

VASILE: A Reconfigurable Vector Architecture for Instruction Level Frequency Scaling

Lucian Petrica

Politehnica University of Bucharest
Bucharest, Romania
Email: lucian.petrica@arh.pub.ro

Valeriu Codreanu

University of Groningen
Groningen, The Netherlands
Email: v.b.codreanu@rug.nl

Sorin Cotofana

Delft University of Technology
Delft, The Netherlands
Email: s.d.cotofana@tudelft.nl

Abstract—Coarse-grained dynamic frequency scaling has been extensively utilised in embedded (multiprocessor) platforms to achieve energy reduction and by implication to extend the autonomy and battery lifetime. In this paper we propose to make use of fine-grained frequency scaling, i.e., adjust the frequency at instruction level, to increase the instruction throughput of a FPGA implemented Vector Processor (VP). We introduce a VP architectural template and an associated design methodology that enables the creation of application requirements tailored VP instances. For each instance, the data-path delays of individual instructions are optimized separately, guided by profiling data corresponding to the target application class, maximizing the performance of frequently utilised instructions to the detriment of those which are less often executed. In this way instructions are divided into clock frequency classes according to their data-path delay and at run time the clock frequency is scaled to the value required by the class of the to be executed instruction. During the application execution different VP instances are dynamically configured in FPGA in order to create the most appropriate hardware support for optimizing the application performance in terms of throughput without increasing power consumption, and therefore reducing energy. As operating frequency changes induce a certain time penalty, which may potentially diminish the actual performance gain, the application code is optimised during the compilation in order to reduce the number of run-time clock switches via, e.g., loop tiling, instruction clustering. We evaluate the effectiveness of the proposed approach on several computational kernels used in image processing applications, i.e., sum of absolute differences, sum of squared differences, and Gaussian filtering. Our results indicate that an average instruction throughput increase of 20%, and a 15% energy consumption reduction are achieved due to the utilisation of run-time reconfiguration and fine-grained frequency scaling.

I. INTRODUCTION

Vector processors (VPs) belong to the Single Instruction Multiple Data (SIMD) class of parallel computers in Flynn's taxonomy [1]. A typical VP consists of an array of Processing Elements (PEs) and a central control unit, which fetches and dispatches instructions and data to PEs. VPs are well suited to applications which involve dense linear algebra, e.g., computer vision [2]. Two of the most important computational kernels in computer vision today are: (i) convolution, which operates on large arrays of pixel data, and (ii) similarity search that is heavily utilized in object recognition and relies on the computation of distance norms between large vectors. As the importance of computer vision applications increases, it is of interest to also increase the VP speed and reduce their energy consumption.

Vector processing applications benefit from a large PE number [3] and, when implemented in FPGA technology, the VP design goal is to fit as many PEs, operating at the highest attainable clock frequency, into the available FPGA device. However, the non-uniform on-die distribution of FPGA embedded multipliers and memory blocks [4] implies that not all PEs can be placed in the vicinity of the embedded resources they utilize. The long signal routes required to reach these resources make it difficult to balance the slack of paths through arithmetic circuitry. Since the maximum operating frequency is given by the longest signal delay, one long route may ultimately cost many MHz top speed lost. This issue can be mitigated by means of multi-cycle paths or route pipelining, but such an approach is device-specific and results in Instruction Set Architecture (ISA) modifications, which are limiting the design portability and require code recompilation.

In this paper we introduce VASILE (VVector Architecture for Scaling at Instruction LEvel), a vector processor architecture designed to overcome the limitations of FPGA routing delays under high logic congestion. It makes use of fine-grained frequency scaling to dynamically adapt the clock frequency to the delay of the currently executed instruction. In this way, instructions always execute at the fastest clock speed permitted by the signal delay through their associated data-path. As a matter of fact, fine-grained frequency scaling mimics asynchronous operation but without suffering from asynchronous design traditional problems. Following the principle of making the common case the fastest, we also introduce a hardware-software co-design methodology, whereby the synthesis tool optimisation effort is focused on the most commonly executed instruction data-paths, resulting in a VP implementation optimised for the target application. Conversely, at compile time, the application is optimized through loop unrolling, loop tiling, and instruction reordering in order to create clusters of instructions of the same class, which maximizes the performance gain by reducing the number of required clock frequency switches.

