TTA-SIMD Soft Core Processors

Kati Tervo*, Samawat Malik, Topi Leppänen and Pekka Jääskeläinen†
Tampere University, Finland
{kati.tervo,malik.samawat,topi.leppanen,pekka.jaaskelainen}@tuni.fi
*https://orcid.org/0000-0001-6050-7100 †https://orcid.org/0000-0001-5707-8544

***Abstract*—Soft processors are an important tool in the Field Programmable Gate Array (FPGA) designer's toolkit, and their Single Instruction Multiple Data (SIMD) organizations are an efficient means to utilize the parallelism of FPGAs. However, the state-of-the-art SIMD processors are hindered by the additional logic complexity resulting from dynamic features. By minimizing such constructs, it is possible to design soft processors that are efficient but still flexible enough to operate within an application domain.**

To this end, we propose a family of instruction set programmable multi-issue wide SIMD soft cores. The template is based on a highly static Transport Triggered Architecture (TTA) and a design time customizable shuffle unit to minimize inefficient dynamic features while remaining compiler programmable. The cores are evaluated on the PYNQ-Z1 board against the ARM A9 hard processor system with NEON vector extensions. The proposed cores reach up to 2.4x performance improvement over the ARM, can fit up to 1024 bit wide SIMD units onto the relatively small FPGA, while still operating at above 100 MHz. The scalability of TTA enables state of the art vector widths. The multicore scalability of the template is preliminarily tested with a 14-core design on a XCZU9EG FPGA customized for real-time convolutional neural net inference.

***Index Terms*—Field-Programmable Gate Array (FPGA), Transport-Triggered Architecture (TTA), Single Instruction Multiple Data (SIMD)**

I. Introduction

Soft processors are an important tool in designing *Field-Programmable Gate Array* (FPGA) systems. They present a familiar level of abstraction for software-oriented engineers, and can be used without significant hardware design knowledge. Integration of a predesigned soft processor into an FPGA design is simple, and is further simplified by FPGA *Systems-on-Chip* (SoCs), where a fixed hard processor system can be used to orchestrate execution soft processors on the FPGA fabric.

Similarly, *High-Level Synthesis* (HLS) tools also lower the barrier to entry to logic design for software engineers, but do not remove it, as hardware design experience is still needed for tuning the HLS implementation for good results [2]. Nevertheless, with the growing complexity of logic design projects, additional design abstraction away from traditional register-transfer level design is increasingly needed. Additionally, lowering the initial cost of logic designs can help offload to FPGA devices easily available in a data center context. Pre-synthesized soft processor overlays are a simple way to take advantage of available FPGA resources.

This work is part of the FitOptiVis project [1] funded by the ECSEL Joint Undertaking under grant number H2020-ECSEL-2017-2-783162.

The tradeoff in the increased flexibility and ease of design is the overhead introduced by the additional programmability provided by the overlay. The effect of this is amplified in FPGA designs due to the relative inefficiency of control logic implementation compared to hardened arithmetic blocks [3].

For the same reason, it can be very beneficial to customize the architecture for an application domain, picking simpler static operations over complex dynamic ones. Furthermore, aspects of the processor that do not commonly change during an application, such as the required arithmetic precision, can be fixed at design time. If these need to be changed between applications, the device can be partially or fully reconfigured.

In the past, the statically scheduled *Transport-Triggered Architectures* (TTAs) have been shown to deliver instruction-level parallelism (ILP) for relatively small soft processors. The programmer-exposed datapath of TTA can support parallel function units with minimal added complexity [4]. However, to the best of our knowledge, scaling TTA soft processors for high logic utilization and throughput by occupying all the resources available on the FPGA device has not yet been evaluated.

In this paper, we propose a high-performance TTA-SIMD soft processor architecture template, which utilizes data parallelism by a *Single Instruction Multiple Data* (SIMD) organization on top of the ILP capabilities by means of vector function units. We make the following contributions:

- A family of TTA-SIMD soft processors with vector function units (FU) of varying SIMD element count and arithmetic precision, designed to scale up to a very wide SIMD width use scenarios, while simultaneously supporting instruction-level parallelism.
- Evaluation of the processor family with OpenCL benchmarks, comparing against the ARM A9 hard processor in the used SoC.
- A soft core design case study of a *Convolutional Neural Network* (CNN) accelerator as a case study of a domain-specific multicore design using the SIMD TTA template.

