Sustainable Computing: Informatics and Systems 35 (2022) 100742

Contents lists available at ScienceDirect

Sustainable Computing: Informatics and Systems

journal homepage: www.elsevier.com/locate/suscom

EXTREM-EDGE—EXtensions To RISC-V for Energy-efficient ML inference at the EDGE of IoT

Vaibhav Verma [a,*], Tommy Tracy II [b], Mircea R. Stan [a]

[a] *University of Virginia, Department of Electrical and Computer Engineering, Charlottesville, 22903, VA, USA*
[b] *University of Virginia, Department of Computer Science, Charlottesville, 22903, VA, USA*

A R T I C L E I N F O

Keywords:
Artificial intelligence
RISC-V ISA extensions
Machine learning
Hardware/software co-design
AI hardware

A B S T R A C T

Artificial intelligence (AI) and machine learning (ML) have emerged as the fastest growing workloads ranging from applications like object detection, natural language processing and facial recognition to self-driving cars. The proliferation of these compute-intensive workloads resulted in numerous hardware accelerators to fill the gap between the performance and energy-efficiency requirements of AI applications and the capabilities of current architectures like CPU and GPU. In most cases these accelerators are specialized for a particular task, are costly to produce, require special programming tools, and can become obsolete as new ML algorithms are introduced. To solve these problems, we present EXTREM-EDGE, a hardware/software co-design approach to add custom extensions to the open-source RISC-V Instruction Set architecture (ISA) for designing a scalable and flexible ML processor architecture. EXTREM-EDGE augments the RISC-V processor with hardware AI functional units (AFU) along with ISA extensions which directly target these AFUs. EXTREM-EDGE is a system-level solution which is easy to program, enables royalty-free production and provides flexibility for future workloads. It enables the designers to quickly adapt to any hardware or ISA/software changes and allows the design-space exploration of various available hardware, instructions and software options. This enables a processor architecture which addresses the requirements of current AI/ML workloads, gives the flexibility to hot-swap AFUs when better hardware is available and scales with new AI instructions in response to rapidly evolving AI algorithms while providing a streamlined development flow for both hardware and software. EXTREM-EDGE provides 1.75x (MAC) to 17.63x (PIM VMM) performance improvements for a GEMV kernel and 1.41x (MAC) to 4.41x (PIM VMM) reductions in processor clock cycles for ResNet-8 neural network model from MLPerf Tiny benchmark depending upon the size of added accelerators and complexity of added instructions.

1. Introduction

Artificial Intelligence (AI) has been around for decades [1] but the demand for practical machine learning (ML) and AI systems has grown exponentially in the last few years. This deluge in the demand for systems with intelligent capabilities was triggered in part by ML models [2] achieving better than human accuracy ($\approx$ 95%) in the ImageNet Large Scale Visual Recognition Challenge [3], and Deep Neural Networks (DNN) like AlphaGo [4] winning against the human Go champions. On the other hand, availability of high-performance hardware like Graphics Processing Units (GPU) to run huge neural network models further added to the popularity of ML systems. Since AI applications are highly compute and memory intensive, the majority of the hardware designed for and used in early ML implementations was aimed at the data center scale cloud AI applications. Recently, this trend has changed due to the emergence of new computing paradigms such as Artificial Intelligence of Things (AIoT) [5] and Edge Intelligence [6] where edge IoT devices and sensors are continuously being strengthened with AI and ML processing capabilities. The main idea of edge AI is to compute the data as close as possible to data generation sources like IoT sensors to alleviate the "Memory Wall" problem in computer architecture [7,8] by reducing the amount of data that is transferred to the cloud. Edge AI also offers enhanced security and privacy by processing the data in the edge consumer device itself. Hence, enabling edge AI is an important step in realizing the goal of truly smart devices with embedded intelligence.

Before proceeding, we will try to clarify the term edge device. The term "IoT edge" is used to refer to a wide array of devices ranging from tiny microcontroller units (MCU) to self-driving cars. The main focus of this work is on the resource and power-constraint devices and

* Corresponding author.
E-mail addresses: vv8dn@virginia.edu (V. Verma), tjt7a@virginia.edu (T. Tracy II), mircea@virginia.edu (M.R. Stan).

https://doi.org/10.1016/j.suscom.2022.100742
Received 22 June 2021; Received in revised form 3 March 2022; Accepted 16 April 2022
Available online 21 April 2022
2210-5379/© 2022 Elsevier Inc. All rights reserved.

applications at the "extreme" edge of IoT. Such devices are also referred to as TinyML devices in recent literature [9].

Many hardware solutions have been proposed to accelerate the processing of AI/ML workloads such as Convolutional Neural Network (CNN) [10], Recurrent Neural Network (RNN) [11] and Long Short-Term Memory (LSTM) [12]. Domain Specific Architectures (DSA) for neural network acceleration like the Tensor Processing Unit (TPU) [13] have also been proposed in the industry to keep up with the ever rising compute requirements of Deep Neural Networks. But these large accelerators are not an ideal fit for edge devices as most of them utilize large batch sizes (up to 250 in some cases) to achieve high utilization in order to improve the energy-efficiency and performance. But this cannot be the case with edge AI applications mainly due to power and size reasons. Edge devices mostly operate on small batch sizes (≈ 1). For example, in the use case of a smart security camera, the camera operates and needs to run inference on 1 frame at a time. Another major issue in adopting these large accelerators for edge devices is that edge devices are complete systems and need to support both AI and non-AI tasks. But there is no streamlined development flow to integrate these accelerators in edge processors to enable support for both AI and non-AI tasks on the same system. Finally, these large AI accelerators are costly to develop and they are custom designed for a few specific AI workloads. But AI and ML algorithms are evolving at a faster pace than AI hardware development. As a result, many AI accelerators can become obsolete as new algorithms emerge in literature. Thus there is a need for a new AI hardware solution for edge devices that is inexpensive, flexible towards evolving workloads and that can handle small batch size use cases efficiently. There is also a need for a standardized development methodology to integrate AI accelerators into existing and emerging edge processors to support both AI and non-AI tasks on smart edge devices.

