OpenCL Programmable Exposed Datapath High Performance Low-Power Image Signal Processor

Joonas Multanen, Heikki Kultala, Matias Koskela,
Timo Viitanen, Pekka Jääskeläinen, Jarmo Takala
Tampere University of Technology, Finland
Email: {joonas.multanen, heikki.kultala,
matias.koskela, timo.2.viitanen,
pekka.jaaskelainen, jarmo.takala}@tut.fi

Aram Danielyan, Cristóvão Cruz
Noiseless Imaging Ltd, Finland
Email: aram@noiselessimaging.com,
cristovao@noiselessimaging.com

***Abstract*—Sophisticated computational imaging algorithms require both high performance and good energy-efficiency when executed on mobile devices. Recent trend has been to exploit the abundant data-level parallelism found in general purpose programmable GPUs. However, for low-power mobile use cases, generic GPUs consume excessive amounts of power. This paper proposes a programmable computational imaging processor with 16-bit half-precision SIMD floating point vector processing capabilities combined with power efficiency of an exposed datapath. In comparison to traditional VLIW architectures with similar computational resources, the exposed datapath reduces the register file traffic and complexity. These and the specific optimizations enabled by the explicit programming model enable extremely good power-performance. When synthesized on a 28nm ASIC technology, the accelerator consumes 71mW of power while running a state-of-the-art denoising algorithm, and occupies only 0.2mm^2 of chip area. For the algorithm, energy usage per frame is 7mJ, which is 10x less than the best found GPU-based implementation.**

I. Introduction

Contemporary high quality video and image processing algorithms require high-performance hardware to be able to run in real-time. For non-mobile devices, the energy budget is usually not a limiting factor in contrast to mobile devices where the battery technology sets an additional energy constraint. Therefore, running programs utilizing state-of-the-art computational imaging algorithms on mobile devices is challenging due to their high computational performance requirements combined with the energy efficiency demands.

What helps in reaching the goals is the fact that image processing algorithms can often utilize parallel hardware resources due to the nature of the algorithms involved. Due to this, in the past years, GPUs with abundance of data parallel resources have been widely used [1]–[3] for image processing. However, GPUs designed originally for graphics rendering with added hardware to enable more general purpose uses are not the most energy-efficient possible. This makes them inoptimal for battery-dependent use cases such as complex image processing on satellites, drones or battery driven surveillance cameras. In addition, computational imaging algorithms are often highly memory-intensive, causing a memory bottleneck as data is constantly accessed.

For the lowest power consumption, some sort of specialized accelerators are typically used instead of general purpose CPUs and GPUs. The least power consuming type of accelerators is a fixed function accelerator, which can achieve extremely energy-efficient operation with the expense of programmability and flexibility. Fixed function accelerators reach the power performance by means of tailored arithmetics and simplified control hardware with fixed logic. However, fixed function hardware development and verification is time consuming. In addition, often it is desirable to execute more than one image processing algorithm on a single device. *Application-specific Instruction-Set Processors (ASIPs)* or "domain-specific processors" are typically used as accelerators, when more programmability is required, while still reaching low energy consumption. In the image processing field, *Image Signal Processors (ISPs)* is the common term used for processors customized for typical image tasks such as demosaicing, filtering or correction of camera imperfections.

This paper introduces an ISP targeting low-power use cases. It achieves extremely good power performance by means of an explicit programming model which simplifies the control logic and reduces register file power consumption. The processor was designed to not include excessive function units, which reduces the area and improves the power foot print. On the other hand, specialized function units very avoided to maintain generality. The memory-intensity of image processing algorithms was considered while defining the architectural features. This resulted in an extremely energy-efficient design which uses no special hardware, yet delivers high computational performance.

This paper is organized as follows. In Section II, the accelerator design is presented. Section III describes the evaluation of the design and presents the results. Section IV reviews related work. Finally, Section V concludes the paper.