The rest of the paper is organized as follows. In Section II, we present an overview of the TTA processor design paradigm with a focus on FPGA implementations. This is followed by Section III, introducing the scalable TTA-SIMD soft core template. Next, in Section IV, the synthesis results and the performance of the proposed architectures is evaluated. In Section V, we present the CNN accelerator based on the TTA-SIMD approach. This is followed by an overview of previous work as well as our conclusions from this work.

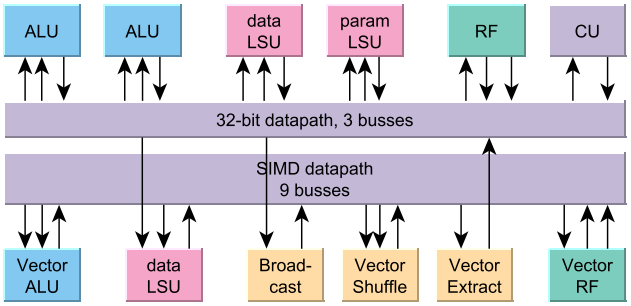

Fig. 1: Simplified view of the wide-SIMD TTA template.

II. Transport-Triggered Architectures

TTA processors have been studied extensively as application or domain specific co-processors, but most of the focus has been on ASIC implementations. In early studies [5], the TTA paradigm was proposed as an improvement on *Very Long Instruction Word* (VLIW) processors that reduced Register File (RF) and bypass network complexity.

Like the parallel operations in VLIW, TTA instructions are composed of parallel *moves*, specifying a data transport typically from one port of a FU or RF to another. TTA moves are fine grained, each transferring a single operand. Operation execution can be started once the other operands are in place with a move to a specified trigger port.

Since the operands can be transferred directly from FU ports, the RF does not necessarily need to be accessed, as in traditional write-back bypassing but without additional bypassing logic. Furthermore, if every RF read of a value can be eliminated, the value doesn't need to be written to the RF at all. This is prohibitively difficult to implement in traditional architectures, as it would require *dynamically* checking if the result register is read between the operation and the next write to it [6].

Despite the similarities to VLIW, the RF simplification generalizes to also simple scalar cores. A comparison against MicroBlaze showed that TTAs can add ILP with little overhead, in particular thanks to the simple register file [4]. However, the very explicit instruction format leads to long instruction words. While the required instruction memory size tends to be larger, TTA binaries had fewer total instructions than the MicroBlaze processors thanks to increased ILP.

Overall, TTAs have streamlined instruction decoding thanks to the lack of dynamic control features, such as hardware bypassing, and can use simpler RFs with greater efficiency. This is of particular benefit on FPGAs, as their memory primitives are limited, with at most two write ports. Increasing port count requires the data to be duplicated in more than one memory primitive [7], [8].

III. TTA-SIMD Soft Core Template

The original TTA approach provides scalable support for ILP. However, for high throughput, it is essential to support data level parallelism (DLP) as well. In this section, we describe a soft core template with wide SIMD arithmetic, shown in Figure 1. The template was designed using an extended version of the TTA-based Co-design Environment (TCE) [9], with additional features for SIMD architectures [10]. TCE includes a compiler and a cycle-accurate simulator that retarget automatically to the designed architectures, and can be used to generate a synthesizable RTL implementations of the designs.

A. Datapath

The datapath of the TTA-SIMD template consists of two sets of busses; one for the scalar FUs used primarily for address calculation and branching, and another for SIMD FUs, typically responsible for most of the number crunching. The busses are not fully connected: redundant and rarely-needed connections have been removed to reduce the logic utilization.

The two halves share an instruction word, and execute in lockstep. To ensure that address calculation does not slow down the SIMD datapath, there are two identical scalar Arithmetic-Logic Units (ALUs). The operation set for these ALUs is minimal: basic arithmetic, comparison and logic operations, and a 4-cycle barrel shifter. The vector ALU is left customizable in the template: Its operations, operation latencies, and the SIMD width are selected per application.

