Architecture Virtualization for Run-Time Hardware Multithreading on Field Programmable Gate Arrays

Michael Metzner[1](✉), Jesus A. Lizarraga[2], and Christophe Bobda[3]

[1] Department of Computer Science, University of Potsdam, Potsdam, Germany
metzner@cs.uni-potsdam.de
[2] Instituto Politecnico Nacional, Mexico City, Mexico
[3] CSCE Department, University of Arkansas Fayetteville, Fayetteville, AR, USA
cbobda@uark.edu

Abstract. In this paper a novel coarse-grained architecture virtualization for Field Programmable Gate Arrays (FPGA) is presented which can be used as basis for run-time dynamic hardware multithreading. The architecture uses on-chip networking to interconnect routers and computational elements providing a flexible and highly configurable structure. Quadratic routers are reducing total router count while ensuring short communication paths and minimal resource overhead.

Keywords: Architecture Virtualization · On-Chip-Networking · Hardware Multi-Threading

1 Introduction

Decreasing size of transistors and increasing packing density have led the transition from high-clocked single processors to multicore. However, pure multicore solutions without customized computing components will not be able to provide the required performance in many computational fields such as image processing, oil and gas exploration, and programmatic financial trading, which requires complex 3D convolutions on large data arrays and tight requirements on memory and IO [10]. Studies are predicting that increase in power as result of chip density will drastically reduce the usage of manycores to a maximum of 75% [7]. Heterogeneous architectures made upon general purpose processors and specialized computing components, intelligent methods to dynamically adapt chip resources to run-time computational and power consumption requirements can improve the performance of future multicores [10]. Reconfigurable logic such as FPGAs can be used for this purpose, as part of heterogeneous multiprocessors, either on the same die or as separate co-processor. However, for such platforms to be successful, the FPGA must be able to dynamically accommodate multiple threads running as hardware tasks. Dynamically placing and replacing those threads on the FPGA to allow critical parts of applications to be accelerated at

K. Sano et al. (Eds.): ARC 2015, LNCS 9040, pp. 167–178, 2015.
DOI: 10.1007/978-3-319-16214-0_14

run-time is done today using partial reconfiguration, a process technically and conceptually limited in currently FPGAs. Firstly, there is currently no support for communication among tasks arbitrarily placed on FPGA at run-time. The modular design flow used for partial reconfiguration in today's FPGAs places high restrictions on designs, requesting computational bins and direct communication tracks to be specified at compile-time. Once fixed, this structure cannot be modified at run-time, which drastically reduces the flexibility. The process is done manually for every design and is very sensitive to small changes. Secondly, FPGA configuration is very fine-grained, which increases reconfiguration overhead, particularly when tasks consuming large amount of resources must be frequently placed on the device. Finally, the programmability of FPGAs is essentially reduced to hardware design, a difficult process that prevents its adoption in the software community where the bulk of programmers are to be found. Hardware virtualization can help overcome the aforementioned hurdles by placing a coarse-grained reconfigurable hardware overlay between hardware tasks and raw FPGA. This reduces the configuration overhead, allows design decision to be made on-line and reduce the programmability burdens through compilers that can map instructions to coarse-grained processing elements. Overlay architectures recently proposed for FPGAs are based on existing concepts of coarse-grained reconfigurable architectures. They consist of microarchitectural components like ALU, multiplexers and direct interconnection among neighboring or distant cores. Existing overlays are accessible as a single large monolithic block that can accommodate only one hardware function at a time. Their communication is based on direct interconnection paradigm, which must be fixed at design time and never changed at run-time. As result, their communication infrastructure does not promote hardware multithreading. Tasks cannot to be randomly placed on the reconfigurable device at run-time and be accessible by other tasks, either on the same device or on external devices.

In this work, we propose a novel virtualization architecture for FPGAs which allows for unrestricted communication among hardware tasks running on the device. The architecture combines direct interconnection for high-performance at local level within a task's boundary, while using reduced overhead network-on-chip for global communication beyond component boundary. The architecture is based on a quadratic router access mechanism that drastically reduce the number of routers, thus leading to short communication paths and reduced resource overhead. We address the inefficient resource usage with a novel class of routers that can be dynamically recycled as computing resource in modules in the boundary of which they are placed at run-time. The viability of our approach is demonstrated with benchmarks in signal and image processing, which shows a performance difference between applications running on raw FPGAs and the same applications running on the virtualization layer.

