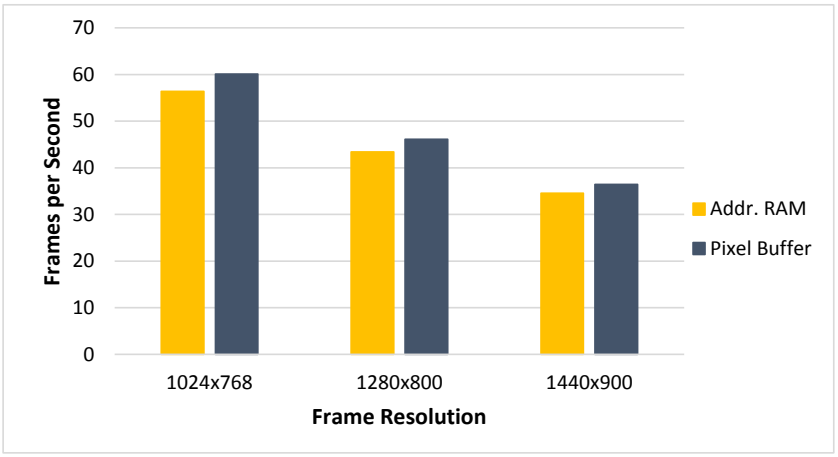

Fig. 3. Average performance of a 5×5 Laplace operator running on an FPGA-based TCPA implementation at 60 MHz and using different buffer schemes.

buffer can constantly deliver data. The output results are displayed by using a dedicated connection to a DVI interface. Hence, the AMBA bus is not involved in this operation and these two processes do not compete with each other. However, if there is any other application using the bus at the same time as the DMA attempts to read data from the local memory, it would result in a bus contention and the system performance would be affected.

Before starting the computation on the TCPA, it is necessary to define the mode of operation of all input and output channels of the reconfigurable buffer. For that, we use the LEON3 to load a configuration data into the buffer structure. Unlike [6,7], our proposed work has the possibility to change the mode of operation at runtime. The reconfiguration overhead is equal to 840 cycles, i. e., 7 µs, since the bus operates at 120 MHz. For measuring the system performance, we consider three different frames sizes, i. e., 1024×768, 1280×800, and 1440×900. For evaluating the performance, we first configure the input buffers as addressable RAM and observe the average throughput obtained from the TCPA. In the second experiment, we configure the input buffers to work as a pixel buffer. In both cases, the channel size is equal to the width of the input image. Thus, it is possible to perform the kernel computation of entire lines without fetching new data from the global memory. Fig. 3 presents the average performance in frames per second by taking into account the reconfiguration overhead for switching between the different buffer schemes. There, we observe that although the addressable RAM can be very customized for irregular memory access, it does not propagate the input data to adjacent input PEs, which share data in their borders.

By applying Eq. (1), we conclude that these additional data amount to 16, 20, and 22.5 KB for the three different frame sizes, i. e., 1024×768, 1280×800, and 1440×900, respectively. The latency for transferring data from the SRAM to input buffers corresponds to 8 cycles per transfer and using Eq. (2), we observe that the performance loss depicted in Fig. 3 corresponds exactly to the time for copying the pixels located at the border of the image. However, by using the pixel buffer, it is possible to achieve higher performance.

This is because it is designed for increasing data locality by means of propagating the input data to different channels. Thus, we can avoid to provide redundant data copies to the input buffers and consequently may increase the memory bandwidth utilization, in this example up to 7%.

6 Conclusion

This paper presents a reconfigurable buffer structure for coarse-grained reconfigurable arrays. In addition to traditional address-based memory banks, the buffer architecture can deliver data in a streaming manner to the processing elements of the array. Moreover, to minimize data transfers to the buffers, the design contains an interlinked mode which is especially targeted at 2-D kernel computations. For demonstrating the advantages of our reconfigurable I/O buffer structure, we synthesized a heterogeneous architecture consisting of a RISC processor and a tightly coupled processor array (TCPA). The processor is used for starting DMA transfers between a SRAM memory and the TCPA composed of a 5×5 array of VLIW processing elements. The target application chosen for performance evaluation of different I/O buffer modes is an edge detection algorithm that is widely used in computer vision and embedded applications. In the case of such stream-based applications, the pixel buffer mode outperforms the addressable RAM mode. As image lines will be needed in subsequent steps of the image kernel computation, the feedback concept reduces the amount of required memory transfers to the buffers to a minimum by propagating the image data from one memory to the next. Therefore, by means of selecting the right buffer configuration, a considerably higher performance may be achieved. In the case of an input frame resolution of 1024×768, the performance could be increased by 7%. Due to the higher data locality, the TCPA was able to compute 60 frames per seconds, while the reconfiguration overhead was only 7 µs. By performing such an ultra-fast reconfiguration, the overhead for switching between the different buffer schemes can be neglected.