B. SIMD Interlane Communication

A common source of complexity in SIMD processors is the communication between the SIMD lanes. In the proposed design, this connectivity has been reduced to its absolute minimum. The only direct communication between the SIMD and scalar busses is through two FUs, one for a scalar-to-SIMD broadcast operation and another for SIMD-to-scalar element extraction operation. To reduce the implementation complexity further, a third FU contains a set of pre-selected shuffle operations with constant indices. This is a major simplification over a full shuffle crossbar.

C. Memory System

Each core has a set of static local memories in the template with two *disjoint* address spaces: `data` for the OpenCL buffers and command queue, and `param`, for the stack. `data` can be accessed either by scalar or aligned SIMD accesses, while `param` is solely scalar. To simplify the LSU implementation, the memory operations to the `data` address space were divided into two LSUs, one for aligned vector accesses, and another for scalar accesses. The scalar LSU shares its port with the AXI bus to the host processor.

IV. Evaluation

We created a set of benchmarks to show the programmability of design instances targeting an application domain to evaluate the presented architecture. The benchmark set consists of vector addition, polynomial, matrix multiplication, 64-tap FIR, 5-by-5 convolution filter, and Sobel edge detection.

For these benchmarks, we selected four vector ALU operations: addition, subtraction, multiplication, multiply-add and left shift, and four shuffle operations with static indices.

A total of 16 machines were made by changing SIMD element width and count. The 4x8b machine and the 128x32b machine were discarded, as they were too narrow and wide, respectively, to be supported by the TCE compiler.

The tests were executed with the *pocl* OpenCL runtime [11]. In addition to supporting ARM, pocl has a driver for TTAs designed with TCE. For more accurate performance measurements, the benchmarks were executed on TTA processors synthesized for the FPGA. The compilation and the runtime environment of the TTA processors and the ARM comparison were identical, save for the device drivers in pocl.

Hand-vectorized benchmarks were implemented as OpenCL C kernels using explicit vector datatypes with parametrized arithmetic precision and vector width. The benchmarks were ran on a cycle-accurate simulator. As expected, the cycle count roughly halves every time the SIMD lane count is doubled.

For vectors of up to 16 elements, shuffles were implemented with standard OpenCL C swizzle notation. However, OpenCL C supports vector datatypes of only up to 16 elements [12]. Wider shuffles were implemented with a Clang compiler intrinsic `__builtin_shufflevector`. This custom extension aside, the benchmark code was ensured to be portable, and free of architecture-specific intrinsic calls.

A. Synthesis Results

The subset of processors that could fit on a Zynq Z7020 FPGA were synthesized, prioritizing clock frequency. Despite the relatively small size of the device, SIMD lane widths of up to 1024 bits could be realized. The results can be seen in Figures 2 and 3. In addition to LUTs, the processors consumed 76 RAM blocks, 3 DSP blocks for every 32-bit multiplier and 1 DSP block for every 16- or 8-bit multiplier. While the maximum clock frequency decays with wider SIMD lanes, we consider the attained frequencies to be good for the device in question.

In order to analyze the resource utilization on a component-by-component basis, the machines with 8-bit arithmetic were synthesized out-of-context at a 100 MHz clock frequency, while keeping the RTL hierarchy intact. The lower frequency is to make sure all machines reach timing closure despite the lack of cross-hierarchy optimizations. Additionally, an FU with a naive implementation of a generic shuffle operation was synthesized for comparison.

As can be seen in Figure 4, the vector FUs take up the most of the LUTs on wide machines, followed by the interconnect. The interconnect, data memory, and vector FUs grow linearly with the lane count, with the exception of the shuffle FU, of which growth fits better to a quadratic function with a small (< 0.1) second-degree coefficient. The generic shuffle unit fits a similarly quadratic function, but with a more significant (> 3.9) second-degree coefficient. The 128x8b constrained shuffle FU used 2561 LUTs, less than half of the 32x8b generic shuffle, while a 128x8b generic shuffle used over 72782, 136 percent of the available LUTs of the Zynq Z7020 device.