The paper is organized as follow: Section 2 provides a motivation of the problem solved in this paper with help of a case study. In section 3 we discuss coarse-grained reconfigurable architectures and recent work in FPGA virtualization. In section 4, we present our architecture and explain design choices we

made to address problems of existing overlays. Section 5 shows our evaluation approach and the meaning of the results, while section 6 concludes the paper and provide some indication of our future work.

2 Motivation Example

The problem addressed in this work, namely hardware multitasking on coarse-grained reconfigurable devices is illustrated in Figure 1. Consider a device with a 2-D mesh and a global and flexible form of communication, in this case a network on chip. The first set of threads on the device represent the execution of a video communication application consisting of a video capture module, a video compressor after the H.264 protocol and a communication using WLAN. This implementation is constrained by the two already placed components CR1 and CR2, which cause the communication between the compressor and the WLAN modules to go through 5 hops. Upon completion of CR1 and CR2, a relocation can be done, which reduces the distance between the compressor and the WLAN to just 3 hops. The WLAN modules can further be replaced by a Bluetooth module with a much smaller footprint, lower power and performance, which reduces the distance between the two modules to just 2 hops, without altering the distance between the compressor and the video capture module.

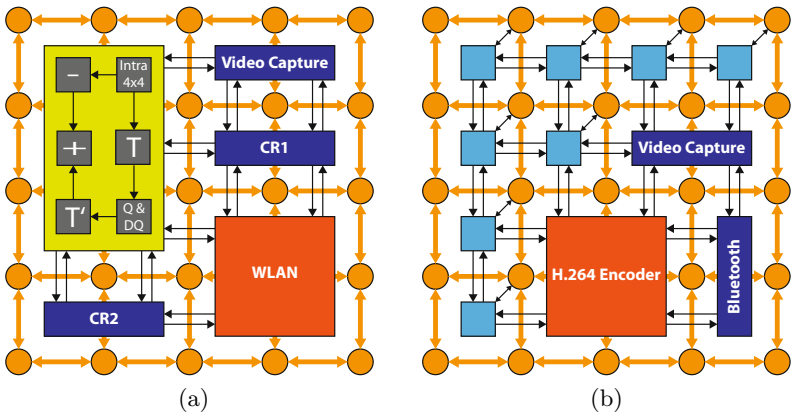

Fig. 1. Video capture, compression and transmission (left) and performance improvement through reconfiguration and relocation.

Resource organization for efficient temporal execution of those tasks at run-time is very challenging. *High performance* and *accessibility* are the two main objectives to be reached. High performance is defined by architectural considerations and efficient run-time organization and management, while accessibility allows hardware tasks running of the device to be reachable regardless of their placement location and later relocation. While virtualization on FPGA provides

an answer to the first challenge, existing implementation do not address accessibility since they are tailored only for fixed usage. As consequence, no existing coarse-grained architecture and overlay can handle the scenario previously described.

Our main contribution in this work is the design of a novel virtualization infrastructure allowing a flexible and unrestricted communication among hardware tasks arbitrarily placed on the virtual hardware at run-time.