II. Accelerator Design

A. Transport Triggered Architecture

A popular architecture style used in contemporary DSPs and ISPs is the so called *Very Long Instruction Word (VLIW)*. In order to exploit *Instruction Level Parallelism (ILP)*, multiple different operations can be executed in parallel in a VLIW processor. This is achieved with multiple parallel *Function Units (FUs)* that can implement common operations, or customised operations to accelerate bottlenecks in programs. ILP is a more free form of fine grained parallelism than *Data*

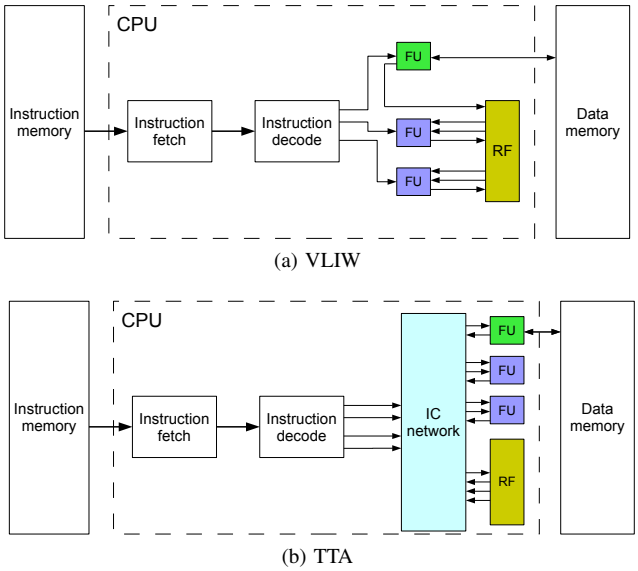


Fig. 1. VLIW and TTA processor architectures.

Level Parallelism (DLP) where a single operation is applied to elements of a vector of data.

VLIW architecture is illustrated in Fig. 1a. The flow of instruction execution proceeds from left to right. Instructions are fetched with an instruction fetch component, after which they are decoded into control signals for the rest of the processor. The instruction word controls the FUs explicitly, thus its size is proportional to the number of parallel FU operations, hence its name VLIW.

In VLIWs, the order of instructions is decided by the compiler during compile-time. This is called *static scheduling*. An opposite approach is *dynamic scheduling*, where the processor's control hardware detects dependencies between instructions and reorders them on the fly. This is common especially in contemporary general purpose CPUs. The simplified control hardware of VLIWs makes the control logic smaller and less power hungry.

VLIW allows multiple parallel operations, however, when multiple operations finish at the same time, they need to store their results to a *Register File (RF)*, or *bypass* it and move their result straight to another FU. The worst case is, when all FUs want to write to the RF at the same time, because in that case the number of RF write ports required is the number of parallel FUs [4]. Also, scaling the architecture larger in size increases the complexity of the bypass logic. Bypassing means wiring the output of an FU directly to another FU's input instead of first storing it into the RF. This helps decreasing the execution latency as there is no need to wait for the register update stage to consume the result of a previous operation. All or a subset of the FUs can be connected in this manner.

The "MOVE architecture" [5] was presented by Lipovski in 1976. In this architecture, operations were triggered by moving operands to inputs of FUs. Thus, the operations were *transport triggered*. This was different from the traditional paradigm of *operation triggering* used also in VLIWs, where in addition to transporting the input operands and destination to the FU, an operation opcode is given. After this, the hardware performs the data transfers required for the operation. In a transport triggered architecture, the data transfers are explicitly controlled in the assembly code (which can be produced by a compiler from higher level languages).

TTAs were later studied extensively by Corporaal et al. [6]. They proposed the *MOVE* architecture to be used as an improvement to the traditional operation triggered VLIW architecture to alleviate their issues with scalability and register file complexity.

In Fig. 1, the structure of VLIW and TTA is compared. Both are statically scheduled architectures and use instruction fetch and decode units to control the CPU. However, TTAs have an *exposed datapath*, where the programmer controls individual moves between function units and/or register files. These happen on the *InterConnection (IC)* network. In a sense, the only instructions in a TTA are *Move* and *No Operation (NOP)*. The actual computation happens in FUs as a "side effect" of moving data to FU *trigger ports*. FUs still need an opcode port to indicate the operation to perform.