The other components, divided here to processor and memory interface, are roughly constant across the architectures. Included in this group are the instruction fetch and decode stages, which require less than 470 LUTs, or 9 percent of the smallest architecture and 1.4 percent of the largest. Thus, as expected, with wider SIMD widths, the relative programmability overhead decreases.

B. Comparison with ARM

A subset of TTA-SIMD soft core instances selected for synthesis was used to execute the benchmarks on the FPGA SoC. For comparison, the same benchmarks were run on the ARM Cortex-A9 processor with the NEON vector extension also present on the SoC, clocked at 650 MHz. Both TTA and ARM execution was single-core, and kernel execution time was measured, excluding buffer transfers, and averaged from 32 subsequent executions of the kernel for each benchmark and architecture. On ARM, the benchmarks were ran for all vector widths, and the best average runtime was selected as the result for a given arithmetic precision. While the longer instruction word of the TTA machines makes the binaries larger, the TTA binaries have fewer instructions due to the added parallelism, mitigating the size increase to an average of 163 percent.

Figure 5 shows the performance comparison between the TTA processors and the ARM hard processor system. The 32x32b architecture performs equal or better than ARM for all benchmarks, despite the 6.2x difference in clock frequency, with a 2.4x improvement for the polynomial benchmark. While the architectures with narrower arithmetic fall short of overall improvement, both still show modest improvement on some benchmarks, primarily sobel and polynomial.

C. Register File Efficiency

In order to illustrate the benefit of the TTA programming model on RF logic utilization efficiency, we will compare our architecture with a similar VLIW machine. To reduce the number of FUs and to get a more fair comparison, we will merge the vector broadcast, shuffle, and extract FUs into a single one, leaving us with three vector FUs. To simultaneously serve three FUs with two input operands and one result, the RF needs three write ports and six read ports. To implement this with the 1 write, 3 read port distributed RAM primitives present on the FPGA with the Live Value Table (LVT) approach [7], the design would need six copies of the primitive: doubled to get six read ports, tripled to get three write ports. A two-issue RF would only need four copies.

The distributed RAM utilization of our 1024-bit register files is 1544 LUTs or about 4 percent of the largest cores. All else being equal, five additional copies of the RF would increase the logic utilization of our core by about 20 percent. If we were to merge the vector datapath into two function units, the 2-issue VLIW RF would need three additional copies, leading to a 12 percent increase. Note that additional logic is needed for the LVT itself and the multiplexers used to select the correct values, so the resource usage increase would likely be higher.

V. Real Time Application Case Study

In order to study the scalability limits of the TTA-SIMD approach on larger FPGAs, we designed a TTA soft core

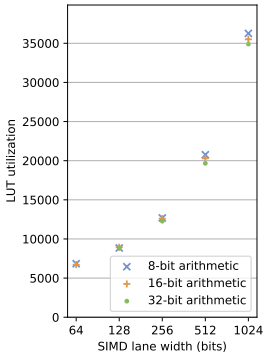

Fig. 2: LUT utilization of the synthesized processors.

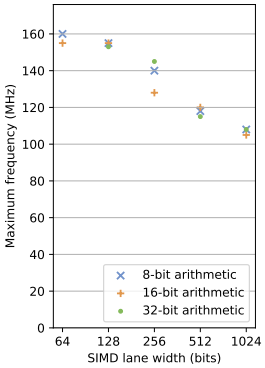

Fig. 3: Maximum clock frequency of the synthesized processors.

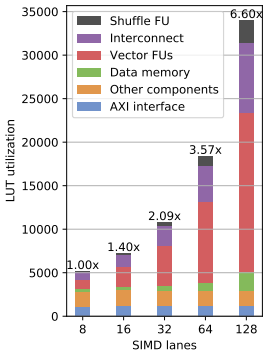

Fig. 4: Utilization breakdown for 100 MHz synthesis of 8-bit machines.