Open-source ecosystems like the RISC-V ISA [14] provide an excellent opportunity to solve many of the problems listed above. RISC-V enables cheaper edge devices by removing the license royalty costs associated with popular edge processor architectures like Arm ISA. RISC-V also leverages open-source efforts to continuously improve both the software and the hardware as new algorithms and workloads evolve. RISC-V even supports custom extensions to develop new ISA instructions to support emerging algorithms [15] or enable faster processing of existing algorithms. RISC-V ISA already supports a rich set of standard extensions like the Vector "V" extension, DSP and packed SIMD "P" extension and the bit manipulation "B" extension. But developing these extensions in separate silos from the hardware development leads to the problem of hardware playing catch-up with the ISA and the software. Hence, a hardware/software co-design methodology is required to unleash the full potential of open-source RISC-V ISA extensions.

Here we present EXTREM-EDGE, a hardware/software co-design methodology for designing energy-efficient processors for edge AI. EXTREM-EDGE extends the RISC-V ISA with custom AI instructions and the RISC-V processor microarchitecture with custom AI accelerators as tightly integrated hardware functional units. ISA extensions directly map to the functionality of added accelerators and hence allow a standardized development flow to integrate these accelerators into the system design and the software stack. Details about AI accelerator functional units are provided in Section 2. Section 3 explains the AI specific extensions to the RISC-V ISA. The rest of the paper describes the EXTREM-EDGE hardware/software co-design methodology in Section 4 while Section 6 provides specific case study examples of ISA extensions with EXTREM-EDGE. The evaluation methodology has been thoroughly explained in Section 5 and Section 7 discusses the results of evaluation. Section 8 discusses the related research while Section 9 provides concluding remarks for this article.

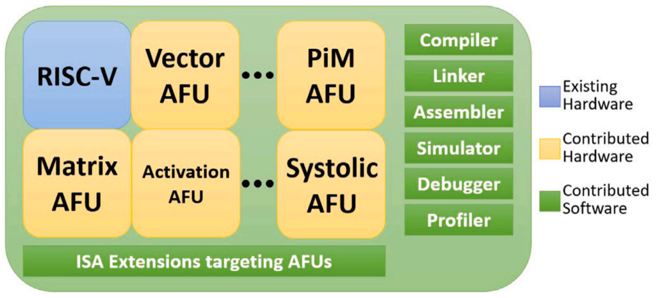

Fig. 1. Top-level overview of EXTREM-EDGE processor design methodology.

2. AFU - AI accelerator as a Functional Unit

EXTREM-EDGE follows a system-level design approach where emphasis is placed on modular design of both hardware accelerator functional units and associated ISA extensions. One of the key ideas in the EXTREM-EDGE design methodology is that custom AI accelerators are incorporated as tightly integrated AI Functional Units (AFU) in the RISC-V processor pipeline.

AFUs are essentially hardware accelerators that are appropriately scaled to be a part of the RISC-V processor pipeline. AFUs can either be single-cycle or multi-cycle execution units which are part of the extended RISC-V execution stages. Individual AFUs accelerate one specific part of a neural network and as a whole they can be used to accelerate the whole network which is part of the AI workload. Along with accelerating the AI section of the workload, the standard RISC-V processor pipeline is used to execute pre/post-processing and control steps required as well as standard workloads on the same processor. Fig. 1 provides a top-level overview of the proposed processor design methodology. As shown in Fig. 1, AFUs can range from matrix and vector processing units to hardware accelerators for activation functions to more specialized processing-in-memory accelerators or systolic arrays. The interface of each AFU is designed to enable the flow of inputs from and sending the outputs back to the processor pipeline stages. For example, in a 5-stage RISC-V processor implementation, the MAC AFU receives input operands in the execute stage from the decode stage while the output of MAC AFU is written back to the processor general-purpose register file (GPR) in the writeback stage. Hence, the I/O interface of the MAC AFU is similar to the ALU I/O interface utilized in register transfers and does not involve any system bus communication outside the processor. Similarly, the interface of the processing-in-memory (PIM) AFU is similar to the register file interface of the processor. The PIM AFU receives the input operand in execute stage from the decode stage, returns output in memory stage and the output is written back to the processor GPR in the writeback stage. This tight integration of AFUs enables EXTREM-EDGE to execute complete applications and not just the AI/ML kernels on a single processor. The finer grained offloading of AI tasks is a result of extending the RISC-V ISA with special instructions such as type conversion, vector–matrix multiplication (VMM), vector–vector multiplication (VVM) and general matrix multiply (GEMM). The new ISA instructions are an AI extension to the base RISC-V ISA and are explained in greater detail in Section 3.

The AFUs are close in spirit to the Floating-Point Units (FPU) and Vector Processing Units (VPU) found in modern processors. Historically, FPUs also started as separate hardware but eventually became a part of modern CPU ISAs as floating-point operations became popular in mainstream workloads. VPUs followed a similar path in the RISC-V ecosystem where vector accelerators started as separate coprocessors [16] but are now becoming part of the RISC-V ISA via the "V" ISA extension. Similarly, as AI/ML workloads become more prevalent in different application domains, we believe that specialized AI instructions will also highly benefit from becoming part of the CPU ISA. This is

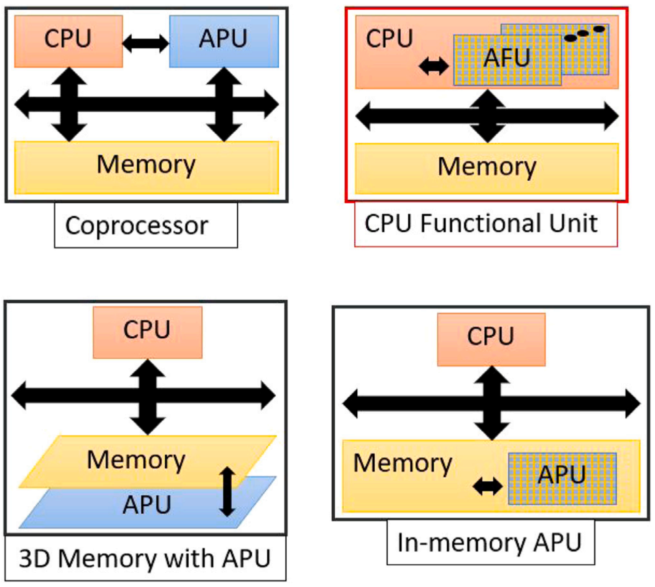

Fig. 2. AI Functional Units (AFU) are tightly integrated in the processor pipeline unlike prior AI Processing Units (APU) which communicate with the processor over an external bus.
Source: Adapted from [18].