To evaluate the effectiveness of the proposed approach, we optimised VASILE VPs for the Sum of Absolute Differences (SAD), Sum of Squared Differences (SSD), and Gaussian convolution. We implemented the VPs on the Zedboard [5] Zynq 7020 development platform and evaluated the performance and energy effectiveness of our implementations on the mentioned algorithms, as well as on an image processing workload which makes use of them. Our results indicate that on average we achieve a 20% performance increase, with as much as 38%

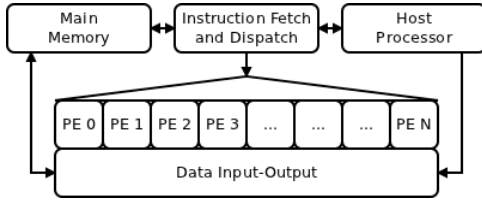


Fig. 1. Vector Processor Organization.

performance increase for the Sum of Absolute Differences algorithm. The performance improvement allows for a quicker workload execution, which enables the use of aggressive power management techniques, i.e., clock and power gating, reducing the energy consumption by 15% overall.

The rest of the paper is structured as follows. Section II provides background information on FPGA vector processor architectures. Section III introduces the VASILE vector architecture and design methodology. Section IV presents a VASILE implementation and its performance evaluations on an object recognition workflow, while Section V provides some concluding remarks.

II. FPGA VECTOR PROCESSORS

Vector Processors (VPs) are an established research area [6] and have the potential to provide significant speedup for multimedia applications when compared to scalar processors [7]. Figure 1 presents a typical VP structure, as implemented in the Connex Array [7]. A central fetch unit reads instructions from main memory and dispatches them to all PEs simultaneously. This removes the need for control logic in the PE itself, which only contains logic related to the actual computation, i.e., register file, arithmetic, local memory. Data are transferred to the PEs through an Input-Output (IO) controller, which has Direct Memory Access (DMA) capability and is programmed by the host processor.

Recently there has been interest in FPGA implemented vector coprocessors. The VENICE processor [8] provides vector extensions to an FPGA soft processor, claiming up to 30% performance increase when compared to an Intel Core2 processor, for matrix multiplication applications. Work on the VESPA [9] processor suggests that application performance scales with the number of vector lanes, making it advantageous to always provide the maximum number of vector lanes, which fit into the target FPGA device. However, the clock speed loss caused by FPGA architecture and congestion is not discussed.

The Xilinx [4] and Altera [10] FPGA fabric architectures consist of alternating columns of logic, memory, and embedded multipliers. FPGA parts vary in the number of columns, the particular mix of columns, and their placement relative to each-other. Figure 2 illustrates the layout of the Zynq FPGA silicon die, extracted from the Xilinx PlanAhead software. The device presented in the figure is the mid-range Zynq 7020. The maximum distance between embedded multiplier columns is roughly half the length of the die, which means that signal routes from a logic element may travel up to a quarter of the die length to reach a multiplier. Memory columns are placed closer together, consequently the maximum route length from a logic element to a memory element is about 15% of the die length.

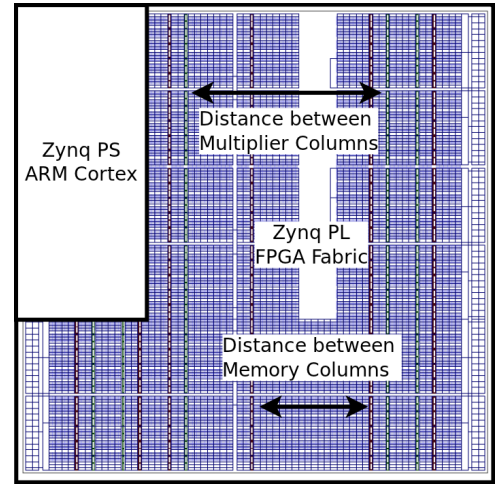


Fig. 2. Zynq Die Floorplan.

