Domain Adaptive Processor Architectures

Florian Fricke*, Safdar Mahmood[†], Javier Hoffmann*, Muhammad Ali[‡], Keyvan Shahin*, Michael Hübner[†], Diana Göhringer[‡]

* Chair for Embedded System for Information Technology

Ruhr-Universität Bochum, Germany

{florian.fricke; javier.hoffmann; keyvan.shahin}@rub.de

[†] Chair for Computer Engineering

BTU Cottbus - Senftenberg, Germany

{safdar.mahmood; michael.huebner}@b-tu.de

[‡] Chair of Adaptive Dynamic Systems, TU Dresden, Germany

{muhammad.ali; diana.goehringer}@tu-dresden.de

Abstract. The ongoing megatrends in industry and academia like the Internet of Things (IoT), the Industrial Internet of Things (IIoT) and Cyber-Physical Systems (CPS) present the developers of modern computer architectures with various challenges. A novel class of processors which provide more data throughput with a simultaneously tremendously reduced energy consumption are required as a backbone for these “Things”. Additionally, the requirements of CPS like real-time, reliability, dependability, safety, and security are gaining in importance in these applications. This paper shows a brief overview of novel processor architectures providing high flexibility to adapt during design- and runtime to changing requirements of the application and the internal and external system status.

1 Introduction

In the domain of low power, high performance embedded systems, the usage of dynamic reconfigurable hardware has been proven as highly efficient for signal and data processing in several application domains [SCH15] and [RBG17]. Reconfigurable hardware is indeed not that energy- and computationally-efficient as application specific integrated circuits (ASIC), but has the advantage, that adaptivity during runtime in soft- and hardware can be exploited [GRI15]. Therefore, algorithms realized in software as well as in hardware can be adapted according to the demands of the current point of operation, which can be functional (e.g., precision) or non-functional (e.g., reliability, real-time capability, safety, and security) [JAN15]. In this paper, we aim to present an overview considering current developments in the field of design- and runtime-adaptive architectures for data processing. These include custom processor designs, but also other architectures that can serve as co-processors or accelerators. Adaptivity can be highly beneficial in the whole application area of processors starting with ultra-low-power designs in the embedded market, going through reliable processors for cyber-physical systems up to high-performance systems in the HPC domain.

The first section covers processor designs which features can be adapted to the application domain at design time, in the following section runtime reconfigurable architectures are presented. Some architectures from the first section have the ability to change a subset of their parameters at run-time, detailed information is given in the related sections. The article concludes with a summary and an outlook on current research topics in this domain.

2 Design-Time Reconfigurable Architectures

2.1 Ultra-Low Power Microcontrollers for Real-Time Applications

Major advantages of General Purpose Processors (GPPs) are, among others, their low cost and variety of possible scenarios where a device can be implemented. GPPs are used widely, for example, as the intelligence responsible for collecting and processing sensor data, however, advanced scenarios require also that sensors are able to respond under constraints of hard real-time deadlines, mainly if safety considerations are targeted. Considering the limitations of batteries, this is proven to be a complex mission. Even though modern microcontrollers are able to work in the, so-called, ultra-low-power modes, as for example the MSP432 from Texas Instruments [MSP18], the energy stored in battery cells is still finite. Thus, the manner on how tasks are scheduled has to be dependent on the fact that energy is not infinite, and that safety tasks have to be carried out independently on the current state. One solution to this problem is to dynamically modify the scheduler, in order to save energy for the really important tasks.

Although GPPs have a decent price-performance ratio, several tasks require more dedicated hardware, which is able to perform specific tasks with more computational efficiency. However, a hardware device designed for and dedicated to only one purpose is a high price to pay and carries a particular risk: in case the task varies even in small proportion the hardware is not useful anymore.

Therefore, reconfiguration is a viable approach to enable special purpose hardware to encounter different problems. Reconfiguration is thus a key research area, and the following sections will give an insight into these technologies.

2.2 The Tensilica XTensa Processor

The XTensa Processor, which is commercially available from Cadence is a processor, that consists of a core-ISA, which is to be found in all different kinds of Xtensa processors and a considerable number of custom modules, that can be added to the processor to fit the application requirements. Basic information on the ISA design can be found in [GON00]. [CAD16] provides an overview of the features of the current processor generation. Many elements of the processor core can be customized. These include various features of the ISA like the availability of specific arithmetic instructions, the number of general-purpose registers, the length of the processor pipeline and the existence respectively the complexity of a memory management unit. Furthermore, it is possible to customize the memory interface including the availability and the configuration of

cache memory and additional interfaces to attach external hardware. These customizations can be done within an IDE. The feature, which allows a developer to build a domain-specific XTensa core is the option to develop custom instruction set extensions using the TIE-language. Using TIE, an XTensa core's instruction set can be extended by custom hardware which is directly accessible through assembly instructions. In comparison to external processor accelerators, domain-specific processor instructions have some advantages: they share the processor core's state and can therefore directly access all registers, and the memory. The direct integration into the processors ISA makes using the instructions much cheaper regarding time (cycles) and data transfer.

