Using VLIW Softcore Processors for Image Processing Applications

Joost Hoozemans, Stephan Wong, Zaid Al-Ars
Delft University of Technology
Mekelweg 4, 2628 CD Delft, The Netherlands

***Abstract*—The ever-increasing complexity of advanced high-resolution image processing applications requires innovative solutions to ensure addressing this issue efficiently and cost effectively. This paper discusses the utilization of reconfigurable general-purpose softcore processors in image processing applications such that hardware resources are efficiently utilized and at the same time ensure high image processing performance for the targeted application. Results show that the rVEX softcore processor can achieve remarkably better performance compared to the industry-standard Xilinx MicroBlaze (up to a factor of 3.2 times faster) on image processing applications.**

I. Introduction

Whenever image/video processing is an integral function of the end user product, the stringent performance and power consumption requirements (that are hard to meet in software) are often fulfilled by embedding the imaging/video functionality as hardware accelerators implemented in FPGAs or as ASICs. The main advantages of embedded accelerator implementations of image/video processing functions lie in computation speed, high energy and area efficiency, etc. However, the transition from pure software prototypes towards production-grade FPGA or ASIC-based systems is associated with high engineering and manufacturing cost. Moreover, hardware development requires expensive toolsets and dedicated know-how, which usually results in a relatively high per-unit cost due to smaller production quantities and higher customization overhead. As a result, the development cycles of hardware approaches increasingly lag behind the demands of the fast-paced markets.

In recent years, however, software-based approaches on commodity hardware, notably on embedded graphics processors (GPUs) and multi-core CPUs, have increasingly gained attention. Although GPUs seem like the most logical choice to accelerate imaging applications [1], they have a number of characteristics that cause them to fail requirements in some cases. In these cases, FPGAs may be preferred. Firstly, they draw considerable amounts of power. Secondly, some product areas such as medical imaging systems (e.g., X-ray) require the availability of system components over an extended period of time (up to 15 years). However, in these areas there are conflicting requirements such as a high degree of maintainability, that are normally not compatible with FPGA acceleration (changes to the software could lead to required changes in the acceleration fabric which is tedious). Therefore, it is highly desirable to have an acceleration fabric that is more easily programmable than the reconfigurable logic itself. Putnam et al. [2] shows the viability of using application-tailored softcores to speed up datacenter applications. [3] presents a softcore specifically designed to perform fast fourier transforms at efficiency levels comparable to that of dedicated FPGA circuits. A softcore-based environment benefits from standardized hardware design and a pre-existing development platform and toolchain that allows it to be programmed using common programming languages. A single softcore will not be able to achieve performance levels similar to dedicated accelerators written in VHDL (or synthesized using high-level synthesis). However, the total system performance for data-intensive applications (such as image processing) will often be bound by the available memory bandwidth (as is true for any multicore system [4]). Moreover, the parallel nature of image processing provides a high level of scalability for multicore systems. This means that if the number of softcores that can be placed on the FPGA is sufficiently high as to achieve "wire-speed performance" (i.e., fully utilize the available bandwidth to the FPGA), the application speedup of the softcore-based system will be equal to an implementation that uses dedicated FPGA accelerators but with reduced development effort and increased maintainability.

In this paper, we propose to use the rVEX softcore based on the VEX ISA for image processing applications (which is one of the main application domains for this architecture) considering the aforementioned scenario. Our rVEX softcore implementation is design-time reconfigurable and run-time parametrizable which allows it to adapt to varying requirements of applications. This has the promise of providing low development cost and good maintainability as well as efficient resource utilization to achieve efficient image processing power. In this paper, we will show that the rVEX exhibits good performance in the application domain compared to an industry standard softcore (the Xilinx MicroBlaze).

The paper is organized as follows. Section II discusses related work. Section III presents the rVEX platform including the ISA, the toolchain and the softcore design. Section IV discusses the applications used in the evaluation. Section V presents the test setup and the measurement results, and Section VI concludes the paper and discussed possible future directions for research.