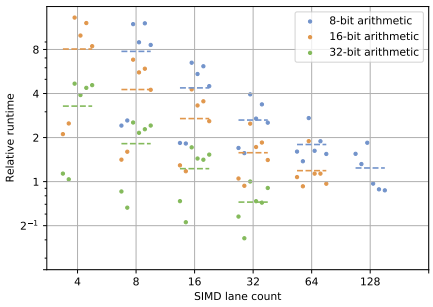

Fig. 5: TTA performance relative to best corresponding ARM result. Dashed lines indicate average of benchmarks for that SIMD width and arithmetic precision.

for a CNN inference application that can utilize the three common forms of computational parallelism: ILP through a multiple-issue architecture, DLP with SIMD FUs, and thread level parallelism by a multicore design.

We also used customized operations and data types, which, together with a high degree of parallelism, is essential in reaching competitive computation performance in FPGA-based implementations.

The single core architecture was designed using TCE, and was based on an earlier design called AivoTTA, which was originally optimized for ASIC implementation [13]. Two object recognition CNN applications implemented in C were augmented with intrinsic calls for the custom operations.

A. Core Architecture

The core vector operation in the design is the vector multiply-add (MADD) FU. It computes 32 convolutions on the same output feature map, enabling weight sharing. The MADD unit multiplies a vector of 32 8-bit pixels and multiplies that with a vector of 16-bit weights or the same weight for all elements. Finally, the result is added to 32-bit inputs for accumulation.

The neural network activation function is applied to the convolution result. To avoid complex activation functions but keep the activation function flexible, we decided to only support linear interpolation in this design. Linear interpolation consists of a multiplication and an addition, and can be performed vector-wise on the multiply-add unit. The interpolation unit produces the coefficients and offsets for the linear convolution. The precalculated points are loaded from memory, so the choice of activation function is defined by software.

This design uses shuffles to shift values from an aligned load to adjacent convolution windows to save on memory bandwidth and LSU complexity, as well as for 2x subsampling. The design uses reduced shuffles rather than a full shuffle crossbar.

In the ASIC design, the vector shift unit was split across four FUs, and the 4-pixel interleave operation was in its own unit. For the FPGA implementation, these were merged together to reduce the number of connections to the interconnect network and to reduce fanout from control signals to the FUs.

In TTAs, RFs are the second lowest level in the software accessible data memory hierarchy; the FU port registers are the lowest. One of the most interesting features of the design case in terms of FPGA implementation is its simplified RFs enabled by the TTA programming model. All the general purpose RFs in the design have only a total of two ports; a single read and a single write port. There are four RFs in total: two 32-bit RFs, one 256-bit and one 1024-bit RF.

B. Results

In order to measure scalability to FPGAs with different sizes and speed grades, we synthesized designs for two different Xilinx Zynq devices: The medium-size Z7020 (speedgrade -1) and a larger Ultrascale+ device, XCZU9EG (-2) with approximately 7 times the LUT count of the Zynq Z7020.

The clock frequency and utilization results can be seen in Table I. In addition to the TTA cores and AXI interconnect IP, the results for the Zynq Z7020 device include IPs for processing the input video stream from HDMI, overlaying the detections to the video stream and outputting the to an HDMI sink. This corresponds to 14 percent of the LUT usage of that design. Even without the video pipeline, the design would not fit a third core, and the impact on maximum frequency is negligible. The TTA processor on the Zynq Z7020 design reached a clock frequency of 145 MHz, while the UltraScale+ design was clocked at 300 MHz.

The theoretical maximum performance (assuming that a MADD operation is issued every cycle) is 18.9 GOPS for the Zynq Z7020 design, and 273 GOPS for the UltraScale+ design. In practice, however, this is limited by the efficiency of compiler scheduling and the latency of the memory accesses. The measured performance from fully executing one frame of CNN inference with the face detection network was 3.7 GOPS and 48.5 GOPS, respectively. The primary bottleneck in the design is the lack of a cache between the TTA processors and the main memory, where the image is being read.

TABLE I: FPGA utilization of the designs. Percentages indicate utilization relative to the total FPGA capacity.