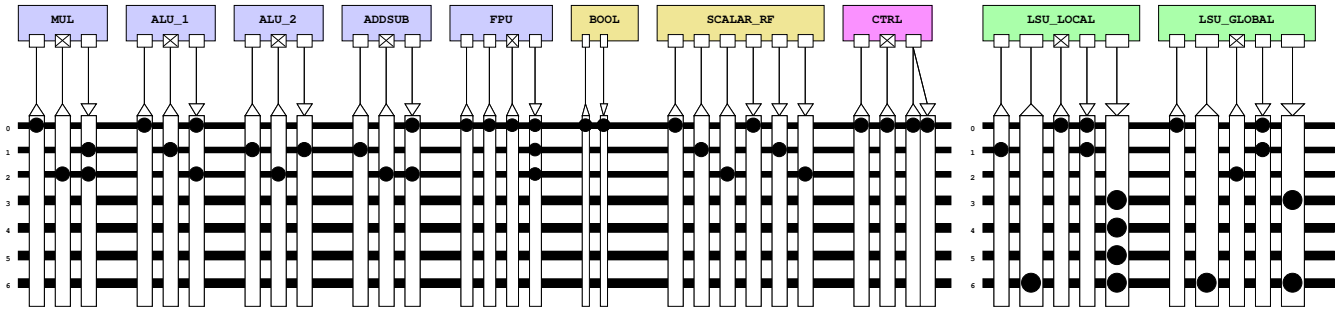
TTA addresses the VLIW scaling bottleneck with its programming model. Unlike in VLIW, the number of required general purpose RF ports does not directly depend on the number of FUs in TTAs. This is due to the fact that TTAs provide extensive freedom to program the data transports. Operands can be moved to registers in the FU input ports earlier than when the operation needs to execute, same way as results can be moved out later than when they are ready. This reduces the register file pressure, allowing smaller RF sizes and less RF ports. Both TTAs and VLIWs can perform register file splitting to reduce the number of ports in a single RF, but it adds even more complexity to the compiler due to the register file assignment decision. [7]

Register file bypassing in TTAs is done with *software bypassing*. That is, data can be moved directly between FUs by the programmer. This alleviates the bypass network scaling issue in VLIWs in addition to reducing the RF bottleneck. *Operand sharing* can be used to reuse input operands when two or more operations share the same input value. The value is kept in the input register until it is no longer needed. Software bypassing also allows *dead result elimination*, where results are not moved from the FU result register if they are not used, again saving register file accesses.

B. The Processor Design

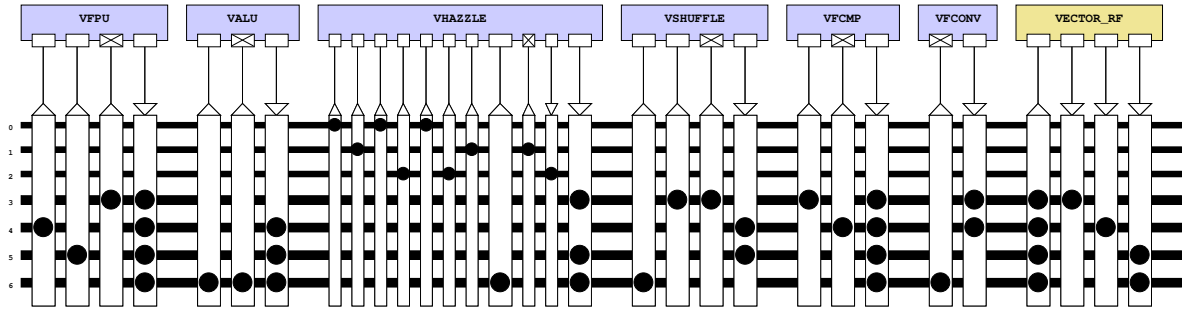
In addition to the RF simplification benefits, its modular and flexible nature makes TTA an interesting architecture for low-power programmable devices and as a template for rapid ASIP design. Due to the modularity, automatic *Register Transfer Level (RTL)* code generation of TTA processors is rather straightforward. However, simplicity of the hardware is traded off to the increased complexity of the high-level language compiler.