II. Related work

Spyder [5] appeared as the first softcore VLIW processor. The provided toolchain was not complete and the processor was not run-time reconfigurable. An FPGA-based design of a softcore VLIW processor based on the ISA of the Altera NIOS-II soft processor is presented in [6]. The compilation scheme consists of a Trimaran [7] as the frontend and the extended NIOS-II as the back-end. Due to the licensed Altera NIOS-II, this VLIW design is not very flexible and not open-source. Additionally, the design is not run-time reconfigurable. In [8], a modular design of a VLIW processor is reported. Certain parameters of the processor architecture could be altered in a modular fashion. In [9], the architecture and micro-architecture of a customizable softcore VLIW processor are presented. Additionally, tools are discussed to customize, generate, and program this processor. The limitation is the absence of a compiler. A VLIW processor with reconfigurable instruction set is presented in [10]. In this case, a reconfigurable unit is coupled to a VLIW processor. The co-processor can be configurable for any custom instruction. The rVEX is different from this design in the sense that it does not couple a reconfigurable co-processor. We can add a custom unit to the data paths of our processor at design time and reconfigure the issue slots at run-time. In [11], we present the rationale and the design and implementation of an open-source softcore VLIW processor. This processor is design-time parametrized and can be configured to make its issue-width adjustable during run-time [12][13].

III. The rVEX platform

A. The VEX system: ISA and toolchain

The VEX stands for VLIW Example [14]. The VEX is developed by Hewlett-Packard (HP) and STMicroelectronics. The VEX instruction set architecture (ISA) is a 32-bit clustered VLIW ISA that is scalable and customizable to individual application domains. The VEX ISA is loosely modeled on the ISA of HP/ST Lx (ST200) family of VLIW embedded cores [14]. Based on trace scheduling, the VEX C compiler is a parameterized ISO/C89 compiler. A flexible programmable machine model determines the target architecture, which is provided as input to the compiler. A VEX software toolchain including the VEX C compiler and the VEX simulator is made freely available by the Hewlett-Packard Laboratories [15].

B. The rVEX VLIW processor

The rVEX is a configurable (design-time) open-source VLIW softcore processor [12]. The ISA is based on the VEX ISA [15]. Different parameters of the rVEX processor, such as the number and type of functional units (FUs), number of multiported registers (size of register file), number and type of accessible FUs per syllable, width of memory buses, and different latencies can be changed at design time.

Figure 1 depicts the organization of a 32-bit, 4-issue rVEX VLIW processor. The rVEX processor consists of fetch, decode, execute, and writeback stages/units. Operations take place in either the parallel Arithmetic logic unit (A) and multiplier (M) units, or the branch (CTRL) or load/store (MEM) units. All jump and branch operations are handled by the CTRL unit, and all data memory load and store operations are handled by the MEM unit. The different write targets could be the general register (GR) file, branch register (BR) file, or data memory. All operations normally have a delay of one cycle, except for MEM and MUL operations which need an extra cycle. The core contains forwarding logic to minimize pipeline stalls. Additionally, the rVEX processor supports reconfigurable operations, as the VEX compiler supports the use of custom instructions via pragmas within an application code. The instruction and data caches for the processor are implemented with BRAMs (Block RAM resources on the FPGA). The GRLIB SoC library [16] is used to connect the processor core to off-chip DDR memory and peripherals via an AMBA bus system. This setup allows us to use any IP that is compatible with this bus, and to use existing tools to connect to the board in order to load applications, start/stop the core, etc. The GRLIB library also contains the framebuffer used to visually inspect the result images.

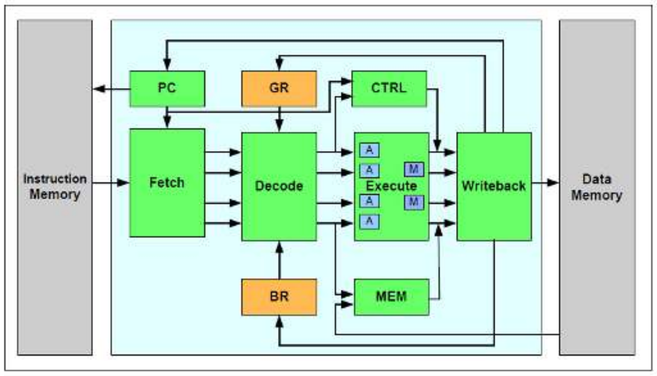
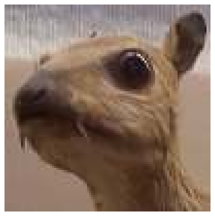


Fig. 1: Design overview of a 4-issue instance of the rVEX VLIW softcore processor.