However, there are also some disadvantages: First, the complexity of a single instruction is limited, because a too complex instruction would reduce the maximum clock speed of the processor. This is caused by the fact, that the custom instructions are integrated into the processor pipeline and the execution of the instruction must fit into the execute stage. However, it is possible to spread an instruction over more than one cycle, but this introduces more complexity. The second drawback is that the processor still has to control the execution whereas external accelerators (or co-processors) would allow the processor to really offload the calculation.

The ability to seamlessly integrate custom instructions and the significant number of available coprocessor IP make the XTensa processor an interesting candidate in comparison to the diversity of available processor IP.

The IDE enables hardware-software codesign, as the processors' configuration and the software are developed within the same environment. Therefore, the software can easily be tested on every processor configuration using the Instruction Set Simulator (ISS) or the Cycle-Accurate Simulator which are both integrated into the IDE. Furthermore, the IDE supports the hardware and the software developer to iteratively improve the designs created using feedback mechanisms, which take analysis results from previous stages to automatically improve the code generation. A brief overview of the IDE's features can be found in [CAD14].

2.3 RISC-V Cores

Design-time configurable architectures can be configured before deployment on FPGA and can be optimized with respect to the application. These types of architectures are very helpful in building prototypes fast and reducing the time to market of a design.

RISC-V is an open source instruction set architecture (ISA) developed by the University of California, Berkeley. The RISC-V project initially started for research and academics purpose but quickly gained popularity among available open source ISAs. RISC-V now has a big ecosystem (soft-processors, simulators, etc.) and companies like Microsoft, Nvidia, Qualcomm, etc. [RIS18] are now supporting the project. RISC-V supports 32-bit and 64-bit processor architectures. It also supports highly-parallel multi-core implementations. The main feature of RISC-V is that it supports custom instruction extensions. This means, that designers can add their own custom instructions to the processor architecture and define its functionality on hardware. RISC-V also comes with a free available toolchain (simulator, compiler, etc.). Custom instructions can also be added to the toolchain and used for custom architectures.

There are a lot of RISC-V implementations available. Some popular implementations are e.g. Rocket-chip [MOR16], RI5CY [MIC17] and TAIGA [MAT17]. TAIGA's implementation is very promising because it focuses on reconfigurable implementation depending on the application.

TAIGA is a RISC-V soft-processor introduced by Simon Fraser University. It is a 32-bit processor, which supports multiply/divide and atomic instruction extensions (RV32IMA) of RISC-V. In comparison to some competitors which also implement the RISC-V instruction set like Rocket-chip and BOOM, TAIGA is a smaller implementation. Nevertheless it is designed to support Linux-based shared-memory systems. Taiga processor is implemented in SystemVerilog and is highly configurable. It enables users to configure caches, TLBs, choice of bus standard and inclusion of multiply/divide operations in the processor. The Taiga processor pipeline details and components are shown in Figure 1. The Taiga pipeline structure is built with multiple independent execution units, and these execution units have no restrictions on latency. This design technique enables adding new functional units to the processor. In RISC-V, instruction extensions are not mandatory and can be added if necessary, as per application of the processor. Taiga's implementation exploits this and presents configurable options for the processor.

Fig. 1. TAIGA architectural overview [MAT17]

Three configurations (minimal, comparison and full) of Taiga are presented in [MAT17], and their comparison is given in Table 1. Minimal configuration is without multiply/divide execution unit, TLBs/MMUs, and caches. Comparison configuration has 2-way 8KB data and instruction cache and multiply- and divide- instructions support. This configuration was used as a comparison with Rocket and LEON3 processors. Moreover, in full configuration Fetch and Load Store Units share 4-way 16 KB data cache and 32 KB of local memory. The full configuration is approximately 1.5 times

larger than minimal configuration. LEON3 is about 1.5 times bigger as compared to Taiga comparison configuration, but Taiga can be clocked approximately 39% faster and is also more flexible in the pipeline.

Table 1. Resource comparison of the TAIGA configurations [MAT17]

	LUTs	FFs	Slices	BRAMs	DSPs	Freq(MHz)
minimal	2614	1592	906	9	0	111
comparison	3998	2942	1371	10	4	104
full	4079	2959	1394	32	4	103

2.4 Many-Core Vision Processor

Another highly efficient hardware architecture, especially for algorithms in the domain of sensor data fusion and machine learning [MÖN16], was published in [JUD17]. In this work, a new concept with distributed arithmetic logic units (ALUs) is used to process image and signal data maximal parallelized. The concept even allows to inject signal data directly into the data path of the hardware with the effect to achieve lowest latencies and increased data throughput. This architecture is runtime adaptive as well and highly capable of hosting machine learning- and sensor fusion algorithms, since the hardware structure supports the “architecture” of the algorithms directly.