3 Related Work

Published work in coarse-grained reconfigurable architectures and FPGA overlays such as [3,11,15] are essentially *dataflow machines*, ususally consisting of small arithmetic and logic units, register files, all of which are immersed in an switch-based interconnect structure. Data-flow machines are mostly used in systems as co-processors for acceleration of a single tasks, which can be replaced by configuring the entire device. Communication is based on direct interconnect among neighbor or distant processing elements. While data-flow machines fulfill high-performance requirements, they do not address accessibility, which is mostly enforced by the communication mechanism. Bus systems currently in use in system-on-chips represent a potential communication alternative in coarse-grained architectures. However, current bus-based architectures will not be able to sustain the communication load among hundreds of processing elements a coarse-grained overlay would be able to accommodate in future FPGAs. The overhead needed to manage hundred of connected cores on a single bus will drastically reduce system performance and offset all parallelism gains. Network-on-Chip (NoC) was designed as a means to overcome the communication bottleneck in high density chips in the future [1,6]. In NoC, messages are exchanged among cores, memories and peripherals, all of which are connected in a network infrastructure on the chip, instead of using dedicated wires. While the advantage of NoCs have been quantitatively and qualitatively elucidated in some publications [2,13], they also present some drawbacks, mostly due to the adoption of regular computer network paradigms in the chip domain. NoCs are usually provided as a 2-dimensional grid with routers placed at intersections between lines and columns and connected to homogeneous PEs. This approach creates three main problems: 1) communication within the boundary of complex components split across several PEs is message-based instead of directed and dedicated, thus leading to performance degradation; 2) Routers in the 2D Mesh are tailored for the general case. As consequence, dedicated topologies must be mapped to the 2D mesh, which increases resource redundancy; 3) the number of routers on many routes in the 2D are usually more than needed to route a packet to destination, which increases the routing time. Routing techniques such as virtual-cut-through are able to bypass some idle routers to improve the speed, but they don't address the redundancy issue. As solution to the three mentioned problems, Application Specific NoCs (ASNoC) have been proposed in several publications [5,8,12]. In a ASNoC large components spanning beyond a single PEs are implemented in

dedicated area, where direct interconnections within the module boundary are used to increase their speed. The topology of the network is usually determined by the communication pattern of the system, which eliminates redundancies and improves performance. However, the main drawback of ASNoCs is their lack of flexibility. For each application an ASNoC-Chip must be built, which incurs high design and production costs that most companies cannot afford. The resulting chip is optimal for the targeted application, but cannot be used for others applications. The Zippy architecture [4] attempts to provide an answer to accessibility and high-performance though a mixture of direct connection for neighbor PE and bus for global interconnect. Beside low performance, the use of a bus restricts the placement of components to certain location on the chip, thus limiting device's flexibility. In [14] preliminary results of a dynamic network-on-chip paradigms was presented. However, high-router count and large resource overhead of routers prevented the usage of this technology, particularly on FPGA devices.

In this work, we propose a novel architecture that combines coarse-grained elements, a mixture of network-on-chip and direct interconnection. With router reuse in components, our approach overcomes the problems of current coarse-grained reconfigurable devices and FPGA overlays.

4 The Novel Architecture

We present our architecture in this section and discuss our design choices. The overall architecture organization is first explained at high level, with emphasis on local and global connectivity. Thereafter, processing elements and interconnect are explained in more details.

4.1 General Device Organization

We propose the general organization of figure 2d, which goes far beyond the capabilities of the most advance architectures proposed so far. For instance the Zippy architecture [4](Figure 2a) uses direct interconnect locally and a bus for global interconnection. The bus is used to access every single PE from an external processor attached to the device. Interconnection between modules executing on chip must go through an external memory or external processor, which drastically slows down computations. Using a regular NoC (figure 2b) to provide communication among modules on the chip would be very efficient and flexible. However, the performance of large modules split across many processing element will suffer because of the message-based data exchange within a component boundary. Our proposed architecture reduces the number of router by 4 by providing one router for a group of four PEs, similar to the Arteris FlexNoC architecture (figure 2c). However, as oppose the FlexNoC, our proposed architecture also provides direct interconnection to neighbor PE, thus increasing the streaming bandwidth among PEs used in the same module. This fulfills the high-performance requirement. Each component consisting of several PEs will access the network and be reachable through one of the routers in the set of routers

it covers, which fulfills the accessibility requirement. The last innovation of our architecture is that remaining routers will be used to provide additional computational power to that component. To our best knowledge, no existing device has make use of this paradigm. We call our architecture *Quattuor* because it connects 4 PE on a single router.