Component	LUT	DSP blocks	RAM blocks
Zynq Z7020, 2 TTA cores			
TTA cores	25811 (48.5 %)	72 (32.7 %)	64 (45.7 %)
Video pipeline	6077 (11.4 %)	0	1 (0.7 %)
AXI interconnect	8501 (16.0 %)	0	0
Total	40408 (76.0 %)	72 (33 %)	65 (46.4 %)
Zynq UltraScale+ XCZU9EG, 14 TTA cores			
TTA cores	165327 (60.3 %)	504 (20 %)	434 (47.6 %)
AXI interconnect	45469 (16.6 %)	0	0
Total	210770 (76.9 %)	504 (20 %)	434 (47.6 %)

C. Real Time Demonstrator

We created two real time FPGA demonstrators with the TTA design. On the Zynq 7020 board, additional components interfaced with an HDMI stream for a standalone demonstrator. On the Zynq Ultrascale+ ZCU102 board, this was handled on Linux running on ARM.

The cores were connected to the hard processor system with master and slave AXI buses, so that the host processor can control the accelerators and the accelerators can read the input from off-chip memory. Each of the accelerators have their own instruction and data memory, as the block RAM consumption did not necessitate sharing memories.

VI. Related Work

Vector processors on FPGAs have been studied in-depth in the literature, primarily as coprocessors to vendor-supplied scalar processors such as NIOS 2 and MicroBlaze. VESPA [14] and VIPERS [15] are similar processors, based on the same instruction set. However, VESPA allows for custom instruction subsetting, while VIPERS has modified its instruction set to fit an FPGA implementation better.

VEGAS [16] and VENICE [17] are presented as an improvement over VESPA and VIPERS, as they replace the RF and memory system with a unified scratchpad, and add fracturable ALUs. Instead of a full shuffle crossbar, they use a pipelined three-cycle Benes network [18].

Like VEGAS and VENICE, the commercial MXP processor [19] has multistage shifting networks between the ALUs and the banked scratchpad memory to handle element alignment. Kapre [20] has compared MXP and ARM execution on the Zynq 7020 SoC. Kapre's results average a 25 percent improvement over the best ARM runtime with 32-bit arithmetic, and a 73 percent improvement over ARM with 8-bit arithmetic. The differences range from an 11 percent decrease of performance for 32-bit 4-tap filter, to a 3.95x increase for 8-bit vector addition. While utilization numbers are not presented, the SIMD width of MXP is limited to 512 bits, at a frequency of 110 MHz.

Another data parallel approach with recent popularity in the literature [21]–[24] is Single Instruction, Multiple Thread (SIMT) architectures, more commonly used in graphics processing units. The most interesting work, performance-wise, is FGPU [25], [26]. It outperforms MXP by a factor of 11 in matrix multiplication in ideal conditions, but the results for other benchmarks are less dramatic, and FIR is equally fast on the two architectures. It can scale up to 8 compute units with a total of 64 processing elements on the Zynq 7045 FPGA, which is more that twice as large as the one used in our work.

Reducing a shuffle crossbar by customizing it for the application has previously been explored at least by Raghavan et al. [27]. They presented six reduced crossbar varieties for different application domains, such as FFT and GSM. Our work shows that the approach can be generalized to other applications.

To the best of our knowledge, the work presented by Kapre is closest work to what we propose, with similar methodologies executed on the same FPGA device. While Kapre sees greater improvements, particularly with 8-bit arithmetic, our scalability optimizations allow us to reach a SIMD width that is twice as wide while maintaining a similar clock frequency to MXP.

While VLIW has been fairly popular as a soft processor template [28]–[31], to our knowledge there have not been FPGA implementations of SIMD VLIW machines or other multiple-issue SIMD soft processors. This may be due to the increased complexity of the RF in operation-triggered multiple-issue machines, easily leading to a 4x increase in RF utilization compared to a single-issue machine. The proposed TTA-based template enables a multiple-issue architecture with RFs that are as simple as with a single-issue machine.