The design space exploration of the processor’s communication structure and the spatial algorithm distribution on the many-core architecture is discussed in [JUD16]. Within this work, the methodology for automatically generating a SystemC based simulator for the hardware architecture as well as a set of tools to distribute image processing algorithms onto this many-core architecture simulator are presented. These tools have been utilized to explore different implementation alternatives before deciding the implementation details of the implemented architecture.

2.5 FGPU

Recently, a new overlay architecture, called FPGA – GPU (FGPU) was developed [ALK18]. The benefit of this hardware architecture is that it is fully compliant to a general-purpose GPU in terms of OpenCL compatibility but comes with the feature of runtime adaptivity [ALK17]. Furthermore, this hardware architecture is not related to a specific FPGA hardware, and even more, in case of increased computational efficiency, it can be realized even as an ASIC, with the drawback of the loss of hardware adaptivity features. However, the core still stays software parameterizable.

GPUs are considered to be one of the favored platforms for achieving high performances, especially when executing the same operation on a large set of data. This work introduces the means for generating a scalable soft GPU on FPGA which can be configured and tailored for each specific task. The design is written using pure VHDL and

does not use any IP Core or FPGA primitives. The main advantage of having a soft GPU on FPGA can be stated as the fact that the GPU hardware can be configured optimally for running a specific task or algorithm.

The execution model of the GPU is a simplified version of the OpenCL standard. Index spaces of maximum three dimensions are defined for each task to be executed. In each dimension, the index space can have a depth of up to 232 and this depth must be a multiple of work-group size. Work-groups can have up to 512 work-items, and each work-item is defined based on its 3D tuple in the index space.

Each work-item can use its private memory consisting of 32, 32-bit registers or address a global shared memory of maximum size of 4GB. As can be seen in Figure 2, each FGPU consists of several Compute Units (CUs), each having a single array of PEs. Both data transfer and platform control are done using different AXI4 interfaces. The tasks to be executed are stored in the Code RAM (CRAM) and the other information unknown at compile time, have to be stored at Link RAM (LRAM). A Wavefront Scheduler assigns each group of 64 work-items to an array PE. The run-time memory stores the information needed for these work-items during the execution. CU memory controllers are connected to a global memory controller to let the PEs access the global memory.

Fig. 2. FGPU: Schematic overview

3 Run-Time Reconfigurable Architectures

3.1 GPP/ASICs/ASIPs

ASICs and GPP both have their advantages and disadvantages. ASIC can provide ultra-high performance, but lack flexibility and programmability and GPP can provide more programmability but lack competitive performance. A design-time configurable and runtime reconfigurable embedded processor is published in [SOU18]. The processor aims to deliver efficient performance at low energy consumption. For this, a configurable embedded processor is coupled with reconfigurable fabric. This system resulted in improvement of performance by 1.4x and reduced energy by 60% as compared to a configurable processor, but this results in more area.

A reconfigurable ASIC-like implementation is presented in [YAN17] using a reconfigurable mathematic processing unit (MaPU). The experiments show good computing performance and an increase in flexibility. The reconfigurable control unit, based on a

Finite State Machine (FSM), is used to control the bus interface and the computing unit. A reconfigurable processing accelerator with SIMD structure for parallel acceleration is also implemented. Another ASIC design of a low-power reconfigurable FFT Processor is presented in [LIU07]. ASIC design was implemented using Synopsys EDA tools. The aim of this ASIC design implementation was to achieve high speed, low power, and reconfigurability.

[MOC12] introduces an application specific instruction set processor (ASIP) targeting image processing applications. The ASIP design presented, has been developed to find a good trade-off between programmability and performance in the field of image processing applications. Performance evaluation showed, that the proposed ASIP performed better in image scaling and image enhancement when compared to an ARM946E processor or a TMS320DM6437 DSP. Furthermore, it also provides the flexibility to implement several signal processing algorithms as opposed to an ASIC.

3.2 VCGRA Approach

As a special kind of application-specific accelerators based on ASICs or FPGAs, Coarse-Grained-Reconfigurable-Arrays have huge advantages, when combined with a General Purpose Processor (GPP). In comparison to an FPGA, the granularity of both, the processing elements operations, and the communication infrastructure cannot be controlled on bit-level, like in traditional FPGA architectures. This approach reduces the level of flexibility but also lowers the effort for creating configurations (CAD-process) and for reconfiguring the functionality at run-time, due to the reduced size of the bitstreams and the lower complexity of the system's state. Many different approaches for coarse-grained compute architectures have been proposed, most of them suited for specific workloads and providing significant speedups. Nevertheless most architectures did not have commercial success. The reason for the commercial failure allows various explanations: First of all, these specific architectures have significant advantages only for a limited number of applications. In other cases, the additional effort required cannot be justified. Furthermore, the designer cannot rely on support by well-engineered tools for the creation and the configuration of many novel architectures, in comparison to less efficient but more widespread architectures.

