Synthetic Aperture Radar Algorithms on Transport Triggered Architecture Processors using OpenCL

Niklas Rother*, Leonard Mätzner*, Pekka Jääskeläinen†, Topi Leppänen†, Jens Schleusner* and Holger Blume*
*Institute of Microeletronic Systems
Leibniz Universität Hannover, Hannover, Germany
E-Mail: {rother,schleusner,blume}@ims.uni-hannover.de
†Customized Parallel Computing Group
Tampere University, Tampere, Finland
E-Mail: {pekka.jaaskelainen,topi.leppanen}@tuni.fi

***Abstract*—Live SAR imaging from small UAVs is an emerging field. On-board processing of the radar data requires high-performance and energy-efficient platforms. One candidate for this are Transport Triggered Architecture (TTA) processors. We implement Backprojection and Backprojection Autofocus on a TTA processor specially adapted for this task using OpenCL. The resulting implementation is compared to other platforms in terms of energy efficiency. We find that the TTA is on-par with embedded GPUs and surpasses other OpenCL-based platforms. It is outperformed only by a dedicated FPGA implementation.**

***Index Terms*—SAR, TTA, energy-efficiency, backprojection**

I. Introduction

Imagery collected using Synthetic Aperture Radar (SAR) can be used in a wide variety of fields, ranging from agriculture over civil protection to military applications. Currently, most SAR data is collected using aircraft or satellites as carrier platforms and processed after collection by large computer clusters [1]. Lately, there has been interest in using small unmanned aerial vehicles (UAVs, "drones") as carrier platforms [2]–[4]. When the collected data could be processed in real-time on board the drone, a live feed of the SAR image could be sent to ground, similar to optical video feeds from current UAVs. This could open new applications in the aforementioned fields.

The raw radar collected by the drone has a high data rate and can hardly be transferred to a ground station in real-time. This makes on-board processing essential. The resulting images can be compressed lossy to fit the limited downlink channel capacity. A key challenge for this is the high computational complexity of SAR data processing and the limited weight, space, and power allowance of the UAV.

In this work, we explore the use of customized processors specialized for SAR processing (application-specific instruction set processors, ASIPs) to tackle this problem. One architecture that is specially optimized to reach high performance while maintaining low power consumption is the Transport Triggered Architecture (TTA) [5], [6]. In this processor design, only data transports are explicitly coded, operations happen as a side-effect thereof.

The OpenASIP project [7] provides a framework to generate, simulate, and implement ASIPs based on the TTA paradigm. The resulting processors can be programmed in OpenCL using the open-source implementation PoCL [8]. The main contribution of the work is the description of a TTA processor optimized for SAR processing and its throughout comparison to other platforms.

To handle the non-linear flight path of a wind-exposed drone, Backprojection [9] is used as the image formation algorithm, augmented by Backprojection Autofocus [10] for residual error correction.

II. Related work

The Backprojection algorithm for image formation has been implemented on a variety of platforms. To the best of the author's knowledge, the use of TTA processors has not yet been explored for synthetic aperture radar algorithms.

Broich [11] reports on a custom DSP architecture optimized for Range-Doppler processing, which draws some ideas from TTAs. The proposed architecture shows a 14 × reduction in cycles, compared to a Texas Instruments C66x. Aside from that, there are some reports of TTAs being used for software-defined radio (SDR) implementation of mobile communication standards [12]–[14]. The TTA described in [13], which is optimized for decoding of polar codes, is reported to outperform state-of-the-art ASIP implementations fivefold in throughput while consuming an order of magnitude less energy. Compared to an x86 processor, the throughput is increased by 37 % and the energy consumption is two orders of magnitude lower. In [12] the authors describe a multi-TTA architecture for LTE baseband processing. They find that their TTA implementation achieves a 1.85 to 10.73 better mW/MHz rating than comparable architectures. The "LordCore", a TTA processor for the decoding of Turbo-Codes [14] is said to outperform GPU-based implementations by three orders of magnitude in the performance/power ratio. Compared to fixed-function implementations of the algorithm, the TTA implementation requires more power, but "[the] results show that the energy

This work was supported by a fellowship of the German Academic Exchange Service (DAAD) and a scholarship of the Gradudiertenakademie of the Leibniz Universität Hannover.