TTA-based Co-design Environment (TCE) [8] is a design and programming suite for TTAs, developed at Tampere University of Technology. It allows the user to generate *Hardware Description Language (HDL)* code from a processor designed with a graphical interface and includes automatically retargeting compilers and simulators for programming support. TCE was used to design the proposed ISP TTA.



(a) Scalar FUs, scalar RFs and control unit.

(b) Load-store units.



(c) Vector FUs and vector RF.

Fig. 2. The structure of the proposed TTA accelerator design.

TABLE I. SUMMARY OF THE PROPOSED ARCHITECTURE FEATURES.

local memories	8kB instruction SRAM, 4kB scratchpad Data SRAM
instruction width	128b
compute FUs	2xALU, 8 lane SIMD ALU, 8 lane SIMD FMA FPU
transport buses	3x32b, 4x128b
registers	2x1b, 32x32b, 32x128b
register file ports	scalar: 3 write, 3 read vector: 1 write, 3 read
peak floating point perf.	16 GFLOPS (FP16, 600 Mhz)

The aim of the processor design was to be able to run various image processing algorithms efficiently on the platform, yet achieve low-power high performance. To accomplish this, no special function units were exploited and only generic integer and floating point units were implemented. To exploit the data-level parallelism typical to image processing algorithms, *Single Instruction Multiple Data (SIMD)* operations were implemented and heavily utilized in the processor. SIMD operations have the benefit that only a single operation code needs to be encoded for multiple performed operations. Thus, the instruction bits per operation ratio is lucrative.

To achieve the low-power target, 16-bit floating point arithmetics were chosen for the accelerator. Using floating point operations also simplifies programming of the processor in contrast to fixed point operations, where data has to be manually scaled. This allows decrease the instruction memory usage. In addition to 16-bit floating point operations, we also included basic 32-bit scalar integer operations for addressing and other occasional needs.

With these initial decisions, the design process continued by defining the function units and interconnect. The number of function units was found empirically by simulating the processor and adding or removing FUs when needed. The main resources of the processor design are listed in Table I. Experimenting with different architecture features was quite effortless, thanks to TCE's automatic HDL generation and cycle-accurate simulator.

Finding the optimal configuration of the IC in TTAs is complicated due to the fact that changes to the IC can drastically affect the scheduling and thus the compiled program. In our design, the InterConnection network and register file amounts and sizes were discovered with an automated optimizer algorithm [10]. The interconnect was optimized to allow high performance yet keep the silicon area utilization low. The designed architecture is illustrated in Fig. 2. For clarity, Scalar FUs, LSUs and vector FUs are separated into 2a, 2b, 2c, respectively. The architecture has two load-store units: one

TABLE II. PARAMETERS OF THE EVALUATION CODE.

block size	8
matches in 3D group	8
image size	3072x1536
step size between reference blocks	7
3D transformation method	Haar

TABLE III. COMPARISON OF THE EVALUATED PLATFORMS.

	Mali-T628 MP6 [9]	Adreno 330	Intel HD5500	TTA
process tech.	28nm	28nm	14nm	28nm
FP vector width	8x16b	-	8x32b	8x16b
clk freq. (MHz)	600	450-578	850-950	600
processing elements	6 cores	4 cores	24 execution units	1 core

for the local on-chip scratchpad data memory and one for the external global memory. The InterConnection network connections are shown in the bottom part, where the top 3 buses are for scalar data and bottom 4 for vector data.

In order to alleviate the data memory access bottleneck, we added a local memory. Increasing the size of the RF would have been an option, but this would have had a large impact on the RF power consumption. In addition, RFs cannot be indexed dynamically, reducing their applicability. A local scratchpad SRAM was a compromise between access time and power consumption. The local scratchpad data memory was implemented as a banked design so that 8 half-precision floating point values could be loaded in parallel. The parallel fetch was designed to complete in 3 cycles, which is more than feasible for a local scratchpad memory.