For this reason, a toolchain covering both, the generation of the CGRA as well as the implementation of algorithms on top of these architecture have been developed. The CGRA is realized as an overlay architecture on top of commercial FPGAs. The software tools derive configurations for the overlay architecture from algorithm-descriptions in high-level programming languages. The architecture is called Virtual Coarse-Grained Reconfigurable Array (VCGRA) as it exists as an intermediate level between the accelerated algorithm and the FPGA-hardware. The structure of the architecture is presented in [FRI18]. It consists of alternating levels of processing elements and so-called virtual channels, which control the dataflow. In comparison to other CGRA architectures, many aspects are configurable: the size and shape of the array which includes the number of in- and outputs, the widths of inputs and outputs, the number of array layers and the number of processing elements (PEs) for each layer. The bitwidth of the arithmetic units and the connections as well as the operations, which are provided

by the PEs, can also be set at design time whereas the connections within the layers and the operations carried out by the PEs are run-time-(re)configurable. The hardware part of the tool-flow generates the FPGA-bitstream of this architecture, including a wrapper, providing the interface to, e.g. an embedded CPU within a SoC as well as the required HW/SW interface. The software part of the toolchain takes the parameters of the VCGRA-architecture and the description of the algorithm as input and creates the configuration for the overlay architecture. Furthermore, a template for a Linux application is provided and adapted for running the hardware-accelerated application on the target system. The whole toolchain is depicted in Figure 3.

Accelerators based on this architecture always require a GPP as they are currently not able to compute all kinds of algorithms. Especially control flow can only be realized by computing both parts of a branch and ignoring the part which is not required. Especially for accelerating compute-intensive applications on huge data-streams coarse-grained arrays can be an interesting addition to a GPP. The fact, that the structure, as well as the, compute units can be configured/tailored to the applications demands enables the adaption of a generic CPU to a specific application domain by using CGRA-based accelerators.

Fig. 3. Overview VCGRA toolflow

3.3 Dynamic Cache Reconfiguration

In the area of hardware reconfiguration, special interest has been dedicated to dynamic adaption of the cache memory. The reason for is based on the important influence of the cache memory on the latency of data transfers between the processor and the memory. Furthermore, caches are responsible for a significant amount of a processor's energy consumption [MAL00]. On this regard, several methodologies were presented using the so-called, Dynamic Cache Reconfiguration (DCR). Navarro et al. [Nav16] propose two online heuristics to improve overall computing performance targeting soft real-time systems. After the tasks are selected by the scheduler, an online profiler analyzes whether a cache reconfiguration is necessary by observing the values of cache hits and cache miss. The first heuristic takes the largest cache to be analyzed first and decreases, tracking the influence on the performance. The second approach softly modifies the previous one, at the same time increasing block size and associativity.

Navarro et al. [NAV17] proposed a methodology to reconfigure the cache configuration regarding associativity and size by analyzing the relative frequency of the dynamic instructions. This is done using machine learning and aims to select the most energy efficient configuration for a determined program. The results obtained are promising and show that such an approach is effective. Even when the algorithm selected a non-optimal configuration, for the majority of the cases the energy consumed was still under acceptable parameters. The authors finally propose to extend the configurations selected and take a further step in order to detect the proper cache completely dynamically just with a percentage of the code analyzed and also to consider the L2 level of cache.

3.4 Run-Time Reconfiguration in Machine Learning and ANN Implementations

Artificial Neural Networks (ANNs) are becoming highly complex as the research in this area progresses. Moreover, training algorithms in machine learning specifically for CNNs in the family of DNNs [SCH15] are enormously compute-intensive as the number of neurons and weighted interconnections between each layer is very high. This makes implementation of ANNs on embedded platforms very challenging especially when it comes to run-time adaptation. Considering DNN as an example, there are two procedures involved; 'Training' and 'Inference'. A training phase or training algorithm requires a neural network to respond to the inputs iteratively depending on the size of the available training data-set, consequently adjusting the weights of the effective interconnections between each layer. Inference on the other hand is the after-training execution of a neural network with the available real-world data or test data.

Adaptive reconfigurable architectures provide a wide range of possible applications in the field of machine learning, especially in realizing artificial neural networks on embedded platforms; for example, DNN inferencing as well as for training neural networks. In this regard, there has been a variety of proposed architectures and designs using reconfigurable hardware [HOF17]. An implementation which relates to runtime

reconfigurable architectures, a bit-width adaptive convolution processor for CNN acceleration is proposed [GUO18]. The motivation of this work [GUO18] is coping with the problem of high resource usage in convolutional layers when each convolution layer might require a different bit-width across the whole network. It uses the idea of partitioning the DSP resources of Xilinx into different bit-width parts for the data as well as weights which would be used for performing convolution inside a CNN. Different convolution processors are associated with different bit-width-wise segregated DSP partitions, which inference the CNN layer in parallel. Each layer adapts to its appropriate DSP partition in order to compute the convolution. In [GUO18], the proposed design offers higher throughput, as well as optimized DSP utilization by using the idea of dynamic bit-width allocation to each CNN layer.

