VLIW-based FPGA Computation Fabric with Streaming Memory Hierarchy for Medical Imaging Applications

Joost Hoozemans, Rolf Heij, Jeroen van Straten, and Zaid Al-Ars

Delft University of Technology
j.j.hoozemans@tudelft.nl, r.w.heij@student.tudelft.nl, j.van.straten-1@tudelft.nl, z.al-ars@tudelft.nl

Abstract. In this paper, we present and evaluate an FPGA acceleration fabric that uses VLIW softcores as processing elements, combined with a memory hierarchy that is designed to stream data between intermediate stages of an image processing pipeline. These pipelines are commonplace in medical applications such as X-ray imagers. By using a streaming memory hierarchy, performance is increased by a factor that depends on the number of stages (7.5× when using 4 consecutive filters). Using a Xilinx VC707 board, we are able to place up to 75 cores. A platform of 64 cores can be routed at 193MHz, achieving real-time performance, while keeping 20% resources available for off-board interfacing.
Our VHDL implementation and associated tools (compiler, simulator, etc.) are available for download for the academic community.

1 Introduction

In contemporary medical imaging platforms, complexity of image processing algorithms is steadily increasing (in order to improve the quality of the output while reducing the exposure of the patients to radiation). Manufacturers of medical imaging devices are starting to evaluate the possibility of using FPGA acceleration to provide the computational resources needed. FPGAs are known to be able to exploit the large amounts of parallelism that is available in image processing workloads. However, current workflows using High-Level Synthesis (HLS) are problematic for the medical application domain, as it impairs programmability (increasing time-to-market) and maintainability. Additionally, some of the image processing algorithms used are rather complex and can yield varying quality of results. Therefore, in this paper, we propose a computation fabric on the FPGA that is optimized for the application domain, in order to provide acceleration without sacrificing programmability. By analyzing the structure of the image processing workload type (essentially a pipeline consisting of multiple filters operating on the input in consecutive steps), we have selected a suitable processing element and designed a streaming memory structure between the processors.

The image processing workload targeted in this paper consists of a number of filters that are applied to the input data in sequence. Each filter is a stage

in the image processing pipeline. The input stage of a filter is the output of the previous stage - the stages *stream* data to each other. Making sure these transfers are performed as efficiently as possible is crucial to provide high throughput.

The processing element used in this work is based on a VLIW architecture. These type of processors are ubiquitous in areas such as image and signal processing. They are known for their ability to exploit Instruction-Level Parallelism (ILP) while reducing circuit complexity (and subsequently power consumption) compared to their superscalar counterparts. In the medical imaging domain, power consumption is not a main concern, but as image processing workloads can be divided into multiple threads easily, a reduction in area utilization will likely result in an increase in total throughput.

The remainder of this paper is structured as follows: Section 2 discusses related work, Section 3 discusses the implementation details, Section 4 and 5 present the evaluation and results, and Section 6 provides conclusions and future work.

2 Related work

A prior study on using VLIW-based softcores for image processing applications is performed in [1], showing that a VLIW-based architecture has advantages over a scalar architecture such as the MicroBlaze in terms of performance versus resource utilization. In [2], an FPGA-based compute fabric is proposed using the LE-1 softcore (based on the same Instruction Set Architecture - VEX), targeting medical image processing applications. This work focuses solely on offering a highly multi-threaded platform without providing a memory hierarchy that can sustain the needed bandwidth through the pipeline. A related study on accelerating workloads without compromising programmability is [3], with one of the design points being a convolution engine as processing element. A well-known prior effort, and one of the inspirations of this work, uses softcores to provide adequate acceleration while staying targetable by a high level compiler is the Catapult project [4]. The target domain is ranking documents for the Bing search engine. A related effort that aims to accelerate Convolutional Neural Networks is [5]. However, this project did not aim to conserve programmability (only run-time reconfigurability), as the structure of this application does not change enough to require this. In the image processing application domain, [6] provides a comparison of convolution on GPU or FPGA using a Verilog accelerator, [7] and [8] present resource-efficient streaming processing elements, and [9] introduces a toolchain that targets customized softcores.

3 Implementation

The computation fabric developed in this work consists of two facets; the processing elements and the memory hierarchy, as shown in Figure 1. The implementation of both will be discussed in this section. Then, the process of designing a full platform using these components is discussed.