For achieving high computational performance, 16-bit floating point SIMD units with 8 lanes were implemented. The SIMD function units always operate on all 8 lanes per instruction. At maximum the design can start an 8-wide SIMD operation in its SIMD ALU, an 8-wide SIMD FMA (commonly counted as two operations), one operation in the scalar FP ALU and one scalar FMA (again counted as two) resulting in a peak floating point performance of 16 half-precision GFLOPS with the targeted 600 MHz clock frequency. Support for denormal numbers was not included in the implementation. This reduced the available precision when presenting floating point numbers, but helped in saving energy due to the denormal handling logic being relatively complex.

The reduction in RF ports in TTA over VLIW can be clearly seen in Fig. 2a and 2c. In VLIW, the worst case scenario would be all 5 scalar FUs and all 6 vector FUs writing into an RF at once. This would require total of 5 write ports in scalar RFs and 6 write ports in SIMD RFs. The same TTA design uses 3 write ports in the scalar RF and 1 write port in the vector RF. In TTA the amount is adjustable according to the performance requirement, as in VLIW it is dictated by the number of FUs.

III. Evaluation

A. Benchmark Application

BM3D [11] is a state-of-the-art image denoising algorithm which provides very good quality results. It was used to evaluate the implemented accelerator, since it is challenging to implement efficiently due to its memory-intensity and thus is well suited for benchmarking. Two main computational steps are identified in BM3D filtering: *Block Matching (BM)* and *filtering* in 3D transform domain. In the BM step, an input image is divided into overlapping blocks, consisting typically of 8x8 pixels. Within the image some blocks are selected as reference blocks. For these reference blocks, a number of (typically 8 or 16) similar blocks are found. The best matches together with their reference block are stacked together to form a *3D group*. BM3D exploits the observation, that real-life images often have similar blocks due to images containing surfaces, textures and similar areas.

In the second phase of BM3D, the groups are individually filtered. First, the groups are 3D-transformed. Depending on the desired execution time and quality of results, algorithms such as *Discrete Cosine Transform (DCT)* or *Haar transform* can be used. Next, thresholding followed by an inverse 3D-transform is performed on the groups.

The benchmark program was implemented as an OpenCL [12] application with kernels that exploit the SIMD instructions explicitly using the vector datatypes. It was compiled for the TTA processor with TCE's compiler that utilizes *Portable Computing Language (pocl)* [13], an implementation of the OpenCL standard. The code performed the filtering phase of the BM3D algorithm, where a 3D Haar transform was performed to the input data, after which thresholding was done. Finally, an inverse 3D Haar transform was performed. Typical to image processing algorithms, the filtering phase is very memory-intensive. Therefore, it offered a good evaluation point for the design. Parameters used for filtering are described in Table II.

The TTA accelerator was compared against three platforms: An Odroid-XU3 with a Mali-T628 MP6 GPU, Dragonboard APQ8074 with an Adreno 330 GPU and an Intel Core i7-5600 with an integrated Intel HD5500 GPU. Summary of the evaluated architectures is presented in Table III. The OpenCL benchmark code was optimized individually for these three comparison platforms.

B. Results

The TTA processor was synthesized with Synopsys Design Compiler, on a 28nm FDSOI technology with a 600MHz target clock frequency. The supply voltage was 1.0V. The timing constraint was still met at 1.1GHz after synthesis, with the critical path in the SIMD floating point unit. For accurate power estimation, the processor was simulated in ModelSim (10.4), from where switching activity of the benchmark program was recorded and used in Design Compiler power analysis. The power consumption for Intel HD5500 was obtained with GPU-Z [14] and for Mali-T628 with the software shipped with the Odroid-XU3 development board. All of the results are produced from post-synthesis analysis.

ASIC synthesis results are presented in Table IV. The area, 0.2mm^2, includes the local instruction and data SRAM memories. The TTA core area occupied 38% of the total area and the memories 62%. The average power consumption was 71mW while executing the benchmark code at 600MHz.