The rVEX core can be configured at design time to be dynamically (run-time) reconfigurable or not. When configured to be dynamic, it can couple or decouple its datapaths to either run in a single-core mode with a large issue-width, or in a multi-core mode with smaller processors that can run separate tasks or threads. Dynamic reconfigurability will result in the flexibility to balance instruction-level parallelism (ILP) for high performance on a single thread with thread-level parallelism (TLP) for applications with low ILP but that can benefit from utilizing multiple threads. This comes at a cost of increased FPGA resource utilization. Partial reconfiguration is not needed for this concept to work; the principle is applicable also for ASIC implementation. This paper focuses on the architecture and not the dynamic reconfigurability. The applications used in this paper for evaluation do not exhibit dynamic behavior (see IV) and as such are not suitable to study these properties. Therefore, the processor core used in this paper is a static (not run-time reconfigurable), 4-issue VLIW. Using a higher issue width and/or dynamic reconfigurability will result in increased resource utilization and possibly lower operating frequency.

IV. Image processing applications

To evaluate the suitability of our architecture for imaging applications, we implemented two basic algorithms that



$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

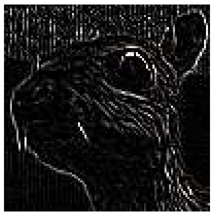


Fig. 2: Example of a convolution kernel that performs edge detection [17].

are commonly found in the application domain: a greyscale converter and a convolution filter with adaptable filter size. Both applications can be used for varying image sizes and the convolution filter can be configured to use different kernels, each performing a different operation on the image. The execution time is independent of the values used in the kernel. The images are represented in memory as an array containing a 32-bit word per pixel. This representation allows it to be displayed directly using a framebuffer device on the FPGA to facilitate visual inspection of the resulting images.

The greyscale converter is essentially a single loop performing a single operation on every pixel. The convolution filter performs a number of operations per pixel, depending on the size of the kernel. Both applications are representative for image processing steps in a medical imaging system. The greyscale converter represents the step of assigning a color value to the output of the sensor, depending on the precision of its output (which is often higher than 8 bits). The convolution filter can be used for a range of operations such as edge detection and image sharpening.

Care has been taken to ensure that the measurements contain as little overhead as possible. The programs do not contain input/output inside the measured parts of the program. The programs run without operating system and there are no interrupts enabled in the system.

V. Results and discussion

We use the Xilinx ML605 board as evaluation platform. The rVEX is synthesized at 75 MHz and the MicroBlaze was synthesized using the platform studio base system wizard included in the Xilinx ISE 13.4 toolset. The MicroBlaze was created using the “maximum performance” setting of the wizard. The system contains a core that is clocked at 150 MHz and an AXI bus at 75 MHz (as are the default maximum settings). The MicroBlaze contains a multiplication unit but no division or floating point unit. Both cores contain 32 KiB of instruction memory and 1 KiB of data memory. In both cases, the `.text` section of the programs is small enough to fit in the instruction cache. Using data cache sizes of more than 1 KiB did not result in any performance increase for either processors. However, a larger cache prevents the MicroBlaze from meeting its timing at 150 MHz, which necessitates lower clock frequencies, further reducing the MicroBlaze performance. Since the rVEX runs at a lower frequency of 75 MHz, cache sizes can be increased without further lowering the frequency. The 150

TABLE I: Synthesis results

	MicroBlaze		rVEX	
	#	%	#	%
Registers	17,477	5%	10,927	3%
LUTs	15,099	10%	18,900	12%
Slices	7376	19%	7506	19%

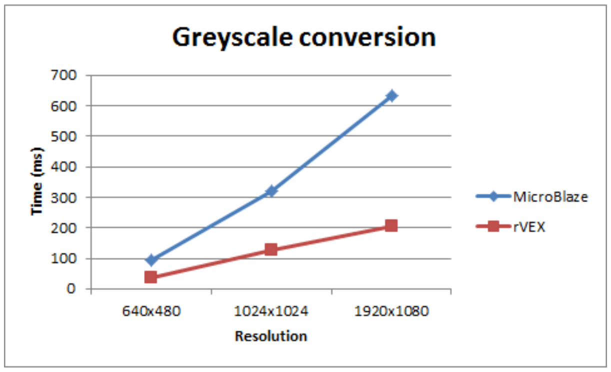


Fig. 3: Execution time for the greyscale conversion algorithm.