VII. Conclusion

We proposed a SIMD-TTA template for soft processors designed to scale to high SIMD lane counts and to utilize high-degrees of ILP with efficient FPGA utilization. Even in the small FPGA used in our evaluation, we show that architectures based on this template can support up to 1024-bit SIMD lanes, surpassing the closest competition, the MXP soft processor. In comparison with the ARM hard processor system on the board, despite the more than 6x slower clock frequency, our architectures with 32-bit arithmetic reach up to 2.4x speedups with the equal ease of programmability of via OpenCL.

In order to evaluate the scalability of the template to larger FPGAs and to demonstrate a real-time scenario, we presented a case study with application-specific optimizations targeting CNN inference. Scalability was demonstrated on a Zynq UltraScale+ board which could fit 14 cores reaching up to 48.5 GOPS while running a face-detection network.

These results demonstrate that the SIMD-TTA template is useful for offloading applications from the SoC and capable of implementing challenging real-time applications. It simplifies the RF compared to the closest static multiple-issue contender, the traditional VLIW processor, by 12 to 20 percent.

In the future, we plan to explore enhanced soft multicore support on top of OpenCL. Additionally, we believe the the customization effort of TTA-SIMD soft processors can be eased by automated design space exploration, as was preliminary shown with AEx [32] for scalar TTA cores.

REFERENCES

[1] Z. Al-Ars, T. Basten, A. de Beer, M. Geilen, D. Goswami, P. Jääskeläinen, J. Kadlec, M. M. de Alejandro, F. Palumbo, G. Peeren, and et al., "The FitOptiVis ECSEL project: Highly efficient distributed embedded image/video processing in cyber-physical systems," in *Proceedings of the 16th ACM International Conference on Computing Frontiers*, ser. CF '19, 2019, p. 333–338.

[2] S. Lahti, P. Sjövall, J. Vanne, and T. D. Hämäläinen, "Are we there yet? A study on the state of high-level synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 5, pp. 898–911, 2018.

[3] H. Wong, V. Betz, and J. Rose, "Quantifying the gap between FPGA and custom CMOS to aid microarchitectural design," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 10, pp. 2067–2080, 2013.

[4] P. Jääskeläinen, A. Tervo, G. P. Vayá, T. Viitanen, N. Behmann, J. Takala, and H. Blume, "Transport-triggered soft cores," in *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2018, pp. 83–90.

[5] H. Corporaal, *Microprocessor Architectures: from VLIW to TTA*. John Wiley & Sons, Inc., 1997.

[6] J. Hoogerbrugge and H. Corporaal, "Register file port requirements of transport triggered architectures," in *Proceedings of the 27th annual international symposium on Microarchitecture*, 1994, pp. 191–195.

[7] F. Anjam, S. Wong, and F. Nadeem, "A multiported register file with register renaming for configurable softcore VLIW processors," in *2010 International Conference on Field-Programmable Technology*. IEEE, 2010, pp. 403–408.

[8] B.-C. C. Lai and J.-L. Lin, "Efficient designs of multiported memory on FPGA," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 1, pp. 139–150, 2016.

[9] O. Esko, P. Jääskeläinen, P. Huerta, C. de La Lama, J.Takala, and J. Martinez, "Customized exposed datapath soft-core design flow with compiler support," *Int. Conf. on Field Programmable Logic and Applications (FPL)*, 2010.

[10] M. Järvelä, "Vector operation support for transport triggered architectures," Master's thesis, Tampere University of Technology, Finland, 2014.

[11] P. Jääskeläinen, C. S. de La Lama, E. Schnetter, K. Raiskila, J. Takala, and H. Berg, "pocl: A performance-portable OpenCL implementation," *International Journal of Parallel Programming*, vol. 43, no. 5, pp. 752–785, 2015.

[12] *OpenCL 1.2 API Specification*, Khronos OpenCL Working Group, revision 19.

[13] J. IJzerman, T. Viitanen, P. Jääskeläinen, H. Kultala, L. Lehtonen, M. Peemen, H. Corporaal, and J. Takala, "AivoTTA: an energy efficient programmable accelerator for CNN-based object recognition," in *Proceedings of the 18th International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation*, 2018, pp. 28–37.