In the case of low-density neural networks other than deep neural networks and recurrent neural networks, the idea of on-target run-time reconfiguration for training is also demonstrated in an implementation [ELD84] done back in 1994. [ELD94] uses Xilinx device X12 to implement the run-time reconfigurable artificial neural network architecture, named as RRANN in [ELD94]. The designed architecture in [ELD94] lays out the backpropagation training algorithm into three stages for reconfiguration, 1. Feed-forward, 2. back-propagation and 3. update. Each stage has its exclusive neural processor which is configured on to an FPGA when it is needed. The feed-forward neural processor in [ELD94] takes the input data and propagates it to the outputs through neurons. Then RRANN reconfigures the FPGA with back-propagation circuit module to find errors in the output layers and back-propagates to each neuron in the hidden layer with a new error value. Finally, the FPGA is reconfigured during runtime to the update stage in order to change the number of weights suggested by error values in the previous ‘back-propagation stage.’

In recent years, advancements in the machine learning field has also posed some challenges in the area of training an ANN like DNNs and RNNs on embedded hardware. A lot of approaches use offline training with the available dataset and online inferencing of neural networks e.g. on an embedded target with an option of runtime reconfigurable weights [HOF17]. Although, ANN implementations with extensive runtime adaptivity and reconfigurability such as in [GIN11] have also been realized. In [GIN11], a reconfigurable back-propagation neural network architecture has been proposed which can be configured using a 31-bit configuration register. The architecture with its own dataflow and control signals, contains 5 functional blocks which are program memory, forward propagation block, back-propagation block, weight array and control unit. In this work [GIN11], the 31-bit configuration register can be programmed for modification of different parameters of the neural network e.g., learn/recall, number of layers, number of neurons and iteration. The work has been implemented on Xilinx devices Virtex-4 XC4VLX60 and tested for non-linear function prediction.

In the implementation [SAJ18], an FPGA based co-processor has been designed to fulfill the compute-intensive training task for deep convolutional neural networks. The proposed reconfigurable co-processor is inferred on a Xilinx Ultrascale FPGA Kintex XCKU085 and can be reconfigured just by using the provided functionality parameters in the Block Memory. The implementation uses PCIe on a host computer to connect to

the FPGA and to transfer the training data. The proposed co-processor architecture follows a similar approach as in [GIN11], with three separate engines for the back-propagation algorithm. The forward propagation engine in [SAJ18] takes the input for MAC-based computations in FPGA DSP slices and predicts the classification on the basis of input feature. Then, the delta propagation engine distributes back all the errors that are computed in the output layer of the CNN. The parameter update engine is used to modify and update the neural network's weights and biases in convolutional layers. Each iteration uses all three engines until the desired weight values and biases are achieved. The co-processor proposed in [SAJ18] can perform training task on different neural network structures by reconfiguration data specified in BRAM of the FPGA. It offers different adaptations for different image sizes for CNN, neural network architecture and learning rates with a maximum throughput of 280 GOP/s.

[YIN17] implements a reconfigurable hybrid neural network processor architecture. The work proposes three optimization techniques, including bit-width adaptation for meeting needs of different neural network layers which is similar to the methodology described in [GUO18]. The other two techniques are on-demand array partitioning and usage of pattern-based multi-bank memory. According to [YIN17], this leads to an improved processing element (PE) utilization of 13.7% and computational throughput by a factor of 1.11 for parallelization of neural networks, and data re-use, respectively.

3.5 Dynamic Partial Reconfiguration

Dynamic partial reconfiguration (DPR) has added new benefits in system design. DPR reduces configuration time and helps to save memory since partial reconfiguration bitstreams are smaller than full bitstreams. DPR permits reconfiguration of a part of a system while the rest of the FPGA is still running. This technique helps to improve dynamic system adaption, reliability and reduces system cost [KIZ18]. DPR can be used for machine learning techniques. Since FPGAs and ASICs are a practical replacement for GPUs for DNN implementations, especially when there are strict power and performance constraints.

[FLO18] proposed a hardware/software co-design technique to explore DPR technique to accelerate Convolution Neural Networks (CNN). This approach helped to achieve 2.24 times faster execution as compared to a pure software solution. It also allows to implement larger CNN designs with larger and more layers in hardware and reconfigure the whole network.

A runtime reconfigurable fast Fourier transform (FFT) accelerator integrated with a RISC-V core is presented in [WAN17]. Although this paper is focused on LTE/Wi-Fi but the integration of a runtime configurable accelerator with an open source RISC-V core is very promising.