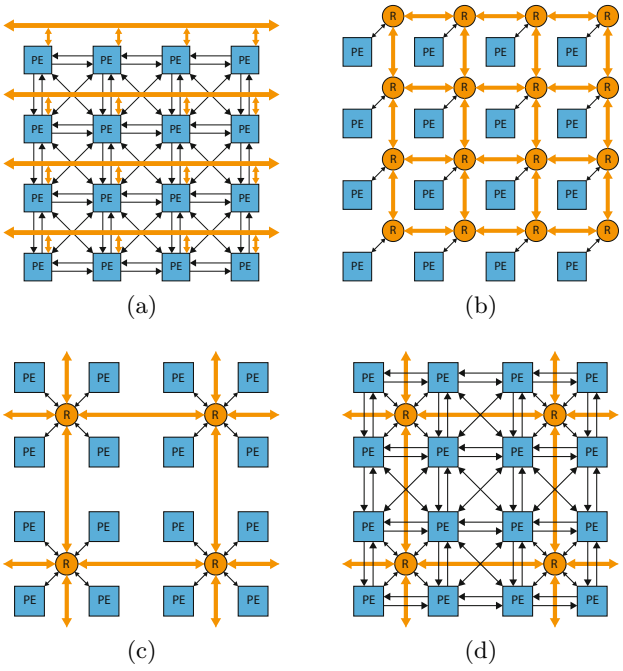

Fig. 2. The Zippy architecture [4] (2a) uses buses for global communication, while regular NoC (2b) exhibits high routers redundancy. Arterix FlexNoC [9] (2c) has less routers, but does not provide direct connection between neighbor modules. Our proposed architecture (2d) extends the best of existing architectures with direct interconnect, network-on-chip and router reuse.

4.2 Processing Elements

Like other coarse-grained reconfigurable architectures, the Quattuor processing elements consist of several arithmetic and logic units for data processing, register files for data storage, multiplexers and demultiplexers for controlling the flow of data among the units. The processing elements (PE) proposed in this section are kept relatively simple due to the early stage of development. More complex functional units are planned for the future. The organization of a Quattuor-PE is presented in Figure 3. The computation is done using ALUs, each of which provides 8 arithmetic and logic operations: addition, subtraction, multiplication, division, nop, and, or and xor.

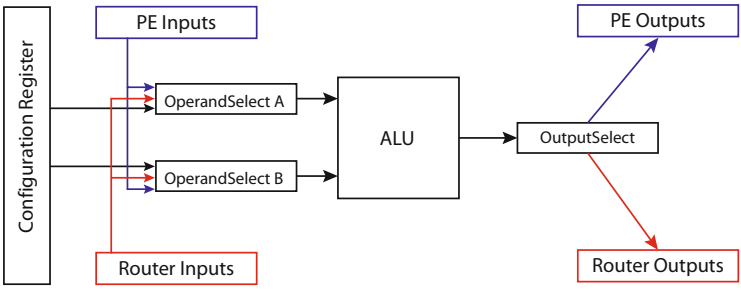

Fig. 3. Structure of a Quattuor-PE

The register files as well as the ALUs within a 4-block are connected to the router and neighbor PEs, so they can receive exchange operands and results of computation. The configuration of a PE is done by the configuration register which is connected to all units in the PE. Configuring a unit consists of defining its behavior at certain point until the next configuration or a repeating operation. Each processing element has a unique address and receives its configuration through the network using its closest router. Sending configuration data through the network removes the need for dedicated configuration lines present in existing coarse-grained architecture and overlays. Because we use one router for 4 PEs, the configuration path is reduced by 4. Furthermore depending on the geographic location on the chip, configuration data can be sent from the closest router to the PE, which further reduce configuration time.

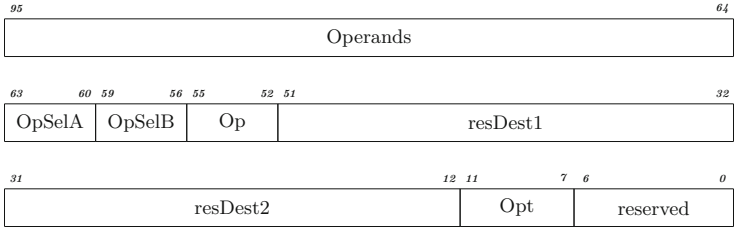

Fig. 4. Configuration Bits of a Quattuor-PE.

The format of configuration data is shown in Figure 4. It consists of a 96-bit packet. The bits 95-64 define operands for the ALU if they are needed. The following eight bits 63-56 control the operand multiplexers and define the source of each operand. The bits 55-52 select the operation for the ALU. The following bits 51-12 define the destinations for the result of the ALU operation. The possibility to define more than one destination offers a great flexibility to implement data-flow graphs in the network. The bits 11-8 define what should be done with the result. Bit 7 can be set to allow a continuous operation of the current PE configuration. The last bits are reserved for future use.