the main motivation for this work. We also realize that as we adapt the RISC-V hardware and ISA for AI applications, it is of utmost importance to develop the software infrastructure to efficiently target this new EXTREM-EDGE processor architecture. These hardware and software co-design efforts for EXTREM-EDGE are detailed in Section 4. One key differentiator in EXTREM-EDGE development is the integration of AFUs inside the processor pipeline as opposed to accelerators being used as coprocessors over a shared bus. As shown in Fig. 2 top left, traditional AI Processing Units (APU) such as [10,13] are introduced as standalone accelerators which share resources with CPU and communicate over a bus. With the advancements in 3D integration technology and the introduction of 3D DRAM memories such as HBM [17], APUs integrated in base logic die of 3D stacked memories were proposed. These 3D-stacked APUs shown in Fig. 2 bottom left aim to accelerate AI/ML workloads close to the memory but still communicate with the CPU over a shared bus. Such coarse-grained offloading of AI tasks is beneficial only in the data-center AI processing where large batch sizes are used to carve out huge chunks of AI workloads. Recently, many APUs with in-memory processing of AI tasks have been introduced [18]. These in-memory AI Processing Units utilize analog characteristics of memories to perform bulk multiply-accumulate operations inside the memory. These in-memory accelerators as shown in Fig. 2 bottom right carve out some portion of memory hierarchy as dedicated accelerator, communicate with CPU over a bus and aim to accelerate complete neural networks inside the memory. But most in-memory accelerators are not able to accelerate all the layers of the neural networks, e.g. activation and pooling layers, and as a result suffer from data transfer costs to process these layers in the base CPU. Our design philosophy is shown in Fig. 2 top right where accelerators are integrated inside the CPU as smaller functional units rather than big processing units as before.

The motivation behind these smaller functional units is to balance the latency of computations with the processor pipeline delays to enable finer-grained offloading of AI specific tasks to these AFUs while processing the workloads with a mix of AI and non-AI instructions as is usually the use case for Artificial Intelligence of Things (AIoT) edge devices. This will help to accelerate AI instructions described in Section 3 seamlessly on the same processor which runs non-AI part of the applications. This helps us to reduce the cost and complexity of AI hardware at the edge of IoT since EXTREM-EDGE is a complete system-level solution rather than an accelerator for AI tasks which depends on a control processor.

The modular design approach of EXTREM-EDGE provides the flexibility to choose and add specific AFUs to accelerate new layers of different neural networks as AI algorithms keep on evolving. It also empowers to replace current AFUs with better ones as new accelerator designs become available, with the minimal effort of adding an I/O interface wrapper around the desired accelerator. Hence, EXTREM-EDGE allows multiple different customized edge AI processors to be defined from a base processor architecture description based on which AFUs are chosen to be a part of the processor pipeline. Further details about how this is enabled during the design process are provided in Section 4. There are several design considerations when AFU hardware is built to be included in the RISC-V processor pipeline. First, there is the question of how big the AFU should be. For example, in case of a Processing-in-Memory (PIM) AFU, we must balance the latency and frequency of PIM operations with the latency for weight update since bigger AFU requires less frequent weight updates but takes longer for each update cycle. Additionally, the maturity and endurance of underlying memory technology also plays a role in finalizing the size of the AFU. Once we have defined the size of a particular AFU, we also need to decide how many of these can we fit in the given area budget. EXTREM-EDGE enables researchers to find answers to all these questions as part of the design-space exploration of hardware AFUs and the impact of each hardware design on the end IoT edge AI application. These AFUs are exposed to the software using the custom ISA extensions described in Section 3.

3. AI instructions as ISA extensions

On the software side, EXTREM-EDGE makes two important contributions – AI ISA extensions to RISC-V at the architecture level and generation of complete Software Development Kit (SDK) for the designed edge processor at the system level with programming support for AI domain-specific languages and frameworks. Along with designing AFU hardware, it is equally important to expose these new functional units in the processor via ISA extensions. One of the ideas of EXTREM-EDGE is to identify and formalize new instructions that will benefit acceleration of AI tasks and add the same to RISC-V ISA. All new instructions under the EXTREM-EDGE design are added to CUSTOM-2 slot of RISC-V ISA opcode space. Hence, compatibility with base RISC-V ISA is always maintained during EXTREM-EDGE design.

These AI ISA extensions have been grouped into 2 separate categories – “Top-down” extensions which are designed based on existing AI algorithms and “Bottom-up” extensions which are designed based on the needs of hardware AFUs. These different versions are explained in greater detail next.

3.1. Top-down ISA extensions:
The top-down ISA extensions consist of the intuitive new ISA instructions which directly accelerate certain portions of AI workloads such as multiply-accumulate (MAC) instruction, vector–vector multiply (VVM) instruction, vector–matrix multiply (VMM) instruction and generalized matrix multiply (GEMM) instruction. These instructions correspond to the most frequently occurring operations in the AI workload profiling. As explained in Section 4, hardware for top-down instructions is automatically generated from the behavioral description of these instructions in the processor model specification. Top-down ISA extensions are inspired form the AI algorithms accelerated by them.