MHz frequency and 1 KiB data cache size were chosen to keep the comparison between the two processors as fair as possible. Both platforms use the off-chip DDR RAM to store the program and data. Synthesis results can be seen in Table I. The resource utilization of both platforms is comparable, with the rVEX utilizing slightly more lookup tables (LUTs) and the MicroBlaze utilizing more registers.

The same code is compiled for both cores, with preprocessor macros selecting the timer and print functions, and the memory locations of the image in/output according to the location of the DDR RAM in each platform’s memory map. For the rVEX, our VEX port of the Open64 compiler is used with full optimization (`-O`). For the MicroBlaze, the default GCC-based compiler is used that is included with the Xilinx toolset. Full optimization was also enabled, but for convolution it appeared that optimizing for size (`-Os`) resulted in significantly better performance. Therefore, size optimization was used when compiling the convolution code for the MicroBlaze.

The resolutions used in these experiments include two industry standards (VGA 640x480 and HD 1080p) as well as 1024x1024, a resolution taken from the requirements of an actual medical imaging system.

The execution of the rVEX is measured by resetting the execution cycle counter that is included in the platform before the program enters the calculation section (thereby removing the overhead of initialization code and printing startup messages to the UART) and reading the number of cycles again when execution has finished. The execution time of the MicroBlaze is measured by starting a timer unit attached to the AXI bus running at 75 MHz, reading its value before and after running the calculations and printing the difference to the UART.

The results can be seen in Figures 3 - 5. For convolution, the rVEX is 80% faster compared to the MicroBlaze for the smallest input size and a factor 3.2 times faster for the largest input size. For greyscale conversion, the rVEX is faster with

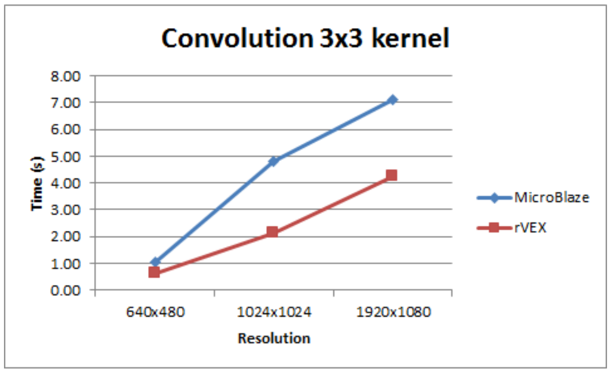


Fig. 4: Execution time for convolution of a 3x3 kernel.

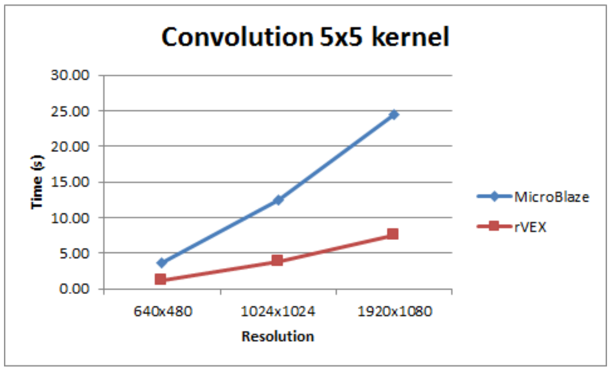


Fig. 5: Execution time for convolution of a 5x5 kernel.

factors of 2.3 to 3 times.

With the rVEX being a 4-issue processor at 75 MHz and the MicroBlaze a single-issue processor at 150 MHz, the expected difference in performance between the cores is a factor of 2. However, inspection of the assembly code shows that the compiler uses loop unrolling to decrease branching delays and, more importantly, fill the issue slots so the VLIW can fully utilize all of its resources. As the processor is calculating values for multiple adjacent pixels at the same time, it is able to keep more input pixel values in registers and/or make use of better cache locality compared to the MicroBlaze that needs to reload input pixel values for every inner loop iteration. This effect will continue to have impact as long as the number of available registers is sufficient, as is shown by the growing performance difference between the cores as the problem size increases.