978-1-6654-8278-3/23/$31.00 ©2023 IEEE

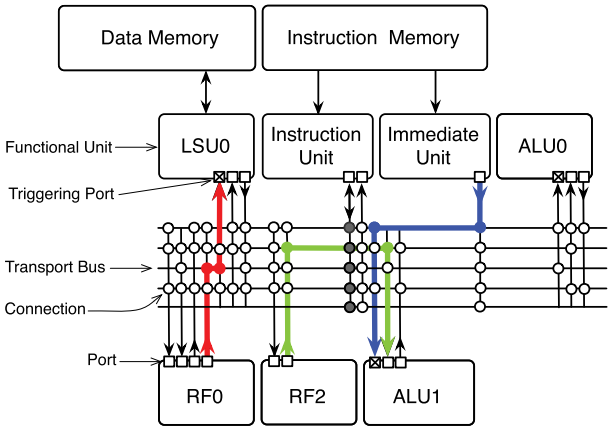

Fig. 1. Structure of a TTA processor with two ALUs, two register files, and five transport buses. The colored arrows indicate data transports that are executed simultaneously. Reproduced from [14].

penalty paid by programmability can be made very small" [14].

The mentioned reports indicate that similar advantages in the performance/power ratio may also be achieved in the field of SAR processing.

III. Transport Triggered Architecture

In the Transport Triggered Architecture (TTA) operations are executed as a side effect of data transport [15]. As a consequence, a TTA program consists of a list of data transport instructions, or "moves". This is in contrast to conventional processors, where the operation on the data is encoded in the instructions, and data transport happens as a side effect. The TTA methodology has some properties that are advantageous, especially for embedded low-power applications: Since the structure and internal architecture of the processor is directly exposed to the compiler, much of the complexity can be moved to the compile-time, reducing the hardware needed for decoding and processing of instructions. Additionally, results can often be moved directly from the output of one functional unit (FU) to the input of the next one, bypassing the register file. This greatly reduces the number of register file ports, which is a major concern for conventional Very Long Instruction Word (VLIW) architectures. Both properties of the TTA concept yield a reduced hardware complexity and therefore reduced power consumption.

Fig. 1 shows an example TTA processor. The horizontal lines indicate transport buses through which data is transferred between the FUs. Each connection between an FU's input or output port and the bus system is called a socket. It shall be noted that not all possible connections between a socket and the transport buses must be realized. Omitting connections reduces the required hardware resources together with the instruction word length, but removes flexibility from the resulting architecture. Each bus can operate independently from the others so that multiple transports (and therefore operations) can happen at the same time.

The number and type of FUs, the number and size of register files, and the interconnection network provide multiple independent customization points to adapt the TTA processor to the application. In this work, the processor template and customization tools from the OpenASIP project [7] were used. More information about the associated workflow can be found in [6].

IV. OpenCL implementation of SAR Algorithms

To implement the selected SAR algorithms—Backprojection and Backprojection Autofocus—on the TTA, the open source OpenCL implementation PoCL [8] was employed. It supports, among others platforms, TTAs designed with the OpenASIP toolkit. More information on PoCL can be found in [8].

The OpenCL programming model consists of buffers, which contain the data being processed, and kernels, which are small programs that describe the operations to apply to the data in the buffers. Kernels can be launched on a grid with up to three dimensions. The OpenCL runtime automatically parallelizes the kernel instances when possible, using multiple cores of the compute device.

The OpenCL host program is implemented in Python using PyOpenCL [16]. It runs on the host processor of the target platform and orchestrates the kernels launched on the compute device. The kernels itself are implemented in OpenCL C.

To avoid the call overhead, each kernel invocation can process a small patch of pixels in a loop. This increases the performance, but limits the image dimensions to multiple of the patch size. We used a size of 16×1. All values are represented using a fixed point number representation.

A. Backprojection

The Backprojection algorithm [9] is implemented as a single kernel, which projects all echo lines onto the image grid. In case that the image does not completely fit into the available memory, the kernel must be launched multiple times to process different parts of the image. In the Backprojection algorithm all pixels are computationally independent, so this can be done without problems. Similar splitting is applied for the echo data input: If the input exceeds the available memory, the kernel is launched multiple times until all collected echo lines are projected and accumulated. The CORDIC algorithm [17] is used for complex rotations and vector length and angle calculations.