The TTA accelerator computes one 3072x1536 image in 230 ms. If assuming linear relation between the runtime and image size and down scaled to full HD (1920x1080) image size, this gives a rough approximation of 101 ms per frame.

TABLE IV. Synthesis results.

primary programming model	OpenCL C
area	0.2mm^2
target clock frequency	600MHz
avg. power with the workload of interest	71mW
ASIC technology	28nm FDSOI

TABLE V. Measured execution times scaled to full HD.

	Mali-T628 MP6	Adreno 330	Intel HD5500	TTA
exec. time (ms)	44.1	76.2	17.6	101.3

TABLE VI. COMPARISON OF POWER CONSUMPTION AND ENERGY-EFFICIENCY.

	Mali-T628 MP6	Adreno 330 330	Intel HD5500	TTA
power (W)	1.57	-	6.5	0.071
energy/ full HD frame (J)	0.070	-	0.115	0.007

To estimate the throughput for the whole BM3D filtering, the BM step is assumed to roughly take the same amount of time as the 3D filtering, according to our experiences. Thus the throughput would be 4.9 frames per second.

The comparison to GPU implementations is presented in Table V. Intel was the fastest with 17.6 ms per frame and Mali second with 44.1 ms per frame. Adreno's performance was 76.2 ms per frame. The proposed accelerator was the slowest with 101 ms. This was expected, since the measured instance is a single-core processor, whereas the other architectures are multicores and contain more processing elements. However, the proposed design is multicore ready as it contains a synchronization unit and connections to a potentially shared memory, and previous experiments with succesful synthesis up to four cores with a 800 MHz target frequency.

As a comparison of the computational performance, the Mali-T628 MP6 has 6 cores, each with an arithmetic pipeline capable of 17 floating point operations per cycle [9]. With its 600 MHz clock frequency this translates to 10.2 GFLOPS per core. For all its six cores, it totals up to 61 GFLOPS. The proposed design only has one core, but has a theoretical peak floating point performance of 16 GFLOPS. The proportion of the peak performances is roughly the same as the execution times in Table V.

To estimate the energy-efficiency, energy per frame was calculated from the runtime and power consumption numbers. Only the GPU power consumption was considered. This is illustrated in Table VI. For TTA, with 71 mW power consumption and 101 ms runtime, energy per frame results in 0.007 J/frame (full HD). Next was Mali, with 0.07 J/frame and Intel with 0.115 J/frame. No tools to measure the Adreno power consumption were available. What is notable is that the proposed TTA's energy consumption is 10 times less than that of the Mali GPU.

IV. RELATED WORK

Not many compute platforms have been evaluated with the BM3D algorithm, or been specifically designed for it. We found only two previous implementations to compare against.

Lebrun [15] implemented a BM3D algorithm on CPU using standard C++. This implementation can be compiled to use multithreading with OpenMP library. DCT is used for the 3D linear transformations.

Sarjanoja et al. [1] implemented a BM3D denoising algorithm on various platforms containing a GPU. They used DCT for the 3D linear transformations. The implementations utilized two different Nvidia GPUs: GTX 650 and Tegra K1. CUDA and OpenCL implementations were evaluated. The authors reported a 7.5x improvement in runtime compared to an existing CPU implementation. However, direct comparison against these is problematic because the reported numbers include the block-matching stage.

V. CONCLUSION

In this paper, a programmable, high performance, low-power transport-triggered architecture ISP was proposed. The implementation area occupation is 0.2mm^2 when synthesized on a 28nm ASIC technology, with a power consumption of 71mW while performing a state-of-the-art image denoising algorithm.

The proposed ISP was evaluated against existing implementations for low power GPUs. The proposed ISP uses 10x less energy compared to the next evaluated implementation per frame for image denoising. This is achieved with the reduced amount of register file traffic and amount of register file ports due to the transport-triggered architecture features. To the best of our knowledge, the proposed design is the most energy-efficient OpenCL C programmable ISP verified with this workload that has been published.