3.1. Bottom-up ISA extensions:
The bottom-up ISA extensions consist of the non-intuitive new ISA instructions which are utilized to directly support integration of various AFUs inside the processor pipeline. These instructions

either perform the complex operation supported by corresponding AFUs e.g. in-memory vector–matrix multiplication in the case of PIM AFU or these instructions perform the auxiliary operations required by the AFUs e.g. pre/post processing of neural network weights/data, loading the result of in-memory VMM from specialized AFU to RISC-V register and storing the weights inside the PIM AFU in desired format to enable PIM operations. Bottom-up ISA extensions might seem complex for RISC ISA or might seem not so useful for AI operations but these extensions serve a very critical role in enabling EXTREM-EDGE designs to process AI and non-AI workloads on a single processor. Bottom-up extensions are inspired by the custom hardware AFU accelerators and they are used by the compiler to direct specific portions of the workload to specific AFUs in a specific format.

These ISA extensions directly target and support the integration of AFUs described in Section 2 and are developed in sync with the hardware. Making these new instructions part of the processor ISA helps with fine-grained offloading of AI tasks for appropriate AFUs since they are executed in the same processor pipeline. Just like a normal add instruction is executed in the processor ALU, a vector add is executed in vector AFU while a matrix operation is executed in matrix AFU. Many ISA extensions are part of the EXTREM-EDGE design including memory operations (load/store vectors and matrices), arithmetic vector–matrix and matrix–matrix operations, non-linear activation operations, fused instructions and operations on non-standard data types like int4 and int8. We expect new ISA instructions will keep adding to EXTREM-EDGE AI extensions as we profile more neural network architectures and as AI algorithms keep on evolving. The modular design methodology as illustrated in Section 4 allows the flexibility to add new instructions and optimize the compiler to use the same.

Along with defining new ISA extensions, this work also builds the complete software infrastructure required to target the newly designed edge AI processor. Another major objective of EXTREM-EDGE methodology is to enable end-users to be able to program their AI applications in Python using programming framework of choice like TensorFlow [19], PyTorch [20] etc. in addition to C/C++ programming support. Majority of the AI software designers prefer Domain Specific Languages (DSL) like TensorFlow, PyTorch, Keras and MxNet etc. to code AI algorithms. Hence it becomes very important to support these advanced languages in the design methodology to enable faster adoption of proposed edge AI processor. Section 4 provides details of how this is achieved using the proposed design methodology.

4. Hardware/Software Co-design

EXTREM-EDGE is a standardized hardware/software co-design methodology for the design of edge AI processors. As explained in Sections 2 and 3, EXTREM-EDGE augments the RISC-V processor with custom hardware functional units and ISA extensions to accelerate the energy-efficient processing of AI/ML workloads at the edge of IoT. The hardware and software enhancements are co-developed in sync. EXTREM-EDGE software development framework is broken down into two different phases as shown in Fig. 3. At the frontend of design, TVM [21] optimizing deep learning compiler is used to take input neural network definitions in a variety of different domain specific languages (DSL) like TensorFlow, PyTorch, CoreML, MXNet, DarkNet, Keras etc. TVM allows to convert input neural networks from different DSLs down to C code. In addition, TVM also supports generating LLVM IR, assembly code and binary executable for different hardware backends. EXTREM-EDGE utilizes this aspect of TVM to allow users to write their AI application in DSL of their choice. Then TVM is utilized to convert the input neural network definition to C code.

At this point, Synopsys ASIP Designer [22] comes into picture as the backend of the design methodology. ASIP Designer takes C

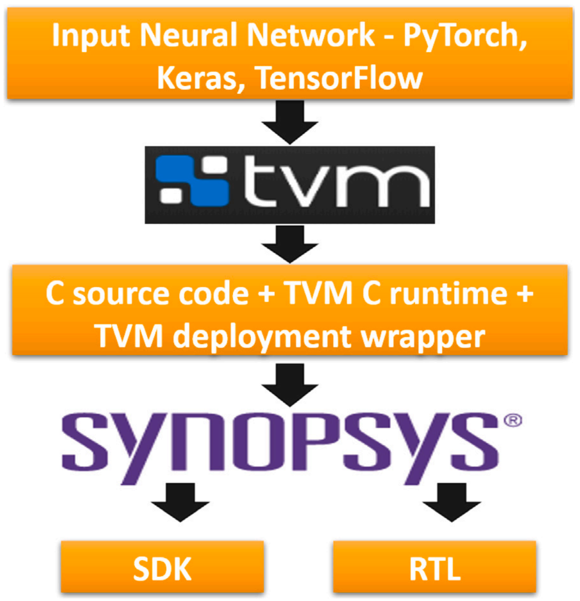
Input Neural Network - PyTorch, Keras, TensorFlow
tvm
C source code + TVM C runtime + TVM deployment wrapper
synopsys
SDK
RTL

Fig. 3. EXTREM-EDGE design methodology uses TVM deep learning compiler as the frontend to map domain specific languages to C and Synopsys ASIP Designer as the backend to generate complete SDK and synthesizable RTL.

code as input and uses a retargettable compiler to compile it for the newly designed EXTREM-EDGE processor. Along with the retargettable compiler, it also generates a complete SDK including assembler, linker, debugger, instruction set simulator and profiler. All these tools are utilized in the complete development of EXTREM-EDGE including profiling for AI applications to look for bottlenecks. Along with the automatic SDK generation, ASIP Designer also helps to generate a synthesizable RTL for the newly designed processor.

Fig. 4 (adapted from [22]) gives a detailed overview of the backend design process. The backend (shown in green color in Fig. 4) of the EXTREM-EDGE design begins with a top-level description of the processor model in a tool specific processor design language called nML [23]. nML is used to describe both the processor microarchitecture and ISA. Once this processor description is complete, it serves as the golden description for the generation of both the SDK and the synthesizable RTL. Since both software and hardware are generated from a single processor description, they are always in sync. nML processor description is used to quickly communicate hardware changes to software and vice-versa. This allows for a very flexible design strategy where users can easily remove or add different hardware modules to the processor design along with respective ISA instructions. By varying the hardware choices and ISA instructions, EXTREM-EDGE users can explore a wide design-space to efficiently target their AI applications to edge processors. Rapid hardware prototyping also allows design decisions to be based on actual hardware implementations. Software stack generation allows extremely short design turn-around times between adding an instruction or adding an AFU to generating actual performance and power/energy metrics before proceeding to next iteration. Users also enjoy the flexibility of choosing from a variety of Domain Specific Languages to adapt AI algorithms to the designed processors. This enables an efficient design process for edge AI processors.