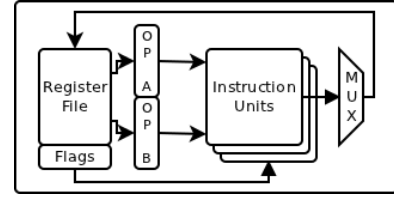


Fig. 3. VASILE Processing Element Organization.

For small designs, the FPGA synthesis tool can place the logic close to the required multipliers and memories. However, when a design approaches the FPGA maximum capacity, some of the logic cannot be placed close to these resources, which leads to performance degradation. The following section introduces an architectural solution to this problem.

III. VASILE

VASILE (VVector Architecture for Scaling at Instruction Level) is a vector processor architecture designed to overcome the limitations of FPGA routing delays under high logic congestion. To this end, VASILE segregates instructions based on the nature of their FPGA based implementation, in order to better control data-path delays. The VASILE PE organization is presented in Figure 3. The PE core logic consists of register file, operand registers, forwarding logic, and a write-back multiplexer. The outputs of the operand registers are distributed to all Instruction Units (IUs), and the IU results are connected to the core logic for write-back. Each IU corresponds to a class of instructions, which utilizes the same FPGA embedded resource(s). Table I lists all VASILE instruction classes along with the FPGA resources they utilize. Algebra instructions, like sum or difference, logic operations and logical shifts are performed in FPGA logic, i.e., LUTs. Multiplication and related operations, like multiply-accumulate, are implemented using embedded multipliers. Loads and stores require access to the embedded memory, while inter-PE communication instructions utilize the FPGA routing network to pass signals from one PE to the other. Inter-PE wires can vary in length, depending on the interconnect topology and logic placement in the FPGA.

VASILE's core assumption is that delay through the IUs

TABLE I. INSTRUCTION CLASSES

Instruction Class	FPGA Resource
Algebra	Logic (LUTs)
Multiplication	Embedded Multiplier
Memory	Embedded RAM
Communication	Routing

may differ significantly. To exploit this feature, the architecture includes a Frequency Selection Unit (FSU), which comprises several clock sources, a clock multiplexer, and associated logic. Each clock source corresponds to one or more instruction classes. During the decode phase, the FSU inspects the instruction stream for clock switch instructions and selects the desired clock source. A configurable delay line ensures that the decoded instructions do not reach the PEs before the clock multiplexer has switched to the proper source for the respective instruction. The switch time is device-dependent and may require a relatively lengthy period of time to complete. At the software platform level, the VASILE architecture relies on a specialized compiler, which tiles loops and clusters instructions together into blocks belonging to the same instruction class, in order to minimise the number of required clock switches. The block size depends on the application nature and the PE register file size.

The processor architecture is complemented by a specialised design methodology for both the VASILE hardware and the software application running on top of it. The implementation starts with the design of the software application. An initial profiling step identifies the most common instructions. The IUs of the most common instructions are optimised for reduced delay. An instruction performance profile is generated from the IU achieved delays and the clock switch time is calculated for each possible frequency transition. This timing information is used by the code compiler to properly cluster the code and insert clock switch instructions. The compiler may choose not to switch the clock if the switch does not result in performance increase. An FSU configuration, which includes the number and frequency of the clock sources, is generated from the final code.

We note that the application-driven nature of the design methodology may yield poor results when the target application comprises several computational kernels with different characteristics. In such a situation, optimising for one kernel may result in poor performance for the other. To handle this situation, two or more VP instances are created and time-multiplexed into FPGA fabric at run-time, if the gain in performance is sufficient to offset the reconfiguration time, which is device-dependent.