In [HUS12], DPR technique was used for K-Nearest neighbor (KNN) classifier to change a specific kernel while rest of KNN remained unchanged. This KNN which used DPR offered 5x speed-up as compared to an equivalent KNN implementation running on a General Purpose Processor (GPP).

[ARI18] implements low power techniques for image processing applications on FPGAs. The main contribution in this paper is the implementation of dynamic voltage

scaling, dynamic partial reconfiguration, and a debugging system for image processing functions. DPR implementation helped reducing power consumption, increased resource sharing and enabled the user to change filter behavior without halting the complete system.

4 Conclusion

This paper provides an overview of selected processor architectures with configuration features during design- and runtime. Some architectures support the parameterization during design time which enables an adaption according to application demands. After an in-depth exploration of the design space, the parameters for a domain-specific architecture can be derived from the requirements of the envisioned use-case. This is a step forward in modern processor design as it leads to better utilization of hardware. However, in times of increased complexity in even smallest sensor devices, the application requirements cannot be predicted anymore. In larger scale, e.g. in the automotive domain, novel advanced driver assistance systems need to be reactive to changes in the application requirements and even to certain situations during the operation. Novel System-on-Chip architectures like e.g. Zynq Ultrascale MPSoC [XIL18], can only be utilized efficiently, when adaptivity during runtime is exploited. A static design implementation with a parameterization during design time cannot lead to a beneficial exploitation of resources on the chip. Therefore, adaptivity in general and specifically, in the processor architecture is required to fulfill the demands of future high performant and low power cyber-physical systems.

5 Future Research

Adaptive and reconfigurable architectures in the domain of machine learning specifically for ANNs on embedded systems still need significant improvements. Existing implementations and designs have limited or partial adaptivity, and reconfigurability as the field of deep learning is growing tremendously fast. As compared to adaptive architectures for ANN inferencing task, there has been less amount of work in the area of adaptively and dynamically reconfigurable architectures for training the ANNs, which hints us about a broader possibility of further investigation and research.

Machine learning can also be an approach to improve a processor's components such as the cache or the way how the scheduler performs. In the first case, the cache configuration with the lowest energy consumption could be selected before a program is executed, also improving the overall system performance. In the second case, the scheduler could be trained to select the tasks so that the system performance increases, for example, by reducing the cache reconfiguration mentioned previously.

Reconfigurable accelerators based on virtual overlay architectures that can be attached to processors show a significant potential: they can be used to develop highly efficient accelerators for data-flow applications with a slightly reduced flexibility compared to an FPGA solution, but with a heavily reduced design effort. In upcoming work,

the flexibility and the efficiency of the presented tool-flow will be improved and evaluated on more application scenarios.

RISC-V is an exciting topic nowadays. It can be used to build a GPP, and since it allows to add custom instructions to the ISA and toolchain, it can be used as an ASIP. A designer can add custom instructions and define its hardware functionality which suits the application. RISC-V implementation provides more flexibility as compared to other available soft processors, and this is an open area for research right now for machine learning and signal processing applications.

Even though the innovation in the field of special purpose hardware modules is making them more attractive for implementation in a variety of areas, this does not mean, that the end of General Purpose Processors is close. On the contrary, GPPs still provide the most practical solution for tasks that do not require high processing capabilities at low costs. Furthermore, the innovation in the development of GPPs is also not stagnating. Innovative developments will continue to be presented in this area as well. Current innovations in this area are e.g. the utilization of novel heuristics or machine learning to enhance the capabilities of low energy devices like sensor nodes. In this case the battery life can be extended, which could solve a major problem in logistics.

References

- [ALK17] M. Al Kadi, B. Janssen, M. Hübner, “Floating-Point Arithmetic Using GPGPU on FPGAs”. In Proc. of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Bochum, 2017, pp. 134-139. doi: 10.1109/ISVLSI.2017.32.
- [ALK18] M. Al Kadi, B. Janssen, J. Yudi, M. Hübner, “General Purpose Computing with Soft GPUs on FPGAs”. ACM Transactions on Reconfigurable Technology and Systems (TRETs), Volume 11 Issue 1, January 2018, ACM New York, NY, USA.
- [ARI18] A. Podlubne, J. Haase, L. Kalms, G. Akgün, M., H. ul Hassan Khan, A. Kamal, D. Göhringer, “Low Power Image Processing Applications on FPGAs using Dynamic Voltage Scaling and Partial Reconfiguration”. DASIP 2018 - Conference on Design and Architectures for Signal and Image Processing
- [CAD14] Cadence, Tensilica Datasheet of the Xtensa Processor Developer’s Toolkit, © Cadence 2014, Source: <https://ip.cadence.com/uploads/102/HWdev-pdf> (2018-11-14)
- [CAD16] Cadence, Tensilica Datasheet of the Xtensa LX7 Processor, © Cadence 2016, Source: https://ip.cadence.com/uploads/1099/TIP_PB_Xtensa_lx7_FINAL-pdf (2018-11-14)
- [ELD94] J. G. Eldredge and B. L. Hutchings, “Density Enhancement of a Neural Network Using FPGAs and Run-Time Reconfiguration”. Proceedings of IEEE Workshop on FPGA’s for Custom Computing Machines, DOI: 10.1109/FPGA.1994.315611
- [FLO18] F. Kästner, B. Janßen, F. Kautz, M. Hübner, G. Corradi, “Hardware/Software Codesign for Convolutional Neural Networks exploiting Dynamic Partial Reconfiguration on PYNQ”. 2018 IEEE International Parallel and Distributed Processing Symposium Workshops