Referring to Fig. 4, starting from a base processor description like RV64IMC in nML, we can profile different AI applications, described in various DSLs and translated to C, to look for bottlenecks in code execution. The profiler generates detailed reports at both function and instruction level. This information is used to identify the bottlenecks and further add new instructions and/or AFUs in nML processor description to accelerate the bottleneck portion of the code. With this new processor description, SDK and RTL are automatically generated and we

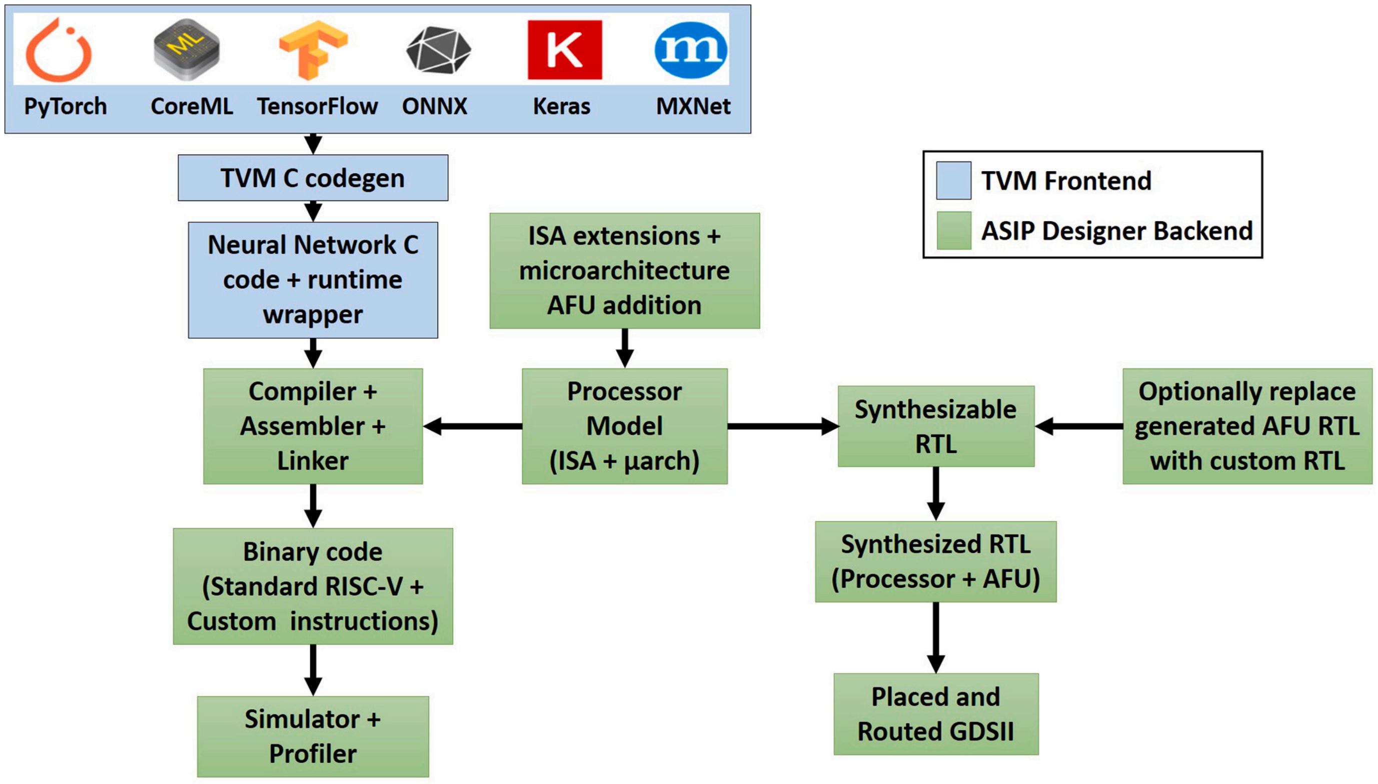
PyTorch
CoreML
TensorFlow
ONNX
Keras
MXNet
TVM C codegen
Neural Network C code + runtime wrapper
ISA extensions + microarchitecture AFU addition
TVM Frontend
ASIP Designer Backend
Compiler + Assembler + Linker
Processor Model (ISA + μarch)
Synthesizable RTL
Optionally replace generated AFU RTL with custom RTL
Binary code (Standard RISC-V + Custom instructions)
Synthesized RTL (Processor + AFU)
Simulator + Profiler
Placed and Routed GDSII

Fig. 4. Detailed overview of the EXTREM-EDGE backend design methodology supported by Synopsys EDA tools [22] to co-design hardware and software together. Adapted from [22].

can run multiple iterations till we have optimized the applications of interest. Then we can synthesize the RTL to check for area, performance and power numbers of the designed processor. If we do not meet any of the design targets, we can again go back and make changes to the nML processor description. We can also run RTL simulations using test bench generated from the C application code to verify functionality in RTL. This allows us to quickly iterate through different hardware and software design steps and at each step emits complete software and hardware infrastructure which are always in sync.

5. Evaluation methodology

The evaluation of the proposed hardware/software co-design methodology is done using two separate case study experiments as explained in Section 6. Both the experiments follow same evaluation methodology as described below.

5.1. Benchmark

The ResNet-8 and MobileNetV1 neural network models from the MLPerf Tiny benchmark [24] trained on CIFAR-10 image database and Visual wake words dataset [25] respectively are used as the evaluation AI workloads representing image classification and visual wake word tasks on edge devices. Both ResNet-8 and MobileNet neural networks were quantized to int8 datatype to accurately represent edge AI workload. General matrix (8x8, 16x16) – vector (1x8, 1x16) multiply (GEMV) kernels are also used to test the performance improvements of the proposed scheme.