B. Backprojection Autofocus

The Backprojection Autofocus algorithm [10] works by finding the optimal phase correction for every echo data line w.r.t. some sharpness metric. The phases are optimized one after the other, in a coordinate descent approach. In the original work by Duersch and Long [10] Brent's algorithm is used to find the minimum of the (negated) sharpness metric. This approach allows only one candidate correction to be evaluated simultaneously. For parallel architectures, like GPUs or FPGAs this can be a bottleneck. The PAFO ("Parallel Autofocus Optimization") variant [18], [19] overcomes this

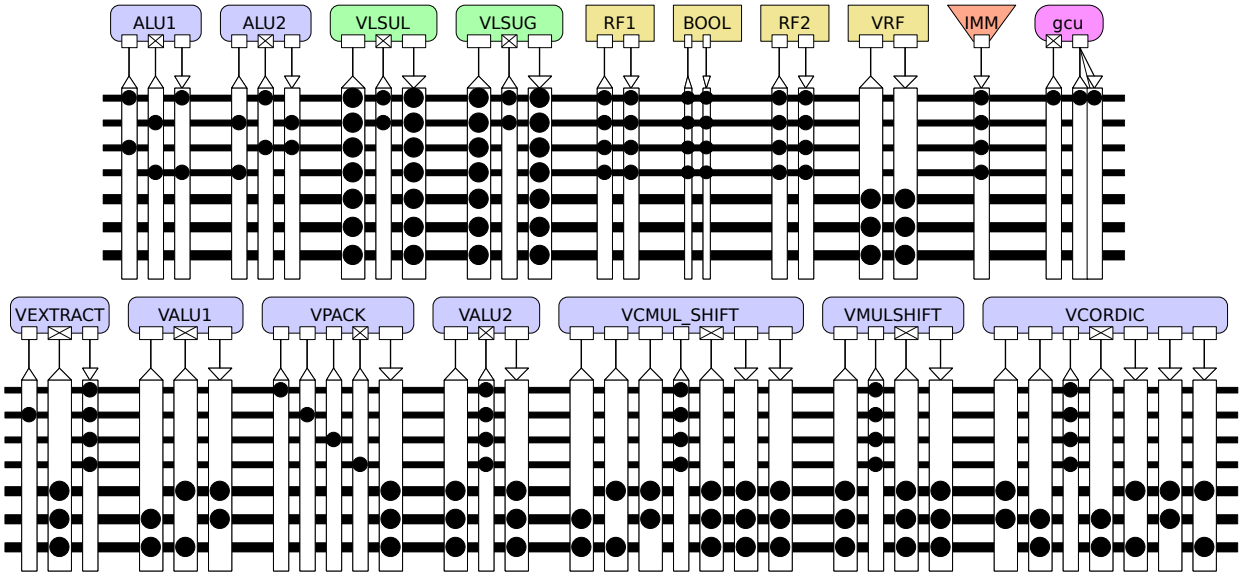

Fig. 2. Scalar (top) and vector (bottom) datapath of the proposed TTA architecture. It consists of nine FUs, two load-store units, four register files (two scalar, one boolean, and one vector), one long immediate unit, and a global control unit.

limitation by using a specialized numeric optimization algorithm that allows multiple samples to be evaluated in parallel. Even for architectures with limited parallelism, like TTAs, this approach is beneficial, since the overhead of evaluating multiple candidate phases instead of one is small. For this reason, the PAFO algorithm is used here.

The Backprojection Autofocus algorithm starts by calculating the contribution of a single echo line to the final image. For this, the same Backprojection kernel as for the initial image formation is used. Afterwards two additional kernels are used to calculate the sharpness metric for multiple candidate pulses, and apply the final phase correction for echo data line, respectively. If the image does not fit into the device memory, the kernels have to be launched multiple times. In case of the sharpness metric calculation, the results from all image slices are summed on the host CPU to find the final value.

The actual numeric optimization algorithm, i.e. the selection of the candidate phases and the interpolation of the final phase correction is executed always on the host CPU using floating point calculations.