However, the addressable RAM mode of operation can be more efficient in the case of partial buffering or non-raster scanning. As a future work, we intend to analyse not only the power efficiency of our approach, but also the system performance by considering different window sizes, partitioning schemes as well as scenarios of concurrent applications competing for shared resources such as in invasive computing [16].

Acknowledgment

This work was supported by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Centre “Invasive Computing” (SFB/TR 89) and Research Training Group 1773 “Heterogeneous Image Systems”. The first author is also grateful to the Brazilian National Council for Scientific and Technological Development (CNPq) for supporting his research.

References

1. N. Goulding-Hotta, J. Sampson, G. Venkatesh, S. Garcia, J. Auricchio, P. Huang, M. Arora, S. Nath, V. Bhatt, J. Babb, S. Swanson, and M. Taylor, “The GreenDroid mobile application

processor: An architecture for silicon's dark future," *IEEE Micro*, vol. 31, no. 2, pp. 86–95, March–April 2011.

2. P. Yongjun, J. Park, and S. Mahlke, "Efficient performance scaling of future CGRAs for mobile applications," in *International Conference on Field-Programmable Technology (FPT)*, Dec 2012, pp. 335–342.
3. F. Hannig, M. Schmid, V. Lari, S. Boppu, and J. Teich, "System integration of tightly-coupled processor arrays using reconfigurable buffer structures," in *Proceedings of the ACM International Conference on Computing Frontiers (CF)*. ACM, 2013.
4. V. Baumgarte, G. Ehlers, F. May, A. Nückel, M. Vorbach, and M. Weinhardt, "PACT XPP – a self-reconfigurable data processing architecture," *The Journal of Supercomputing*, vol. 26, no. 2, pp. 167–184, 2003.
5. F. Bouwens, M. Berekovic, B. De Sutter, and G. Gaydadjiev, "Architecture enhancements for the ADRES coarse-grained reconfigurable array," in *Proceedings of the International Conference on High Performance Embedded Architectures and Compilers*, 2008, pp. 66–81.
6. M. Schmidt, M. Reichenbach, and D. Fey, "A generic VHDL template for 2D stencil code applications on FPGAs," in *Proceedings of the International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops*, 2012, pp. 180–187.
7. X. Liang, J. Jean, and K. Tomko, "Data buffering and allocation in mapping generalized template matching on reconfigurable systems," *Journal of Supercomputing*, vol. 19, no. 1, pp. 77–91, 2001.
8. Z. Guo, B. Buyukkurt, and W. Najjar, "Input data reuse in compiling window operations onto reconfigurable hardware," in *Proceedings of the 2004 ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES)*. ACM, 2004, pp. 249–256.
9. F. Hannig, H. Ruckdeschel, H. Dutta, and J. Teich, "PARO: Synthesis of hardware accelerators for multi-dimensional dataflow-intensive applications," in *Proceedings of the Fourth International Workshop on Applied Reconfigurable Computing (ARC)*, ser. Lecture Notes in Computer Science (LNCS), vol. 4943. Springer, 2008, pp. 287–293.
10. F. Hannig, V. Lari, S. Boppu, A. Tanase, and O. Reiche, "Invasive tightly-coupled processor arrays: A domain-specific architecture/compiler co-design approach," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 13, no. 4s, pp. 133:1–133:29, 2014.
11. D. Kissler, A. Strawetz, F. Hannig, and J. Teich, "Power-efficient reconfiguration control in coarse-grained dynamically reconfigurable architectures," *Journal of Low Power Electronics*, vol. 5, no. 1, pp. 96–105, 2009.
12. D. Kissler, F. Hannig, A. Kupriyanov, and J. Teich, "A highly parameterizable parallel processor array architecture," in *Proceedings of the IEEE International Conference on Field Programmable Technology (FPT)*. IEEE, 2006, pp. 105–112.
13. J. Teich, *A compiler for application specific processor arrays*, ser. Reihe Elektrotechnik. Shaker, 1993. [Online]. Available: https://books.google.de/books?id=WqOGAgAACAAJ
14. J. Teich, L. Thiele, and L. Zhang, "Scheduling of partitioned regular algorithms on processor arrays with constrained resources," in *International Conference on Application-Specific Systems, Architectures, and Processors (ASAP)*, Aug. 1996, pp. 131–144. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/ASAP.1996.542808
15. F. Hannig, H. Dutta, and J. Teich, "Mapping a class of dependence algorithms to coarse-grained reconfigurable arrays: Architectural parameters and methodology," *International Journal of Embedded Systems*, vol. 2, no. 1/2, pp. 114–127, 2006.
16. J. Teich, "Invasive Algorithms and Architectures," *it - Information Technology*, vol. 50, no. 5, pp. 300–310, 2008.