5.2. Baseline

A 5-stage in-order implementation of the 64-bit base RISC-V ISA in RV64IMC configuration is chosen as the baseline for all evaluation experiments. The baseline ISA supports standard RISC-V integer (I), multiplication (M) and compressed instructions (C) extensions. We have chosen RISC-V as the baseline because it is an open-source processor ISA which is increasingly becoming popular for IoT edge systems.

5.3. Compilation

As described in Section 4, two-step compilation strategy is used in all the experiments inline with EXTREM-EDGE system design flow. A TensorFlow Lite for Microcontrollers implementation of the benchmark networks is compiled using TVM deep-learning compiler to C code in the first step. In the second step, this C code along with TVM C runtime is compiled using C compiler generated by Synopsys ASIP Designer for base and modified RISC-V processors to produce the final binary executable of the workloads.

5.4. Simulation framework

All the simulations are performed using a cycle-accurate system simulator generated by Synopsys ASIP Designer for base and modified RISC-V processors based on the processor ISA and microarchitecture specifications as explained in Section 4. All simulations represent cycle accurate performance results for implemented processor hardware without any approximations.

6. Experiments

We performed two different case study experiments based on two types of ISA extensions explained in Section 3. The purpose of these case studies is to illustrate the advantages of using EXTREM-EDGE methodology to add custom extensions of varying complexity to RISC-V ISA for edge AI applications.

6.1. MAC instruction ISA extension

The first case study focuses on adding a simple top-down instruction to the base RISC-V ISA. The extension instruction chosen for this case study is multiply-accumulate (MAC) instruction. MAC is the most common operation in all AI/ML workloads which multiplies two operands and accumulates the result with the third operand. A MAC instruction was added to the base RISC-V ISA in the 3 register operand format. Hardware MAC RTL was auto-generated from the behavioral description of the MAC instruction using the co-design methodology described in Section 4. C compiler was given hints to convert all MAC opportunities in the benchmark neural network to MAC ISA instruction instead of two separate addition and multiplication instructions. After successful addition of scalar MAC instruction, 4-lane vector packed SIMD instruction was also implemented in the simulator as an emulation of the scalar MAC instruction. Results from these experiments are reported in Section 7.

6.2. In-memory VMM instruction ISA extension

The second case study focuses on extending the RISC-V ISA with a more complex bottom-up ISA extension set with 3 different instructions — vector–matrix multiply (VMM) instruction, instruction to store weights in VMM AFU memory from RISC-V registers and instruction to load output of VMM from VMM AFU memory to RISC-V register. VMM instruction is added in 2 register RISC-V operand format while VMM store/load instructions are added in RISC-V load/store instruction format. For a given number N, a 1xNxN vector–matrix multiplication represents the multiplication of 1xN vector of input data with NxN matrix of weights to produce 1xN vector of output activations. A specialized hardware AI functional unit (AFU) with in-memory VMM functionality was modeled and integrated in the RISC-V processor pipeline. VMM AFU operates as a regular memory for load/store operations while it operates as an in-memory VMM AFU for vector–matrix multiply operation reading the input operand in the decode processor pipeline stage and returning the output in memory stage of the 5-stage RISC-V processor pipeline. Special C compiler intrinsics were exposed to TVM to generate compatible C source code for benchmark neural network model. These compiler intrinsics help to compile vector–matrix multiply portion of the source code to VMM assembly instruction. Results from the VMM addition case study have been reported in Section 7.

7. Results

The results from the case study experiments are reported here. The results have been grouped by different instruction extensions which were added to the base RISC-V ISA.

7.1. Performance with scalar MAC instruction

The scalar MAC instruction was added in both 32-bit and 64-bit variants of the RISC-V ISA. As shown in Fig. 5, 32-bit RISC-V reported a performance improvement of 12% while 64-bit RISC-V reported an improvement of 17% measured in reduction of simulation processor cycles to process the ResNet-8 benchmark neural network. Although this is not a substantial improvement but it shows the potential of custom ISA extensions where 1 single instruction addition leads to an overall performance improvement of up to 17% for a complete neural network model with negligible area penalty. As shown in Fig. 7 scalar MAC instruction provides a 1.75x performance improvement for a GEMV (1x8 vector * 8x8 matrix) kernel. GEMV kernels are a central compute block in all major artificial neural network implementations.

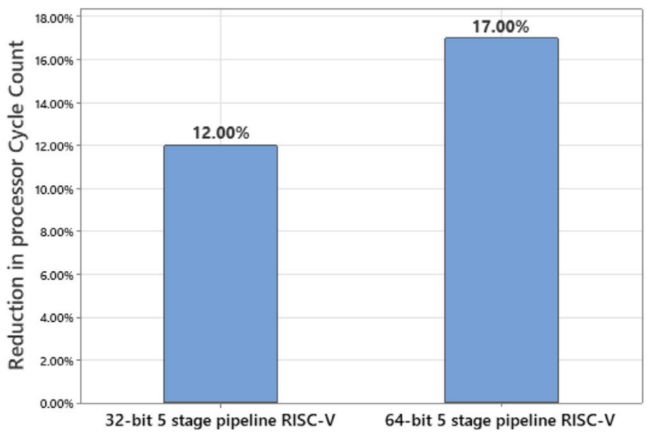

Fig. 5. Cycle count performance improvement with Scalar MAC instruction.

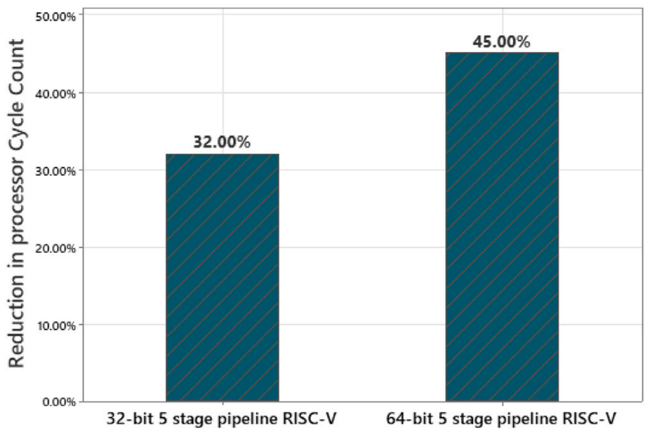

Fig. 6. Cycle count performance improvement with SIMD MAC instruction.