V. TTA Architecture and Optimizations

The implemented TTA processor is a single-core, 4×32 SIMD architecture. A visual overview is given in Fig. 2. It features (mostly) separate datapaths with a total of 9 FU for scalar and vector operations as well as two scalar and one vector register file. The processor has access to two address spaces, "global" and "local". The 512 KiB global address space is accessible via an AXI bus by the host processor and is used for OpenCL buffers and to communicate with the host. The 1 KiB local memory is only accessible by the TTA and contains the stack and OpenCL local and private memory regions, if any. The instruction length of the processor is 122 bit. The processor is controlled using a debug interface over the AXI bus.

A. Special instructions

To avoid the penalty in hardware resources and power consumption introduced by floating-point capable hardware, only fixed-point operations are supported. Furthermore, some key calculations in the SAR algorithms were identified, and custom special instructions were added to support these operations. Every group of instructions is implemented in a similar named FU (cf. Fig. 2).

1) Multiply-and-shift: In every fixed-point multiplication, the result must be shifted to adjust the position of the binary point. A special instruction `MULSHIFT` was added, which performs the multiplication and shift in a single instruction. This additionally waives the need for 64 bit registers for storing the intermediate results of a 32 bit multiplication.

2) CORDIC: As mentioned in Section IV-A, the CORDIC algorithm is used for several vector operations. To support this, two operations, `CORDIC_CIRC_ROT` and `CORDIC_CIRC_VECT`, were added, which perform a single CORDIC iteration in circular rotation and vectoring mode, respectively. A third operation, `CORDIC_QUAD_MAP`, performs the initial quadrant mapping operation.

3) Complex multiplication: To aid with complex multiplication, an instruction `CMUL_SHIFT` was added that performs a complex multiplication of two integers (treated as real and imaginary part) and performs the necessary shift to correct for the fixed point format.

B. SIMD

Every kernel instance operates on a patch of 16 pixels in a loop inside the kernel. Identical operations are applied for every pixel, so the loop can be effectively vectorized with

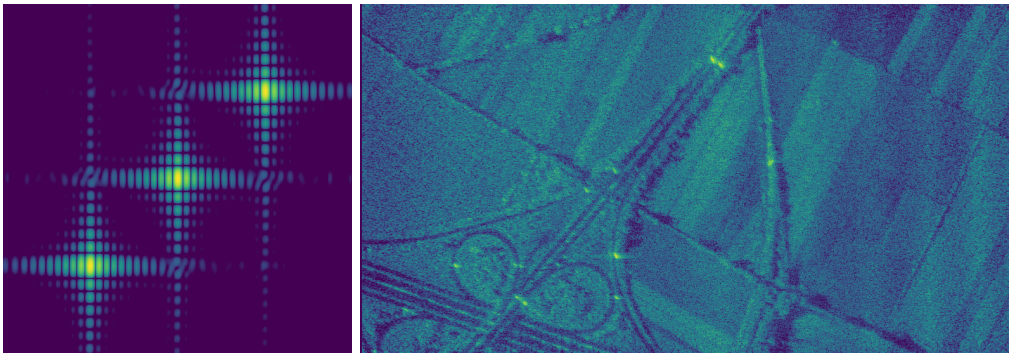

Fig. 3. Datasets used for evaluation. Left: A simulation with three point reflectors. Right: Highway crossing, collected at X-band (9.99 GHz).

SIMD (Single Instruction Multiple Data). A SIMD width of 4 was chosen—resulting in 128 bit wide vectors—so the kernel loop is reduced to four iterations. The vector data path of the processor includes SIMD versions of all basic and custom operations and the Load-Store units are extended to support 128 bit operations.

VI. Evaluation

To assess the performance of our proposed implementation, we synthesized it for an ASIC technology and evaluated its performance based on an FPGA-based prototype. Energy results are compared to other OpenCL-based implementations as well as other implementations of the Backprojection algorithm. For evaluation, we used two datasets (cf. Fig. 3 and Table I): One synthetic, featuring three point reflectors, and a real, showing a highway crossing [20]. The latter was collected at X-band and is reminiscent of a realistic situation for the aforementioned UAV scenario.