IV. PERFORMANCE EVALUATION

Experimental evaluations of the proposed technique were performed on the Xilinx Zynq 7020 heterogeneous processing platform, presented in Figure 2. The Programmable System (PS) consists of a dual-core ARM Cortex-A9 CPU, a DDR memory controller, and several peripheral controllers. The Programmable Logic (PL) consists of FPGA fabric. The PL and PS are powered independently to allow for PL power gating. The ARM processors support several low-power modes,

```

1: procedure SSD( $N, M$ )
2:   for  $i = 1 \rightarrow N$  do
3:     for  $j = 1 \rightarrow M$  do
4:        $R[1] = LS[i]$ 
5:        $R[2] = LS[N + j]$ 
6:        $R[3] = R[1] - R[2]$ 
7:        $R[3] = R[3] * R[3]$ 
8:        $REDUCE\ R[3]$ 
9:     end for
10:  end for
11: end procedure

```

Fig. 4. The SSD Algorithm.

```

1: procedure SSD( $N, M$ )
2:   for  $i = 1 \rightarrow N$  do
3:     for  $j = 1 \rightarrow M/30$  do
4:        $R[0] = LS[i]$ 
5:       for  $k = 0 \rightarrow 29$  do
6:          $R[1 + k] = LS[N + 30 * j + k]$ 
7:       end for
8:       for  $k = 0 \rightarrow 29$  do
9:          $R[1 + k] = R[1 + k] - R[0]$ 
10:      end for
11:      for  $k = 0 \rightarrow 29$  do
12:         $R[1 + k] = R[1 + k] * R[1 + k]$ 
13:      end for
14:      for  $k = 0 \rightarrow 29$  do
15:         $REDUCE\ R[1 + k]$ 
16:      end for
17:    end for
18:  end for
19: end procedure

```

Fig. 5. The Tiled SSD Algorithm.

including deep sleep mode, which gates the clock to most PS peripherals and interconnects [11].

We selected three computer vision algorithms: SSD, SAD, and Gaussian convolution, and added an object recognition workload based on Scale-Invariant Feature Transform (SIFT) keypoint matching [12], [13] in order to integrate the three algorithms into a real-world scenario. The vector processor is responsible for performing (i) the SIFT Gaussian convolution algorithm on high-definition images, and (ii) SIFT descriptors SAD based matching against a known objects database containing 10,000 SIFT keypoints. The recognition workload is repeated every second, and the system goes into low-power mode between recognition events. SIFT typically extracts between 100 and 2000 keypoints from an image, depending on object clutter and other factors.

Clock switching was implemented through the use of the BUFGMUX_CTRL primitive. The clock switch occurs within a precisely defined time period, expressed in Equation 1. T_1 and T_2 are the periods of the two involved clock signals, and the switching time is at most three slow clock periods, depending on their phase alignment.

$$T_{Switch} \leq 3 * \max(T_1, T_2) \quad (1)$$

TABLE II. IU BASELINE AND MAXIMUM FREQUENCIES

Instruction Class	$F_{Baseline}$ [Mhz]	F_{Max} [Mhz]	Density [%]
Algebra	125	163	53
Multiplication	125	117	33
Memory	125	160	11
Communication	125	160	3

TABLE III. INSTRUCTION TILING AND CLOCK SWITCHING

Algorithm	Tile Size	Loop Clock Switches	Speedup
SAD	-	0	1.28
SSD	30	2	1.10
Convolution	10	4	0.90

We manually applied the design methodology outlined in Section III, using Xilinx ISE 14.2 software tools, and were able to fit 128 PEs into the target FPGA device. Table II presents the maximum frequencies attainable for the instruction classes, and the average class density in the target applications. As the table indicates, only the multiplication class differs significantly in performance, which is expected given the large distances between multiplier columns on the target device. Because of the small difference in top frequency for the Algebra, Memory and Communication classes, we re-partitioned the instructions into two classes only, i.e., Multiply, with a top frequency of 117 MHz, and Non-Multiply with a top frequency of 160 MHz.

We also manually performed tiling optimisations where required, as illustrated in Figure 5 for SSD. The SSD inner loop is restructured into several loops, which are then unrolled to form batches of 30 consecutive identical instructions. The tiling optimisations results are presented in Table III, along with speedup figures versus the baseline frequency of 125 Mhz. Outer loops have been preserved, and the table shows how many clock frequency switches are required to execute each iteration of the outer loop. SAD does not require any clock switch because it does not utilize multiplication instructions, therefore tiling was not performed. Convolution does not benefit from the clock switching because the maximum tile size is smaller and data dependencies require the clock to switch too often.