7.2. Performance with SIMD MAC instruction

Single Instruction Multiple Data (SIMD) is an architectural technique to exploit data level parallelism in workloads. Since neural networks exhibit plenty of data level parallelism, a packed SIMD implementation of vectorized MAC instruction was emulated using EXTREM-EDGE with 4 parallel vector lanes. As shown in Fig. 6, this experiment resulted in overall performance improvement of 32% in 32-bit RISC-V and 45% in 64-bit RISC-V processor implementations. The results from SIMD MAC experiment clearly show the advantages provided by EXTREM-EDGE methodology which was utilized to explore instruction design-space by easily and quickly emulating SIMD MAC instruction using an implemented scalar MAC instruction. EXTREM-EDGE users can take advantages of such ISA emulation techniques to explore complex vector ISA instruction without the need to actually implement the bulky vector “V” RISC-V ISA extension.

7.3. Performance with in-memory VMM instruction

The addition of in-memory VMM instruction was evaluated in two steps. First, we ran evaluations with a small GEMV kernel as the workload. The results are shown in Fig. 7. VMM instructions (including the overhead of VMM load/store instructions) resulted in a speed improvement of 10.1x for an 8x8 matrix multiplication with 1x8 vector and 17.63x speed-up for 16x16 matrix multiplication with 1x16 vector compared to performing same vector–matrix multiplications using baseline 64-bit RISC-V ISA.

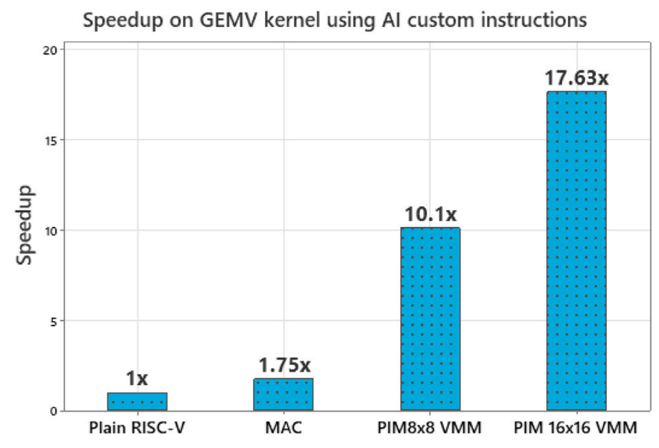

Fig. 7. Speed-up for general vector–matrix multiply (GEMV) kernel with MAC, 8x8 and 16x16 In-memory VMM instructions.

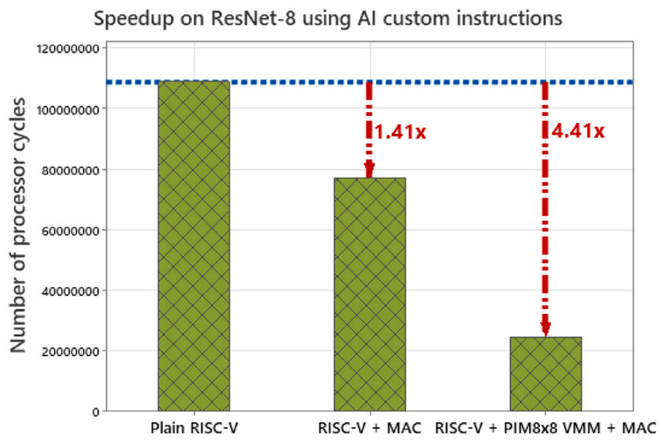

Fig. 8. Comparison of performance speed-up for ResNet-8 model as a result of MAC instruction and in-memory VMM instruction compared to baseline RISC-V processor.

17x improvement on a small VMM kernel does not necessarily translate to a similar improvement in end-to-end AI applications. So, we conducted further experiments to evaluate the effect of in-memory VMM instruction on ResNet-8 benchmark neural network model. In-memory VMM instructions (including the overhead of VMM memory load/store instructions) resulted in an overall speed-up of 4.41x for the ResNet-8 benchmark neural network model as shown in Fig. 8.

In order to further investigate the impact of in-memory VMM instruction, we evaluated the performance on MobileNet V1 benchmark neural network model from MLPerf Tiny benchmark [24]. MobileNet model consists of depthwise separable convolution layers [26] which are different from standard 2D convolution layers in the ResNet-8 model. As show in Fig. 9, EXTREM-EDGE resulted in performance improvement of 1.35x with MAC instruction and 2.15x with in-memory VMM instructions addition to the RISC-V ISA for processing complete MobileNet neural network model.

These results clearly denote the importance of custom AI instruction extensions to effectively accelerate AI workloads on IoT edge devices. EXTREM-EDGE provides a standardized methodology to co-design the hardware and software for these custom AI instructions.

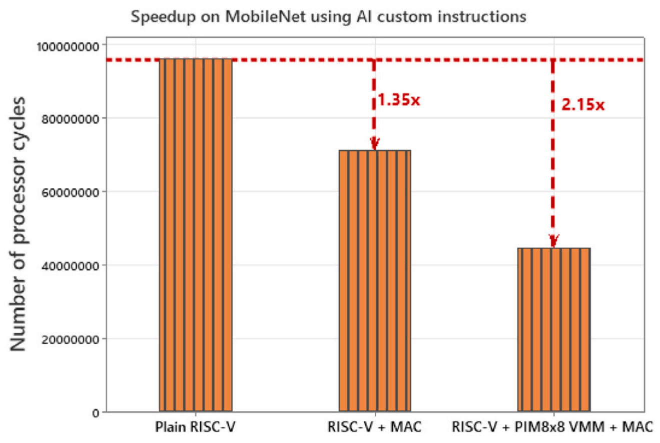

Fig. 9. Comparison of performance speed-up for MobileNet model as a result of MAC instruction and in-memory VMM instruction compared to baseline RISC-V processor.