Data processing consists of two steps: Backprojection image formation and Autofocus processing. The latter can be broken down into the generation of the single pulse image ("SPI") using Backprojection, the sampling step to generate the metric values from candidate corrections ("Sample"), and the resulting phase correction ("Apply").

For each step, platform, and dataset we measured the time spent executing the OpenCL kernels by using OpenCL profiling information. For an evaluation of the energy efficiency, we measured the power requirements of each platform during the execution of the OpenCL program.

We used two PAFO iterations, each with 8 samples, throughout the evaluation. The output image size was 512 px × 512 px, except for the TTA-FPGA prototype, where 128 px × 128 px was used to reduce the runtime. The synthetic and highway dataset consists of 256 and 1000 echo lines, respectively. All measurements were repeated ten times and then averaged.

A. ASIC Synthesis

For evaluation purposes, the system was implemented on a PYNQ-Z1 evaluation board. To run the OpenCL host program and control the TTA processor, Linux was run on the chip's ARM cores.

Afterward, an ASIC frontend synthesis was performed for the commercial 22 nm Fully-depleted Silicon-on-Insulator technology from GlobalFoundries ("22FDX") using Cadence tools for a clock frequency of 700 MHz. A multi-corner analysis with 0.72 V to 0.88 V operating voltage and −40 °C to 125 °C ambient temperature was performed. The resulting chip area is 971 μm^2 with a utilization of 37.6 %. The critical path is formed by the vector register file and the vectorized ALU. In that path, 803 ns (56.2 %) are spend in the register file alone.

The kernel runtime for the TTA/ASIC was calculated from FPGA prototype measurements by scaling with the clock frequency. The power requirements were estimated after the frontend synthesis process. 54 % of the estimated 201 mW are used for the SRAM memory.

B. Reference platforms

To provide baseline results, we compared the implementation to reference platforms whose power and space requirements make them also suitable for use on board UAVs.

1) NVIDIA Jetson AGX Xavier: The Jetson AGX Xavier consists of a 512-core CUDA-capable embedded GPU (Nvidia Volta) as well as eight ARMv8.2 cores (Nvidia Carmel). It is fabricated in a 12 nm process [21]. From prior work, [19] we report on a CUDA implementation of the Backprojection and PAFO algorithm. This implementation is labeled "Xavier/CUDA" later on. As no detailed timing information for the different steps of the PAFO algorithm was available, only Backprojection is considered in the evaluation. Contrary to the other implementations, 32 bit floating point numbers are used here.

Additionally, we also tried using OpenCL code similar to the one for the TTA processor on this platform. As the platform has no native OpenCL support, we used PoCL's *cuda* backend to execute the OpenCL code on the GPU (labeled

TABLE I
System parameters for datasets used in the evaluation.

	Simulated	Highway
System	MATLAB	AER-II (?) [20]
Center freq.	10 GHz	9.99 GHz
Bandwidth	40 MHz	150.1 MHz
Pulse repetition freq.	256 Hz	190.5 Hz
Echo Lines	256	1000
Scene size	200 m × 200 m	551 m × 1026 m
Flight height	1 km	2.3 km

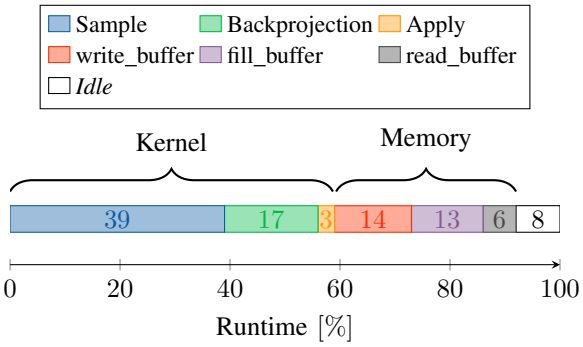

Fig. 4. Runtime distribution for the TTA/FPGA prototype.

"Xavier/PoCL"). Furthermore, we used the CPU for kernel executing, using PoCL's *pthread* backend ("Xavier/CPU"). Power consumption on the Xavier platform was measured using the onboard power sensors and included all power rails except SYS5V (peripheral hardware).