[14] P. Yiannacouras, J. G. Steffan, and J. Rose, "VESPA: Portable, scalable, and flexible FPGA-based vector processors," in *Proceedings of the 2008 International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, ser. CASES '08. ACM, 2008, pp. 61–70.

[15] J. Yu, C. Eagleston, C. H.-Y. Chou, M. Perreault, and G. Lemieux, "Vector processing as a soft processor accelerator," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 2, no. 2, pp. 1–34, 2009.

[16] C. H. Chou, A. Severance, A. D. Brant, Z. Liu, S. Sant, and G. G. Lemieux, "VEGAS: Soft vector processor with scratchpad memory," in *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays*, 2011, pp. 15–24.

[17] A. Severance and G. Lemieux, "VENICE: A compact vector processor for FPGA applications," in *2012 International Conference on Field-Programmable Technology*. IEEE, 2012, pp. 261–268.

[18] V. E. Beneš, "Optimal rearrangeable multistage connecting networks," *Bell system technical journal*, vol. 43, no. 4, pp. 1641–1656, 1964.

[19] A. Severance and G. G. Lemieux, "Embedded supercomputing in FPGAs with the VectorBlox MXP matrix processor," in *Proceedings of the Ninth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*. IEEE Press, 2013, p. 6.

[20] N. Kapre, "Optimizing soft vector processing in FPGA-based embedded systems," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 9, no. 3, p. 17, 2016.

[21] K. Andryc, M. Merchant, and R. Tessier, "FlexGrip: A soft GPGPU for FPGAs," in *2013 International Conference on Field-Programmable Technology (FPT)*. IEEE, 2013, pp. 230–237.

[22] P. Duarte, P. Tomas, and G. Falcao, "SCRATCH: An end-to-end application-aware soft-GPGPU architecture and trimming tool," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, 2017, pp. 165–177.

[23] J. Kingyens and J. G. Steffan, "A GPU-inspired soft processor for high-throughput acceleration," in *2010 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW)*. IEEE, 2010, pp. 1–8.

[24] S. Collange, "Simty: generalized SIMT execution on RISC-V," in *First Workshop on Computer Architecture Research with RISC-V (CARRV 2017)*, 2017.

[25] M. Al Kadi, B. Janssen, J. Yudi, and M. Huebner, "General-purpose computing with soft GPUs on FPGAs," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 11, no. 1, pp. 1–22, 2018.

[26] M. Al Kadi, B. Janssen, and M. Huebner, "FGPU: An SIMT-architecture for FPGAs," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2016, pp. 254–263.

[27] P. Raghavan, S. Munaga, E. R. Ramos, A. Lambrechts, M. Jayapala, F. Catthoor, and D. Verkest, "A customized cross-bar for data-shuffling in domain-specific SIMD processors," in *International Conference on Architecture of Computing Systems*. Springer, 2007, pp. 57–68.

[28] G. Payá-Vayá, R. Burg, and H. Blume, "Dynamic data-path self-reconfiguration of a VLIW-SIMD soft-processor architecture," in *Workshop on Self-Awareness in Reconfigurable Computing Systems (SRCS)*, 2012, p. 26.

[29] M. Purnaprajna and P. Ienne, "Making wide-issue VLIW processors viable on FPGAs," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 8, no. 4, pp. 1–16, 2012.

[30] Y. Lei, Y. Dou, J. Zhou, and S. Wang, "VPFPAP: A special-purpose VLIW processor for variable-precision floating-point arithmetic," in *2011 21st International Conference on Field Programmable Logic and Applications*. IEEE, 2011, pp. 252–257.

[31] F. Anjam, M. Nadeem, and S. Wong, "A VLIW softcore processor with dynamically adjustable issue-slots," in *2010 International Conference on Field-Programmable Technology*. IEEE, 2010, pp. 393–398.

[32] A. Hirvonen, K. Tervo, H. Kultala, and P. Jääskeläinen, "AEx: Automated customization of exposed datapath soft-cores," in *2019 22nd Euromicro Conference on Digital System Design (DSD)*. IEEE, 2019, pp. 35–42.