VI. Conclusions

In this paper, we have shown that the rVEX softcore processor can achieve remarkably better performance compared to the industry-standard Xilinx MicroBlaze (up to a factor of 3.2 times faster) on image processing applications. In order to be able to use the rVEX as a competitive platform capable of accelerating industrial grade image processing applications, a number of improvements can be implemented. These improvements are needed in the following areas:

- How to efficiently stream data to and from the FPGA
- Designing a fast memory hierarchy on the FPGA or a means to efficiently stream data between different cores (each core might perform a certain step in the image processing pipeline, or each core will have a certain part of the image assigned and will perform all the steps)
- Investigating instruction-set extensions that can perform or speed up common image processing operations

VII. Acknowledgment

This work was supported by the European Commission in the context of the ARTEMIS project ALMARVI (project #621439).

References

[1] L. Russo, E. Pedrino, E. Kato, and V. Roda, “Image convolution processing: A GPU versus FPGA comparison,” in *Programmable Logic (SPL), 2012 VIII Southern Conference on*, March 2012, pp. 1–6.
[2] A. Putnam, A. Caulfield, E. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. Gopal, J. Gray, M. Haselman, S. Hauck, S. Heil, A. Hormati, J.-Y. Kim, S. Lanka, J. Larus, E. Peterson, S. Pope, A. Smith, J. Thong, P. Xiao, and D. Burger, “A reconfigurable fabric for accelerating large-scale datacenter services,” in *Computer Architecture (ISCA), 2014 ACM/IEEE 41st International Symposium on*, June 2014, pp. 13–24.
[3] P. Wang, J. McAllister, and Y. Wu, “Soft-core stream processing on FPGA: An FFT case study,” in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, May 2013, pp. 2756–2760.
[4] S. Williams, A. Waterman, and D. Patterson, “Roofline: an insightful visual performance model for multicore architectures,” *Communications of the ACM*, vol. 52, no. 4, pp. 65–76, 2009.
[5] C. Iseli and E. Sanchez, “Spyder: A reconfigurable VLIW processor using FPGAs,” in *FPGAs for Custom Computing Machines, 1993. Proceedings. IEEE Workshop on*. IEEE, 1993, pp. 17–24.
[6] A. K. Jones, R. Hoare, D. Kusic, J. Fazekas, and J. Foster, “An FPGA-based VLIW processor with custom hardware execution,” in *Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays*. ACM, 2005, pp. 107–117.
[7] http://www.trimaran.org.
[8] V. Brost, F. Yang, and M. Paindavoine, “A modular VLIW processor,” in *Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on*. IEEE, 2007, pp. 3968–3971.
[9] M. A. Saghir, M. El-Majzoub, and P. Akl, “Customizing the datapath and ISA of soft VLIW processors,” in *High Performance Embedded Architectures and Compilers*. Springer, 2007, pp. 276–290.
[10] A. Lodi, M. Toma, F. Campi, A. Cappelli, R. Canegallo, and R. Guerrieri, “A VLIW processor with reconfigurable instruction set for embedded applications,” *Solid-State Circuits, IEEE Journal of*, vol. 38, no. 11, pp. 1876–1886, 2003.
[11] S. Wong, T. van As, and G. Brown, “ρ-VEX: A Reconfigurable and Extensible Softcore VLIW Processor,” in *International Conference on Field-Programmable Technology (ICFPT)*, December 2008.
[12] F. Anjam, M. Nadeem, and S. Wong, “Targeting code diversity with run-time adjustable issue-slots in a chip multiprocessor,” in *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, March 2011, pp. 1–6.
[13] ——, “A VLIW softcore processor with dynamically adjustable issue-slots,” in *Field-Programmable Technology (FPT), 2010 International Conference on*. IEEE, 2010, pp. 393–398.
[14] 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.
[15] The HP VEX toolchain, http://www.hpl.hp.com/downloads/vex/.
[16] http://www.gaisler.com/index.php/products/ipcores/soclibrary.
[17] Images created by Michael Plotke, licensed CC BY-SA 3.0. For more information see https://en.wikipedia.org/w/index.php?title=Kernel_%28image_processing%29&oldid=663700413.