2) HardKernel Odroid N2+: The Odroid N2+ features a Mali-G52 GPU (Bifrost) with proprietary OpenCL drivers which could be used directly without PoCL. It is also fabricated in a 12 nm process. The OpenCL code used was identical to the one for the Xavier platform. Power consumption was measured using a laboratory power supply.

3) FPGA Implementation: As a final comparison we used a dedicated FPGA implementation for Backprojection processing [22]. It runs on a Xilinx ZCU102 evaluation board, featuring a Zynq UltraScale+ XCZU9EG-2FFVB1156 MPSoC fabricated in TSMC's 16FinFET+ process. The system is operating at 240 MHz and processes 32 pixels per clock cycle. Only Backprojection is supported. Power consumption was measured using the onboard sensors, including only the FPGA-related power rails.

C. Results

In a first step, we extracted the runtime for the different OpenCL operations on the TTA/FPGA prototype using PoCL's tracing functionality. The results in Fig. 4 show that 59 % of the time is spent executing the kernels while 33 % of the time is used for memory operations. To improve this, a double-buffering scheme similar to the one employed in the FPGA implementation could be used [22].

The most important metric in our study is the energy spent for a single operation (energy per operation, in nJ). It is derived from the operation count, the total processing time, and the power consumption of the platform. The measured average power values are listed in Table II.

For Backprojection, the total operation count was calculated from the number of pixels in the output image multiplied by the number of echo lines. For the SPI and Apply steps of the PAFO algorithm only the number of pixels is used; the operations in the Sample step are defined by the number of pixels and the total number of samples ($2 \times 8 = 16$ in our case). For example, using the simulated dataset with 256 echo lines and a 512 px × 512 px image grid requires $256 \cdot 512 \cdot 512 = 67\,108\,864$ operations for Backprojection, $512 \cdot 512 = 262\,144$ operations for the SPI and Apply step, and $512 \cdot 512 \cdot 8 \cdot 2 = 4\,194\,304$ operations for the Sample step.

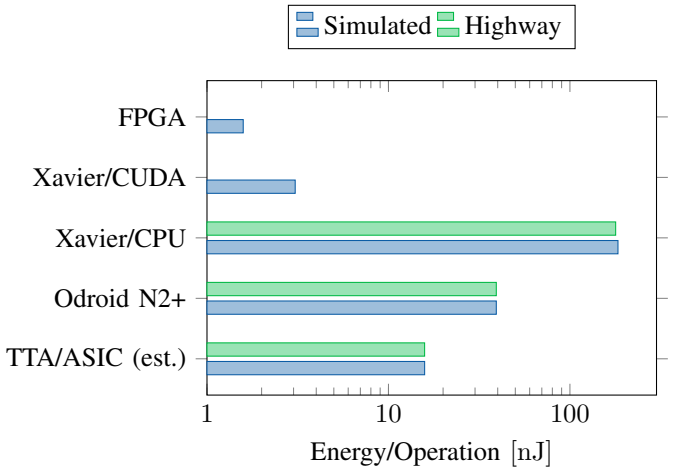

Fig. 5. Energy per operation for Backprojection for different platforms and datasets. The TTA reaches a comparable level of performance with regard to other programmable platforms.

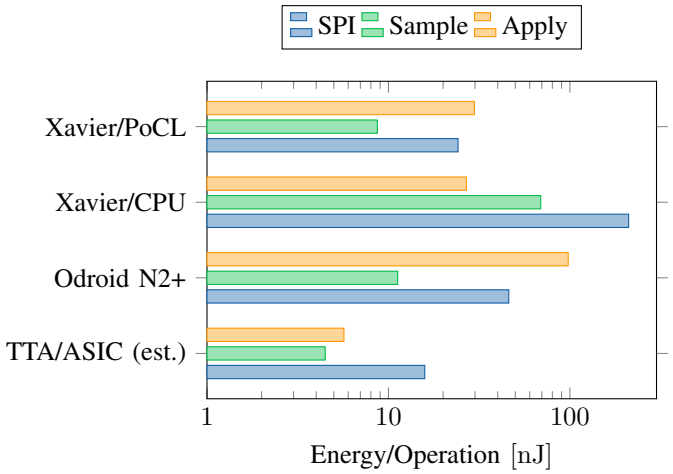

Fig. 6. Energy per operation for three steps of the Autofocus algorithm for different platforms, all using the same OpenCL code and the simulated dataset. The TTA reaches the best energy efficiency. For the Xavier/CUDA and FPGA platform no data was available.