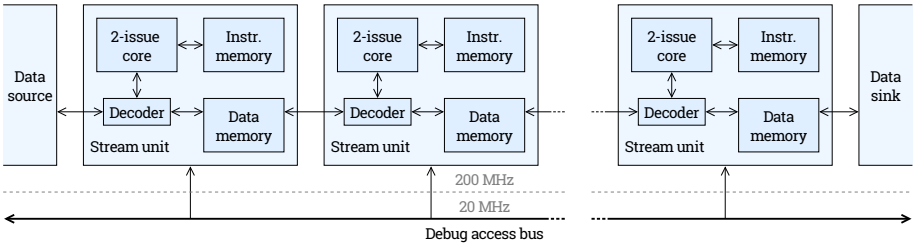

Fig. 1. Organization of a single stream of processing elements (Stream unit) and the streaming connections that link the data memories. Each processor can access the memory of its predecessor. Each processor's memories and control registers can be accessed via a bus that runs on a low clock frequency to prevent it from becoming a timing-critical net.

3.1 Processing elements

This section describes the design and implementation of our fabric. The processor cores in the fabric are derived from the ρ-VEX processor [10]. The ρ-VEX processor is an VLIW processor based on the VEX ISA introduced by Fisher et al [11]. The ρ-VEX processor has both run-time and design-time reconfigurable properties, giving it the flexibility to run a broad selection of applications in an efficient way.

Image processing tasks are highly parallelizable in multiple regards; 1) The code is usually computationally dense, resulting in high ILP, and 2) Every pixel can in theory be calculated individually and it is easy to assign pixels to threads (by dividing the image into blocks). In other words, there is an abundance of Thread-Level Parallelism (TLP). Exploiting TLP is usually more area efficient than exploiting ILP - increasing single-thread performance comes at a high price in power and area utilization and will quickly show diminishing returns. This is why GPUs exploit TLP as much as possible by using many small cores. Therefore, the processing elements of our fabric will use the same approach and we will use the smallest 2-issue VLIW configuration as a basis. This will still allow it to exploit ILP by virtue of having multiple issue slots and a pipelined datapath.

By placing multiple instances of our fabric on an FPGA, TLP can be exploited in two dimensions; by processing multiple blocks, lines or pixels (depending on the filter) concurrently, and by assigning each step in the image processing pipeline to a dedicated core (pipelining on a task level in contrast to the micro-architectural level).

To explore the design space of the processor's pipeline organization, we have measured code size and performance of a 3x3 convolution filter implemented in C. This convolution code forms a basis with which many operators can be applied to an image depending on the kernel that is used (blurring, edge detection, sharpening) so it is suitable to represent the application domain. The main loop can be unrolled by the compiler using pragmas. Figure 2 lists the performance using different levels of loop unrolling for different organizations

of a 2-issue ρ-VEX pipeline; the default pipeline with 5 stages and forwarding, one with 2 additional pipeline stages to improve timing, and one using the longer pipeline and with Forwarding (FW) disabled to further improve timing and decrease FPGA resource utilization. Loop unrolling will allow the compiler to fill the pipeline latency with instructions from other iterations. The performance loss introduced is reduced from 25% to less than 2% when unrolling 8 times. Additionally, disabling forwarding reduces the resources utilization of a core allowing more instances to be placed on the FPGA (see Figure 3).

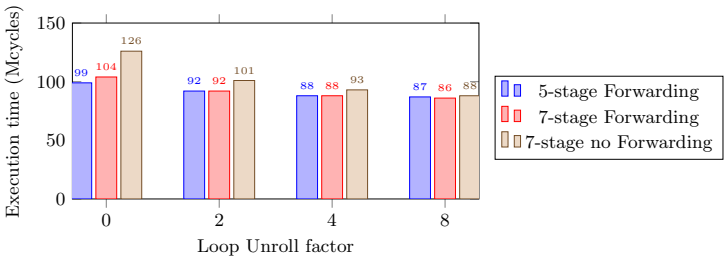

Fig. 2. Execution times of a 3x3 convolution filter on a single processor using different loop unrolling factors.

3.2 Memory hierarchy

In our fabric, processing elements are instantiated in 'streams' of configurable length. This length should ideally be equal to the number of stages in the image processing pipeline. Each stage will be executed by a processor using the output of the previous processor. A connection is made between each pair of ρ-VEX processors in a stream, so that a core can read the output of the previous step (computed by the previous core in the stream) and write the output into its own data memory (making it available for reading by the next core in the stream). The memory blocks are implemented using dual-port RAM Blocks on the FPGA. Each port can sustain a bandwidth of one 32-bits word per cycle per port, so both processors connected to a block (current, next) can access a block without causing a stall. The blocks are connected to the processors by means of a simple address decoder between the memory unit and the data memories.

The first and last core should be connected to DMA (Direct Memory Access) units that move data to and from input and output frame buffers (eventually going off-board).