- [FRI17] F. Fricke, A. Werner, M. Hübner, "Tool flow for automatic generation of architectures and test-cases to enable the evaluation of CGRAs in the context of HPC applications", in 2017 Conference on Design and Architectures for Signal and Image Processing (DASIP)
- [FRI18] F. Fricke, A. Werner, k. Shahin, M. Hübner, "CGRA Tool Flow for Fast Run-Time Reconfiguration" In Proc. of the 14th International Symposium on Reconfigurable Computing: Architectures, Tools, and Applications (ARC), Santorini, Greece, May 2018
- [GIN11] Gin-Der Wu, Zhen-Wei Zhu, Bo-Wei Lin, "Reconfigurable Back Propagation Based Neural Network Architecture". 2011 International Symposium on Integrated Circuits, doi: 10.1109/ISICir.2011.6131881
- [GON00] R. E. Gonzalez, "Xtensa: a configurable and extensible processor," in IEEE Micro, vol. 20, no. 2, pp. 60-70, March-April 2000.
- [GRI15] T. Grimm, B. Janßen, O. Navarro, M. Hübner, "The value of FPGAs as reconfigurable hardware enabling Cyber-Physical System". In Proc. of the IEEE 20th Conference on Emerging Technologies and Factory Automation (ETFA), Luxembourg, 2015, pp. 1-8. doi: 10.1109/ETFA.2015.7301496.
- [GUO18] J. Guo, S. Yin, P. Ouyang, F. Tu, S. Tang, L. Liu and S. Wei, "Bit-width Adaptive Accelerator Design for Convolution Neural Network". 2018 IEEE International Symposium on Circuits and Systems (ISCAS), doi: 10.1109/ISCAS.2018.8351666
- [HOF17] J. Hoffmann, O. Navarro, F. Kastner, B. Janßen, M. Hübner, "A Survey on CNN and RNN Implementations". 2017 The Seventh International Conference on Performance, Safety and Robustness in Complex Systems and Applications
- [HUS12] H. M. Hussain, K. Benkrid, H. Seker, "An Adaptive Implementation of a Dynamically Reconfigurable K-Nearest Neighbour Classifier On FPGA". 2012 NASA/ESA Conference on Adaptive Hardware and Systems (AHS-2012)
- [JAN15] B. Janßen, F. Schwiegelshohn, M. Hübner, "Adaptive computing in real-time applications". In Proc. of the IEEE 13th International New Circuits and Systems Conference (NEWCAS), Grenoble, 2015, pp. 1-4. doi: 10.1109/NEWCAS.2015.7182057.
- [JUD16] J. Yudi, A. Werner, A. Shallufa, F. Fricke, M. Hübner, "A Design Methodology for the Next Generation Real-Time Vision Processors" in Proc. of International Symposium on Applied Reconfigurable Computing, ARC 2016, Rio de Janeiro, Brazil
- [JUD17] J. Yudi, C. H. Llanos, M. Hübner, "System-level design space identification for Many-Core Vision Processors". Elsevier Journal on Microprocessors and Microsystems <https://doi.org/10.1016/j.micpro.2017.05.013>
- [KIZ18] K. Vipin, Suhaib A. Fahmy, "FPGA Dynamic and Partial Reconfiguration: A Survey of Architectures, Methods, and Applications". ACM Comput. Surv. 51, 4, Article 72 (July 2018)
- [LIU07] G. Liu, Q. Feng, "ASIC design of low-power reconfigurable FFT processor". 2007 7th International Conference on ASIC
- [MÖN16] U. Mönks, H. Dörksen, V. Lohweg, M. Hübner, "Information Fusion of Conflicting Input Data". Sensors, ISSN 1424-8220 MDPI AG (Multidisciplinary Digital Publishing Institute), Aug 2016.
- [MAL00] A. Malik, B. Moyer, and D. Cermak, "A low power unified cache architecture providing power and performance flexibility," in Low Power Electronics and Design, 2000.