Results are depicted in Figs. 5 and 6. The energy efficiency for Backprojection is consistent over the two datasets. Compared to the CPU-only implementation and the low-end Odroid, the TTA is superior in energy efficiency. Still, for Backprojection, it is outperformed by the Xavier/CUDA and the FPGA. The former is fabricated using a newer technology node (12 nm vs 22 nm) and—as a commercial product—is

TABLE II
MEASURED POWER CONSUMPTION AND TECHNOLOGY NODES FOR THE DIFFERENT PLATFORMS.

Platform	Power / W	Technology Node
TTA (est.)	0.201	GF 22FDX
Odroid N2+	3.349	12 nm
Xavier AGX (CPU)	9.923	12 nm
Xavier AGX (GPU)	14.054	12 nm
FPGA (ZCU102)	11.686	TSMC 16FinFET+

optimized better, which puts this into perspective. The FPGA is different insofar, that it is a non-software-programmable platform where the algorithm has to be implemented in a hardware description language and therefore requires higher development effort and provides no runtime flexibility. This pays off in terms of energy efficiency, where the FPGA surpasses the TTA nearly by an order of magnitude.

In the Autofocus algorithm, the Sample and Apply steps execute fewer instructions for each operation compared to Backprojection and SPI. Therefore, higher energy efficiency is expected for these steps. This is seen in the CPU platforms (TTA and Xavier/CPU), but not for the GPUs (Odroid and Xavier/PoCL); giving rise to the idea that these parts of the algorithm are not well suited for the architecture of a GPU or at least need special optimizations. For the Autofocus algorithm, the TTA reaches the best energy efficiency of all platforms investigated. While the results for Xavier/CPU and the Odroid are consistent with the observations for Backprojection, the low performance of Xavier/PoCL is unexpected. This might indicate some missing GPU-specific optimizations in the OpenCL code compared to the CUDA implementation.

VII. Conclusion

In this work, we explored the possible use of a Transport Triggered Architecture (TTA) processor in the field of Synthetic Aperture Radar processing, with a particular focus on energy efficiency. The results are multifaceted:

With 15.8 nJ/op for Backprojection, the TTA achieved a comparable level of performance to other software-programmable platforms, being outperformed only by the Jetson AGX Xavier using CUDA. The winner in terms of energy efficiency is the dedicated FPGA platform. With 1.59 nJ/op, it exceeds the TTA by almost an order of magnitude. However, taking into account the different technology nodes and the much higher implementation effort and reduced runtime flexibility for an FPGA implementation, this is put into perspective.

In general, the TTA is found to be a very energy-efficient platform. This is in line with prior reports for different applications (cf. Section II).

Based on these results, further optimizations of the TTA platform can be considered. Multiple TTA cores could be implemented on a single chip to increase the throughput of the system. The OpenCL runtime would automatically scale the implementation to a multicore system. Alternatively, wider SIMD vectors could be explored for a similar effect. In terms of energy efficiency, several TTA-specific optimizations could be used, including an Instruction Register File (IRF) or instruction compression [23].

Acknowledgments

Niklas Rother thanks T. Fiedler for his help with the ASIC synthesis. Tampere University authors thank Academy of Finland (decisions #331344 and #353199).

References

[1] A. Moreira, P. Prats-Iraola, M. Younis, G. Krieger, I. Hajnsek, and K. P. Papathanassiou, "A tutorial on synthetic aperture radar," *IEEE Geoscience and remote sensing magazine*, vol. 1, no. 1, pp. 6–43, 2013.

[2] Z. Xu and D. Zhu, "High-resolution miniature UAV SAR imaging based on GPU architecture," in *Journal of Physics: Conference Series*, vol. 1074, no. 1. IOP Publishing, 2018, p. 012122.

[3] A. Bekar, M. Antoniou, and C. J. Baker, "Low-cost, high-resolution, drone-borne sar imaging," *IEEE Transactions on Geoscience and Remote Sensing*, 2021.