The flexible operand selection allows the mapping of complex data-flow graphs on the PEs. The default choice for the operand selection (0000) is that the ALU takes operands directly from the PE configuration. But to allow more complex data-flows the operand can also be taken from neighboring PEs. This direct connection enables fast exchange of results and operands between multiple ALUs and because of the Quattuor architecture complex operations can be implemented with multiple PEs while avoiding router communication.

To keep flexibility in the network operands can be also sent using the router interconnect. This functionality allows sending operation results to other partitions in the network while no PE has to be blocked for the transfer.

The result output of a PE can be configured in various ways. The default action is to define the ALU result as final result which will be reported back to the processor. The type of result will be sent to the Quattuor network output by using the router interconnect. Another possibility is to deliver the result to a neighboring PE using the direct interconnect. This is the fastest transfer possible in the network. For transferring results to a more distant location in the network the router interconnect can be used.

4.3 Router Architecture

Besides the processing elements, routers are one of the most important components on our dynamic network on chip concept. Our router is created with a set of FIFOs and a controller. There is one input FIFO that collects the four inputs of the router using a round-robin mechanism. All messages get collected and delivered to the designated outputs.

To transfer the configurations for the PEs a small header is added to the 96 configuration bits. This header defines the target router which is connected to the target PE. Considering the growing capabilities of upcoming FPGA series the router address is given by 16 bits allowing to identify 256 routers in every dimension which lead to 262144 PEs.

The routing is performed using dimension ordering routing, so called Y-X routing.

4.4 Quattuor-PE Direct Interconnect

As stated earlier, then main differentiation of the Quattuor-Architecture is the capability of using direct interconnect locally, within a task for fast data exchange among the sub-modules, and global communication using messages beyond components boundary. The first approach is useful when building computational data-flow graphs that leverages parallelism within a task. Direct communication between two neighbor PE takes just 1 clock cycle. In the second case, global communication goes through the router and depends on the distance between the two hardware tasks. We do not address this part in this work and simply assume that a function in change of run-time resource management will keep components which heavily communicate close to each other.

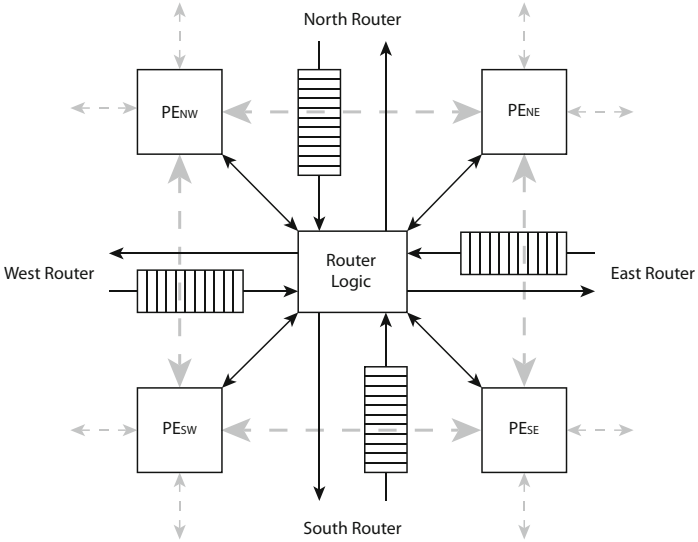

Fig. 5. Internal structure of routers with PEs

The delay of ALU operation depends on the implementation of the single operations. While bit manipulations like shift, and, or, xor etc. can be handled in a single clock cycle, more complex operations like division may take longer. As mentioned before the router implements a round-robin scheduling on the router inputs. That leads to a variable input delay of one to four clock cycles plus a delay depending on the FIFO implementation of the input buffers. All communication between routers implements a hand-shake-algorithm to ensure data integrity even if a router buffer is completely filled. Communication between router and the connected PEs is done in a single clock cycle.

5 Evaluation

For evaluation, we implemented the Quattuor-Architecture on various FPGA in diverse configurations. Our first objective was assess the resource consumption on diverse FPGAs, which will then give an estimate of the size of the overlay that can be accommodated on contemporary devices. Beside resource consumption, we also wanted to measure the performance decrease resulting from the implementation of applications on the overlay compare to their implementation on the raw FPGA.