- ISLPED'00. Proceedings of the 2000 International Symposium on. IEEE, 2000, pp. 241–243
- [MAT17] E. Matthews, L. Shannon, “TAIGA: A new RISC-V soft-processor framework enabling high performance CPU architectural features”. 2017 27th International Conference on Field Programmable Logic and Applications (FPL)
- [MIC17] M. Gautschi, P. Davide Schiavone, A. Traber, I. Loi, A. Pullini, D. Rossi, E. Flamand, F. K. Gürkaynak, and L. Benini, “Near-Threshold RISC-V Core With DSP Extensions for Scalable IoT Endpoint Devices”. IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, VOL. 25, NO. 10, OCTOBER 2017
- [MOC12] M. Asri, Hsuan-Chun Liao, T. Isshiki, D. Li, H. Kunieda, “A Reconfigurable ASIP-based Approach for High Performance Image Signal Processing”. 2012 IEEE Asia Pacific Conference on Circuits and Systems
- [MOR16] M. Nöltner-Augustin “RISC-V — Architecture and Interfaces The RocketChip”. ADVANCED SEMINAR “COMPUTER ENGINEERING”, UNIVERSITY OF HEIDELBERG WT16/17
- [MSP18] MSP432P401R SimpleLink Ultra-Low-Power 32-Bit ARM Cortex-M4F MCU With Precision ADC, 256KB Flash and 64KB RAM. © Texas Instruments, 2018, Source: <http://www.ti.com/product/MSP432P401R>
- [NAV17] O. Navarro Guzman, J. E. Hoffmann, F. Stuckmann, M. Hübner, Jones Y. M. Alves da Silva, “A Machine Learning Methodology For Cache Recommendation”, 2017, 13th International Symposium on Applied Reconfigurable Computing (ARC2017)
- [RBG17] J. Rettowski, A. Boutros, D. Göhringer, “HW/SW Co-Design of the HOG algorithm on a Xilinx Zynq SoC”. J. Parallel Distrib. Comput. 109: 50-62 (2017).
- [RIS18] RISC-V “RISC-V Member Directory”. Source: <https://riscv.org/membership/>
- [SAJ18] S. Remi Clere, S. Sethumadhavan, K. Varghese, “FPGA Based Reconfigurable Co-processor for Deep Convolutional Neural Network Training”, 2018 21st Euromicro Conference on Digital System Design (DSD). doi: 10.1109/DSD.2018.00072
- [SCH15] J. Schmidhuber, “Deep learning in neural networks: An overview”. Neural networks 61 (2015): 85-117
- [SOU18] J.D. Sousa, A. L. Sartor, L. Carro, M.B. Rutzig, S. Wong, A.C.S. Beck, “DIM-VEX: Exploiting Design Time Configurability and Runtime Reconfigurability”. The 14th International Symposium on Applied Reconfigurable Computing (ARC 2018)
- [WAN17] A. Wang, B. Richards, P. Dabbelt, H. Mao, S. Bailey, J. Han, E. Chang, J. Dunn, E. Alon, B. Nikolić, “A 0.37mm² LTE/Wi-Fi Compatible, Memory-Based, Runtime-Reconfigurable 2n 3m5k FFT Accelerator Integrated with a RISC-V Core in 16nm FinFET”. IEEE Asian Solid-State Circuits Conference November 6-8, 2017/Seoul, Korea
- [XIL18] Xilinx Inc. UltraScale Architecture and Product Data Sheet, © Xilinx 2018, Source: https://www.xilinx.com/support/documentation/data_sheets/ds890-ultrascale-overview.pdf
- [YAN17] L. Yang, R. Guo, S. Xie, D. Wang, “A reconfigurable ASIC-like image polyphase interpolation implementation method”. 2017 7th IEEE International Conference on Electronics Information and Emergency Communication (ICEIEC)

- [YIN18] S. Yin, P. Ouyang, S. Tang, F. Tu, X. Li, S. Zheng, T. Lu, J. Gu, L. Liu, and S. Wei, “A High Energy Efficient Reconfigurable Hybrid Neural Network Processor for Deep Learning Applications”, IEEE JOURNAL OF SOLID-STATE CIRCUITS. doi: 10.1109/JSSC.2017.2778281

Open Access Dieses Kapitel wird unter der Creative Commons Namensnennung 4.0 International Lizenz (<http://creativecommons.org/licenses/by/4.0/deed.de>) veröffentlicht, welche die Nutzung, Vervielfältigung, Bearbeitung, Verbreitung und Wiedergabe in jeglichem Medium und Format erlaubt, sofern Sie den/die ursprünglichen Autor(en) und die Quelle ordnungsgemäß nennen, einen Link zur Creative Commons Lizenz beifügen und angeben, ob Änderungen vorgenommen wurden.

Die in diesem Kapitel enthaltenen Bilder und sonstiges Drittmaterial unterliegen ebenfalls der genannten Creative Commons Lizenz, sofern sich aus der Abbildungslegende nichts anderes ergibt. Sofern das betreffende Material nicht unter der genannten Creative Commons Lizenz steht und die betreffende Handlung nicht nach gesetzlichen Vorschriften erlaubt ist, ist für die oben aufgeführten Weiterverwendungen des Materials die Einwilligung des jeweiligen Rechteinhabers einzuholen.