[4] ——, "High-resolution drone-borne sar using off-the-shelf high-frequency radars," in *2021 IEEE Radar Conference (RadarConf21)*. IEEE, 2021, pp. 1–6.

[5] P. Jääskeläinen, "From parallel programs to customized parallel processors," Ph.D. dissertation, Tampere University of Technology, 2012.

[6] P. O. Jäskeläinen, C. S. de La Lama, P. Huerta, and J. H. Takala, "Opencl-based design methodology for application-specific processors," in *2010 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation*. IEEE, 2010, pp. 223–230.

[7] P. Jääskeläinen, T. Viitanen, J. Takala, and H. Berg, *HW/SW Co-design Toolset for Customization of Exposed Datapath Processors*. Springer International Publishing, 2017, pp. 147–164. [Online]. Available: https://doi.org/10.1007/978-3-319-49679-5_8

[8] 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. [Online]. Available: http://dx.doi.org/10.1007/s10766-014-0320-y

[9] M. I. Duersch, "Backprojection for synthetic aperture radar," Ph.D. dissertation, Brigham Young University, 2013.

[10] M. I. Duersch and D. G. Long, "Backprojection autofocus for synthetic aperture radar," in *SPACEGRANT*, 2013.

[11] R. Broich *et al.*, "A soft-core processor architecture optimised for radar signal processing applications," Ph.D. dissertation, University of Pretoria, 2013.

[12] O. Anjum, T. Ahonen, and J. Nurmi, "Mpsoc based on transport triggered architecture for baseband processing of an lte receiver," *Journal of Systems Architecture*, vol. 60, no. 1, pp. 140–149, 2014.

[13] M. Léonardon, C. Leroux, P. Jääskeläinen, C. Jégo, and Y. Savaria, "Transport triggered polar decoders," in *2018 IEEE 10th International Symposium on Turbo Codes & Iterative Information Processing (ISTC)*. IEEE, 2018, pp. 1–5.

[14] H. Kultala, T. Viitanen, H. Berg, P. Jääskeläinen, J. Multanen, M. Kokkonen, K. Raiskila, T. Zetterman, and J. Takala, "Lordcore: Energy-efficient opencl-programmable software-defined radio coprocessor," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 5, pp. 1029–1042, 2019.

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

[16] A. Klöckner, N. Pinto, Y. Lee, B. Catanzaro, P. Ivanov, and A. Fasih, "PyCUDA and PyOpenCL: A Scripting-Based Approach to GPU Run-Time Code Generation," *Parallel Computing*, vol. 38, no. 3, pp. 157–174, 2012.

[17] J. E. Volder, "The cordic trigonometric computing technique," *IRE Transactions on electronic computers*, no. 3, pp. 330–334, 1959.

[18] C. Fahnemann, D. Fallnich, A. Sommer, F. Cholewa, and H. Blume, "Hardware-optimized minimum-search for SAR backprojection autofocus on FPGAs," in *2019 International Radar Conference*. IEEE, 2019, pp. 1–5.

[19] N. Rother, C. Fahnemann, J. Wittler, and H. Blume, "Optimized minimum-search for sar backprojection autofocus on gpus using cuda," in *2020 IEEE Radar Conference (RadarConf20)*. IEEE, 2020, pp. 1–5.

[20] J. Ender, P. Berens, A. Brenner, L. Rossing, and U. Skupin, "Multi-channel sar/mti system development at fgan: from aer to pamir," in *IEEE International Geoscience and Remote Sensing Symposium*, vol. 3, 2002, pp. 1697–1701 vol.3.

[21] D. Schor. Hot chips 30: Nvidia xavier soc. [Online]. Available: https://fuse.wikichip.org/news/1618/hot-chips-30-nvidia-xavier-soc/

[22] F. Cholewa, M. Pfitzner, C. Fahnemann, P. Pirsch, and H. Blume, "Synthetic aperture radar with backprojection: A scalable, platform independent architecture for exhaustive fpga resource utilization," in *2014 International Radar Conference*. IEEE, 2014, pp. 1–5.

[23] J. Multanen, "Energy-efficient instruction streams for embedded processors," Ph.D. dissertation, Tampere University, 2021.