Table 1 summarizes the resource consumption for single Quattuor-Elements as well as Quattuor-Arrays of size (in number of routers) of 3x3 (6x6 PEs) and 4x4 (8x8 PEs) in 3 different FPGA-Types: Cyclone IV (Altera EP4CE115F29C7), Cyclone V (Altera 5CSXFC6D6F31C6), Zynq-7000 (Xilinx XC7Z020-3-CLG484). The chosen FPGAs are considered the low-cost series from Altera and Xilinx. Our tests show that even these device are capable of hosting 3x3 and 4x4 router arrays

providing 36 respectively 64 PEs. Also an important characteristic of the design is the operational frequency. Here we can see that on the Zynq platform the design achieves a significantly higher frequency.

Table 1. Evaluation of the Quattuor-Architecture in term of area and speed on various FPGAs

Area	Cyclone-IV	Cyclone-V	Zynq-7000
3x3 Array	37%	37%	34%
4x4 Array	67%	67%	61%
Speed in MHz	Cyclone-IV	Cyclone-V	Zynq-7000
3x3 Array	147	173	223
4x4 Array	142	165	223

Our second part of the evaluation is more oriented on the application side and performed on a Cyclone IV FPGA running a NIOS2 processor together with a Quattuor network. For the benchmark the system-on-chip performs several operations which will be compared to raw software and raw FPGA implementations. The following section shows how a Quattuor configuration is derived from an application on the example of 3x3 matrix multiplication.

$$a = \begin{pmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{pmatrix} \qquad b = \begin{pmatrix} b_{00} & b_{01} & b_{02} \\ b_{10} & b_{11} & b_{12} \\ b_{20} & b_{21} & b_{22} \end{pmatrix} \tag{1}$$

$$c = \begin{pmatrix} a_{00}b_{00} + a_{01}b_{10} + a_{02}b_{20} \ldots a_{00}b_{02} + a_{01}b_{12} + a_{02}b_{22} \\ a_{10}b_{00} + a_{11}b_{10} + a_{12}b_{20} \ldots a_{10}b_{02} + a_{11}b_{12} + a_{12}b_{22} \\ a_{20}b_{00} + a_{11}b_{10} + a_{22}b_{20} \ldots a_{20}b_{02} + a_{21}b_{12} + a_{22}b_{22} \end{pmatrix} \tag{2}$$

Each component of the final result matrix can be implemented using three multiplications and two additions and be represented using the following data-flow graph (for the first element of the result matrix):

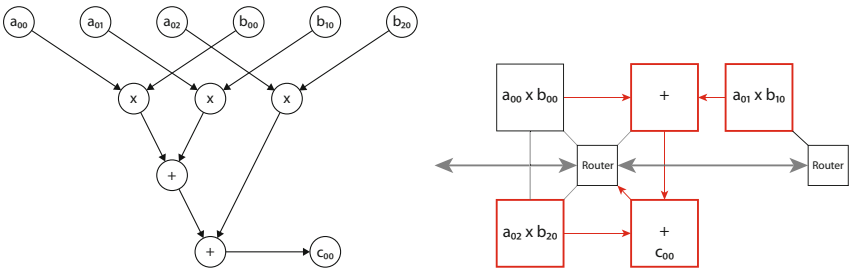

Fig. 6. Data-flow for matrix component c_{00} and mapping on PEs

Now there are some possibilities for the implementation. The first intention is that every component of the result matrix can be build by the same operations only changing the operands. That means one implementation would be to map this particular set of operations shown in the data-flow graph to the Quattuor array like in Figure 6b.

To evaluate the performance of this implementation we created a small system-on-chip on the Cyclone-IV FPGA consisting a NIOS2 soft-core CPU. The complete system (CPU, Quattuor, memory) is running at 50 MHz and the results are compared to others systems.