3.3 Platform

The VHDL code of the components is written in a very generic way and there are numerous parameters that can be chosen by the designer. First of all, the ρ-VEX processor can be configured in terms of issue width, pipeline configuration,

forwarding, traps, trace unit, debug unit, performance counters, and caches. Secondly, there is an encompassing structure that instantiates processors in streams. The number of streams and length per stream are VHDL generics.

4 Experimental setup

Since the target application of the designed system is related to medical image processing, an X-ray sample image is used as input for the evaluation. Typical medical imagers work with images that have a size of 1000 by 1000 pixels. The dimensions of our benchmark images are 2560 by 1920 pixels. The image is resized to other dimensions in order to determine the scalability of system performance. Each pixel is represented by a 32-bit value (RGBA). Using a technique described in the following section, the image may be scaled down to 1280 by 960 and 640 by 480 pixels.

A workload of algorithms based on a typical medical image processing pipeline is used. The first step in the image processing pipeline is an interpolation algorithm used to scale the size of the source image. The bi-linear and nearest neighbor interpolation algorithms both have the same computational complexity making them equally feasible. Because of its slightly higher flexibility, we select the bi-linear interpolation algorithm for the evaluation. Secondly, a gray scaling algorithm is applied. This algorithm is selected because it operates on single pixels in the input dataset. The third stage is a convolution filter that sharpens the image, followed by the final stage, an embossing convolution filter.

5 Evaluation results

5.1 Resource utilization

We have synthesized the platform using various configurations targeting the Xilinx VC707 evaluation board. As stated, the pipeline organization of the processing elements has influence on the resource utilization and timing. In Figure 3, 4 options have been evaluated using the standard synthesis flow (unconstrained). With forwarding enabled, the platform completely fills the FPGA using 64 cores. When forwarding is disabled, this can be increased to 75.

Additionally, we have performed a number of runs where we created simple placement constraints that steered the tool towards clustering the cores per stream so that they are aligned on the FPGA in accordance with their streaming organization. A single stream consisting of 4 cores achieves an operating frequency of 200MHz. Using 16 streams, timing becomes somewhat more difficult as the FPGA fabric is not homogeneous (some cores will need to traverse sections of the chip that are reserved for clocking, reconfiguration and I/O logic, and the distribution of RAM Blocks is not completely uniform). Still, this configuration achieves an operating frequency of 193 MHz at 80% LUT utilization, leaving room for interfacing with off-board electronics.

Pipeline organization		Cores	Resource utilization			Freq.
Forwarding	Stages		LUT	FF	BRAM	(MHz)
Enabled	7	64	99%	29%	81%	149
Enabled	5	64	93%	26%	81%	103
Disabled	7	75	96%	33%	95%	162
Disabled	5	75	98%	30%	95%	143
Disabled	7	4	5%	2%	5%	200
Disabled	7	64	82%	28%	81%	193

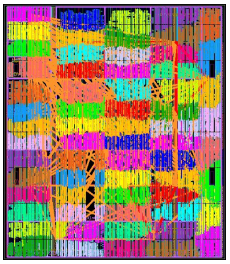

Fig. 3. Resource utilization and clock frequency of different platform configurations on the Xilinx VC707 FPGA board. The layout of the 64-core, 193MHz platform on the FPGA is depicted on the right. Manually created placement constraints were used to group each stream together.

5.2 Image processing performance

Figure 4 depicts the execution times of a 3x3 convolution filter on the various platforms, taking into account the number of cores, execution frequency, code performance on the pipeline organization (using 8x loop unrolling).

The results on using the streaming architecture for consecutive filters versus the same system with caches and a bus are depicted in Figure 5. Enabling streaming of data results in speedup of 7.5 times. Processing an image sized 1280 by 960 requires 94.72 million clock cycles (see Figure 5). Using 16 streams consisting of 4 cores (64 cores in total) at an operating frequency of 193 MHz, this would mean that our fabric can process approximately 34 frames per second.

Note that the difference will increase with the number of stages, so the fabric will perform better with increasingly complex image processing pipelines.

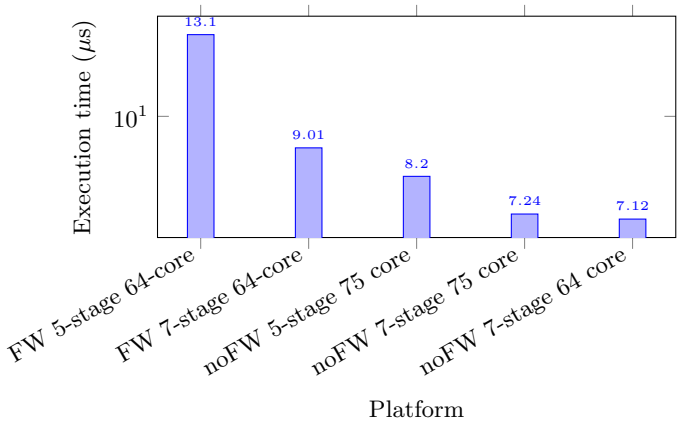

Fig. 4. Execution times of a convolution 3x3 filter for the platforms in the design-space exploration as listed in Figure 3 using 8x loop unrolling (from Figure 2).