Power dissipation was calculated by the Xilinx Power Analyzer from the final FPGA netlist, and we assumed that power gating infrastructure exists off-chip, to enable PL power gating. Table IV lists estimated power dissipation and energy consumption for the object recognition workload in three scenarios: (i) baseline implementation, (ii) Clock Switching (CS) implementation, and (iii) Reconfigurable Clock Switching (RCS), whereby convolution is performed on the balanced implementation and keypoint matching on the clock switching implementation. Both the CS and RCS scenarios provide better instruction throughput and lower energy consumption than the baseline, with RCS being marginally better.

V. CONCLUSION

We have presented VASILE, a vector processor architecture designed for pseudo-asynchronous operation in order to overcome some of the limitations in modern FPGAs. The proposed architecture segregates instructions with similar delay into classes and frequency is scaled according the delay of

TABLE IV. OBJECT RECOGNITION PERFORMANCE

Implementation	Baseline	CS	RCS
$T_{convolution}$ [ms]	180	198	180
$T_{matching}$ [ms]	560	437	437
T_{total} [ms]	740	635	617
FPGA Power [mW]	600	636	636
Zynq Power [mW]	1620	1656	1656
Energy Consumption [mJ]	1199	1053	1021
Speedup	1	1.16	1.20
Relative Energy	1	0.88	0.85

the executed instruction. In this manner, instructions execute at the rate permitted by their datapath delay, much like asynchronous circuits. An associated design methodology was introduced to guide the optimisation effort towards the most commonly utilised instructions, to the detriment of those less common. A code compilation strategy was also introduced, which utilises loop tiling and instruction clustering to form instruction batches, in order to minimise the number of frequency changes and the corresponding time penalty. Experimental evaluations on a Xilinx Zynq 7020 device demonstrate that the proposed architecture provides up to 28% performance increase for image processing workloads. The performance increase enables the device to spend more time in low-power modes, thereby also results in a 15% energy consumption reduction.

REFERENCES

- [1] M. J. Flynn, "Some computer organizations and their effectiveness," *IEEE Transactions on Computers*, vol. 100, no. 9, pp. 948–960, 1972.
- [2] D. Kim, K. Kim, J.-Y. Kim, S. Lee, and H.-J. Yoo, "An 81.6 GOPS object recognition processor based on NoC and visual image processing memory," in *Custom Integrated Circuits Conference*. IEEE, 2007, pp. 443–446.
- [3] P. Yiannacouras, J. G. Steffan, and J. Rose, "Fine-grain performance scaling of soft vector processors," in *International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*, Grenoble, France, October 2009.
- [4] (2003, December) The Xilinx ASMBL architecture. [Online]. Available: http://www.xilinx.com/company/press/kits/asmb/asmb_arch_pres.pdf
- [5] (2013, April) Zedboard development platform. [Online]. Available: <http://www.zedboard.org>
- [6] K. Asanovic, "Vector microprocessors," Ph.D. dissertation, University of California, 1998.
- [7] G. Stefan, "The CA1024: A massively parallel processor for cost-effective HDTV," in *Spring Processor Forum: Power-Efficient Design*, 2006, pp. 15–17.
- [8] A. Severance and G. Lemieux, "VENICE: A compact vector processor for FPGA applications," in *International Conference on Field-Programmable Technology*. IEEE, 2012, pp. 261–268.
- [9] P. Yiannacouras, J. G. Steffan, and J. Rose, "VESPA: Portable, scalable, and flexible FPGA-based vector processors," in *International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*, Atlanta, USA, October 2008.
- [10] (2005, July) The Altera Stratix architecture. [Online]. Available: http://www.altera.com/literature/hb/stx/ch_2_vol_1.pdf
- [11] (2013, March) Xilinx zynq technical reference manual. [Online]. Available: http://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf
- [12] L. Shapiro and G. C. Stockman, *Computer Vision*. 2001. Prentice Hall, 2001.
- [13] D. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.