Table 2. Evaluation of the Quattuor-Performance for a 3x3 Matrix Multiplication

Application	Quattuor/NIOS2	NIOS2	Intel 2.1GHz	FPGA
3x3 Matrix	4.8ms (6us)	504us	2us	405us (720ns)
3x1 Matrix	1.8ms (2us)	120us	1us	149us (240ns)

In Table 2 the results for the performance evaluation are listed. We can state that in this example using the Quattuor network creates a large overhead. The measured execution time is about five milliseconds for the complete 3x3 matrix multiplication but calculating the results takes only 6 μs. The rest of the execution time is taken by the configuration of the PEs which has to be done for every component of the result matrix and the communication between the Quattuor network, CPU and memory. This can also be seen in comparison with a raw FPGA implementation. This was done by creating a small IP core with bus interface implementing the data-flow as seen in Figure 6a. Even in this implementation we can see that for operand writing to and result reading from the IP core there is a large overhead.

6 Conclusion and Future Work

In this paper, we presented a concept and implementation of a novel coarse-grained architecture virtualization for Field Programmable Gate Arrays. The Quattuor architecture allows for run-time dynamic hardware multithreading by using a combination of direct interconnection to allow for high-performance communication within task boundaries, low-overhead network-on-chip for communication beyond component boundaries. Our evaluation shows a performance lost toward raw software and FPGA implementations which is generated by the communication overhead which is needed to configure the PEs via the bus connection of the Quattuor IP core and the interconnect latency in the network. The future work on the topic will focus on effective PE configuration and configuration reusing to minimize the impact for real world applications. Also in our future work we are going to investigate how to automatically map applications on the network without requiring users to recode their programs. For instance binary synthesis will be made possible with our approach, with the goal of extracting parallel kernel at run-time and synthesizing them for FPGA acceleration.

References

1. Benini, L., Micheli, G.D.: Networks on chips: A new soc paradigm. IEEE Computer **35**(1), 70–78 (2002)
2. Bjerregaard, T., Mahadevan, S.: A survey of research and practices of network-on-chip. ACM Comput. Surv. 38(1), June 2006
3. Brant, A., Lemieux, G.G.: Zuma: An open fpga overlay architecture. In: Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 93–96 (2012)
4. Christian Plessl, G., Platzner, M.: Hardware virtualization on dynamically reconfigurable processors. In: Khalgui, M., Hanisch, H.-M. (eds:) Reconfigurable Embedded Control Systems: Applications for Flexibility and Agility. IGI Global (2011)
5. Dall'Osso, M., Biccari, G., Giovannini, L., Bertozzi, D., Benini, L.: xpipes: a latency insensitive parameterized network-on-chip architecture for multi-processor socs. In: Proceedings of the 21st International Conference on Computer Design, ICCD 2003, p. 536. IEEE Computer Society, Washington, DC (2003)
6. Dally, W.J., Towles, B.: Route packets, not wires: on-chip interconnection networks. In: Proceedings of the Design Automation Conference, Las Vegas, NV, pp. 684–689, June 2001
7. Esmaeilzadeh, H., Blem, E., St. Amant, R., Sankaralingam, K., Burger, D.: Dark silicon and the end of multicore scaling. SIGARCH Comput. Archit. News, 39(3), 365–376 (2011)
8. Goossens, K., Dielissen, J., Radulescu, A.: Aethereal network on chip: Concepts, architectures, and implementations. IEEE Des. Test **22**(5), 414–421 (2005)
9. A. Inc., Arteris FlexNoC Interconnect IP. Arteris, Inc. Sunnyvale, CA, USA
10. Michael, M.C.: Challenges in high speed reconfigurable computing (June 2009)
11. Mirsky, E., DeHon, A.: MATRIX: A reconfigurable computing architecture with configurable instruction distribution and deployable resources. In: Pocek, K.L., Arnold, J. (eds.) IEEE Symposium on FPGAs for Custom Computing Machines, pp. 157–166. IEEE Computer Society Press, Los Alamitos (1996)
12. Penolazzi, S., Jantsch, A.: A high level power model for the nostrum noc. In: Proceedings of the 9th EUROMICRO Conference on Digital System Design. DSD 2006, pp. 673–676. IEEE Computer Society, Washington, DC (2006)
13. Probell, J.: Routing Congestion: The Growing Cost of Wires in Systems-on-Chip. Arteris, Inc., Sunnyvale, CA, USA
14. O. F. B. Review. In
15. Toi, T., Awashima, T., Motomura, M., Amano, H.: Time and space-multiplexed compilation challenges for dynamically reconfigurable processors. In: Midwest Symposium on Circuits and Systems, pp. 1–4 (2011)