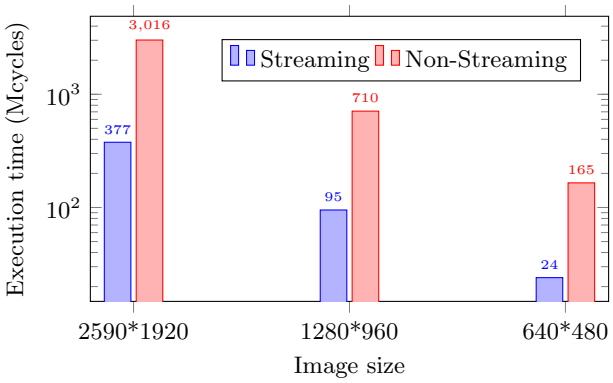

Fig. 5. Execution times of a 4-stage image processing pipeline on a streaming versus non-streaming platform using different image sizes

6 Conclusion

In this paper, we have introduced and evaluated an implementation of a FPGA-based computation fabric that targets medical imaging applications by providing an image processing pipeline-oriented streaming memory hierarchy combined with high-performance VLIW processing elements. We have shown that the streaming memory hierarchy is able to reduce bandwidth requirements and increase performance by a factor of 7.5 times when using a single stream of only 4 processing stages. The platform stays fully targetable by a C-compiler and each core can be instructed to perform an individual task. The platform is highly configurable and designers can modify the organization to best match their application structure. For future work, there is room for further design-space exploration of the processing elements in terms of resource utilization versus performance, introducing design-time configurable instruction sets, increasing the clock frequency, and other architectural optimizations. The platform, simulator and toolchain are available for academic use at `http://www.rvex.ewi.tudelft.nl`.

Acknowledgment

This research is supported by the ARTEMIS joint undertaking under grant agreement no. 621439 (ALMARVI).

References

1. J. Hoozemans, S. Wong, and Z. Al-Ars, "Using VLIW Softcore Processors for Image Processing Applications," in *Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS), 2015 International Conference on*, pp. 315–318, IEEE, 2015.

2. D. Stevens, V. Chouliaras, V. Azorin-Peris, J. Zheng, A. Echiadis, and S. Hu, "BioThreads: a novel VLIW-based chip multiprocessor for accelerating biomedical image processing applications," *IEEE transactions on biomedical circuits and systems*, vol. 6, no. 3, pp. 257–268, 2012.
3. T. Nowatzki, V. Gangadhan, K. Sankaralingam, and G. Wright, "Pushing the limits of accelerator efficiency while retaining programmability," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 27–39, IEEE, 2016.
4. A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, J. Gray, *et al.*, "A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services," *IEEE Micro*, vol. 35, no. 3, pp. 10–22, 2015.
5. K. Ovtcharov, O. Ruwase, J.-Y. Kim, J. Fowers, K. Strauss, and E. S. Chung, "Accelerating Deep Convolutional Neural Networks using Specialized Hardware," *Microsoft Research Whitepaper*, vol. 2, 2015.
6. L. M. Russo, E. C. Pedrino, E. Kato, and V. O. Roda, "Image Convolution Processing: A GPU versus FPGA Comparison," in *2012 VIII Southern Conference on Programmable Logic*, pp. 1–6, March 2012.
7. P. Wang, J. McAllister, and Y. Wu, "Soft-core Stream Processing on FPGA: An FFT Case Study," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 2756–2760, May 2013.
8. P. Wang and J. McAllister, "Streaming Elements for FPGA Signal and Image Processing Accelerators," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, pp. 2262–2274, June 2016.
9. B. Bardak, F. M. Siddiqui, C. Kelly, and R. Woods, "Dataflow toolset for Soft-core Processors on FPGA for Image Processing Applications," in *2014 48th Asilomar Conference on Signals, Systems and Computers*, pp. 1445–1449, Nov 2014.
10. S. Wong and F. Anjam, "The Delft Reconfigurable VLIW Processor," in *Proc. 17th International Conference on Advanced Computing and Communications*, (Bangalore, India), pp. 244–251, December 2009.
11. J. A. Fisher, P. Faraboschi, and C. Young, *Embedded Computing: A VLIW Approach to Architecture, Compilers, and Tools*. 500 Sansome Street, Suite 400, San Francisco, CA 94111: Morgan Kaufmann Publishers, 2005.