Future work includes further improving the performance of the processor by implementing strided fetch for the image data. The BM3D algorithm operates on the same data both row-wise and column-wise, so if the rows can be fetched in parallel, the columns will be fetched as scalars, if no strided memory is used. Also, an optimized multi-core implementation could be used to reach low-power, real-time operation for real time processing of video streams.

ACKNOWLEDGMENT

The authors would like to thank Finnish Funding Agency for Technology and Innovation (project "Parallel Acceleration 3", funding decisions 1134/31/2015 and 1172/31/2015), ARTEMIS JU under grant agreement no 621439 (ALMARVI) and European Union's H2020 Framework Programme (H2020-MSCA-ITN-2014) under grant agreement n° 642685 MacSeNet.

REFERENCES

[1] S. Sarjanoja, J. Boutellier, and J. Hannuksela, "BM3D image denoising using heterogeneous computing platforms," in *Proceedings of Conference on Design and Architectures for Signal and Image Processing*, Cracow, Poland, Aug. 23-25 2015, pp. 1–8.

[2] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22Nd ACM International Conference on Multimedia*, Orlando, Florida, USA, 2014, pp. 675–678.

[3] V. W. Lee, C. Kim, J. Chhugani, M. Deisher, D. Kim, A. D. Nguyen, N. Satish, M. Smelyanskiy, S. Chennupaty, P. Hammarlund, R. Singhal, and P. Dubey, "Debunking the 100x GPU vs. CPU myth: An evaluation of throughput computing on CPU and GPU," in *Proceedings of the 37th Annual International Symposium on Computer Architecture*, Saint-Malo, France, 2010, pp. 451–460.

[4] H. Corporaal, *Microprocessor Architectures: From VLIW to TTA*. Chichester, England: John Wiley & Sons, Ltd., 1998.

[5] G. J. Lipovski, "Architecture of a simple, effective control processor," in *Proceedings of Symposium on Micro Architecture*, 1976, pp. 187–194.

[6] H. Corporaal and H. J. Mulder, "Move: A framework for high-performance processor design," in *Proceedings of Conference on Supercomputing*, Albuquerque, NM, Nov. 18-22 1991, pp. 692–701.

[7] J. Hoogerbrugge and H. Corporaal, "Register file port requirements of transport triggered architectures," in *Proceedings of Annual International Symposium on Microarchitecture*, November-December 1994, pp. 191–195.

[8] O. Esko, P. Jääskeläinen, P. Huerta, C. S. de La Lama, J. Takala, and J. I. Martinez, "Customized exposed datapath soft-core design flow with compiler support," in *Proceedings of International Conference on Field Programmable Logic and Applications*, Washington, DC, Aug. 31-Sep.2 2010, pp. 217–222.

[9] P. Harris. The Mali GPU: An abstract machine, part 3 - the Midgard shader core. Accessed: 2016-08-04. [Online]. Available: https://community.arm.com/groups/arm-mali-graphics/blog/2014/03/12/the-mali-gpu-an-abstract-machine-part-3–the-shader-core

[10] T. Viitanen, H. Kultala, P. Jääskeläinen, and J. Takala, "Heuristics for greedy transport triggered architecture interconnect exploration," in *Proceedings of International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, Uttar Pradesh, India, Oct. 12-17 2014, pp. 1–7.

[11] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian, "Image denoising by sparse 3-D transform-domain collaborative filtering," *Transactions on Image Processing*, vol. 16, no. 8, pp. 2080–2095, Aug. 2007.

[12] *OpenCL Specification v1.2r15*, Khronos Group, Nov. 2011. [Online]. Available: http://www.khronos.org/registry/cl/

[13] 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.

[14] TechPowerUp. GPU-Z. Accessed: 2016-08-12. [Online]. Available: https://www.techpowerup.com/gpuz

[15] M. Lebrun, "An analysis and implementation of the BM3D image denoising method," *Image Processing On Line*, vol. 2, pp. 175–213, Aug. 2007.