8. Discussion and related work

Current state-of-the-art hardware solutions for accelerating AI applications consist of many different options. Initially, researchers focused on CPUs as the hardware of choice but quickly realized that CPUs had a hard time keeping up with highly compute-intensive deep learning algorithms. Still some solutions like SLIDE [27], XNN [28] and Neural-Magic [29] are utilizing CPU-based AI acceleration. But these solutions are specialized for special layers like fully-connected or sparse layers, rely on specific hardware (Intel multi-core CPUs) for acceleration or cater to very niche application areas like connectomics. Additionally, these solutions are not the easiest to program as some of these have been written in C++ and with inline assembly rather than popular ML frameworks like TensorFlow or PyTorch. On top of that, all of these solutions require aggressively multi-core CPU which is usually not available in IoT edge devices. Hence, EXTREM-EDGE provides a viable direction for designing edge AI processors with the right balance of performance, cost, ease of programming and complexity.

EXTREM-EDGE embodies the philosophy of custom ISA extensions and tightly integrated hardware functional unit design along with supporting software development infrastructure to accelerate AI applications at the edge in an energy-efficient manner. Although in this article EXTREM-EDGE utilizes TVM and Synopsys ASIP Designer as the frontend and backend compilers of choice but EXTREM-EDGE is not limited to these choices. Other open source deep-learning compilers can be used in place of TVM but TVM offers a great breadth which covers most of the popular machine learning frameworks like TensorFlow and PyTorch. Other deep-learning compiler frameworks lack the open-source community development momentum which TVM enjoys. Open-source contributors keep improving TVM stack to support additional AI algorithms, operators, models, frameworks and backends. That is the reason TVM was chosen as the frontend compiler for EXTREM-EDGE.

Similarly, other backend compilers and processor design tools can also be used to replace Synopsys ASIP Designer as the backend software stack. There are many other industrial grade options which are viable candidates for backend compiler, simulator and hardware development e.g. SiFive Freedom Studio, Codasip Studio and Cadence Tensilica Xtensa Xplorer. But Synopsys ASIP Designer provides a more intuitive development environment and has been proven through many industry and academic processor implementations.

Users can even completely bypass proprietary backend compiler and directly add new RISC-V instructions in RISC-V GCC compiler and assembler along with RISC-V Spike simulator. Recently LLVM compiler stack has also released support for RISC-V ISA and thus users can even directly add new instructions to the LLVM compiler stack. But adding new instructions to all these compilers and simulators is a very cumbersome process and requires deep software design expertise. Additionally, there is no defined standard flow to do these custom instruction additions. EXTREM-EDGE solves these problems by providing a standardized, modular and agile design flow for custom instruction extensions to the RISC-V processor ISA.

9. Conclusion

In this paper, we presented EXTREM-EDGE hardware/software co-design methodology for adding custom instruction extensions to the RISC-V ISA along with custom hardware accelerators for high-performance and low-power hardware solution for AI applications at the edge of IoT. EXTREM-EDGE adopts a tight integration philosophy for addition of AI functional units (AFU) in the processor pipeline which allows both AI and non-AI tasks to be performed on the same processor. EXTREM-EDGE designs have shown up to 45% performance improvement with single MAC instruction addition to RISC-V ISA and upto 4.41x speed-up using a set of 3 (vmm.ld, vmm.sd and vmm) custom instructions on ResNet-8 neural network model from MLPerf Tiny benchmark.

CRediT authorship contribution statement

Vaibhav Verma: Conceptualization, Development, Experiments, Writing – original draft. **Tommy Tracy II:** Experiments, Manuscript preparation. **Mircea R. Stan:** Conceptualization, Funding acquisition, Writing – original draft.

Declaration of competing interest

No author associated with this paper has disclosed any potential or pertinent conflicts which may be perceived to have impending conflict with this work. For full disclosure statements refer to https://doi.org/10.1016/j.suscom.2022.100742.

Acknowledgment

This work was funded by SRC, USA under GRC AIHW task 2945.001.

References

[1] A.M. Turing, I. Computing machinery and intelligence, Mind LIX (236) (1950) 433–460.
[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, Deep residual learning for image recognition, in: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770–778.
[3] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, Li Fei-Fei, Imagenet large scale visual recognition challenge, Int. J. Comput. Vis. (IJCV) 115 (3) (2015) 211–252.
[4] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, Demis Hassabis, Mastering the game of go with deep neural networks and tree search, Nature 529 (7587) (2016) 484–489.
[5] J. Zhang, D. Tao, Empowering things with intelligence: A survey of the progress, challenges, and opportunities in artificial intelligence of things, IEEE Internet Things J. (2020) 1.
[6] S. Deng, H. Zhao, W. Fang, J. Yin, S. Dustdar, A.Y. Zomaya, Edge intelligence: The confluence of edge computing and artificial intelligence, IEEE Internet Things J. 7 (8) (2020) 7457–7469.
[7] Steven A. Moyer, Performance of the IPSC/860 Node Architecture, Citeseer, 1991.
[8] Wm. A. Wulf, Sally A. McKee, Hitting the memory wall: Implications of the obvious, SIGARCH Comput. Archit. News 23 (1) (1995) 20–24.
[9] Pete Warden, Daniel Situnayake, Tinyml: Machine Learning with Tensorflow Lite on Arduino and Ultra-Low-Power Microcontrollers, O'Reilly Media, Inc., 2019.
[10] Yu-Hsin Chen, Tushar Krishna, Joel Emer, Vivienne Sze, Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks, in: IEEE International Solid-State Circuits Conference, ISSCC 2016, Digest of Technical Papers, 2016, pp. 262–263.
[11] J. Lee, D. Shin, H. Yoo, A 21mw low-power recurrent neural network accelerator with quantization tables for embedded deep learning applications, in: 2017 IEEE Asian Solid-State Circuits Conference (a-SSCC), 2017, pp. 237–240.
[12] D. Kadetotad, S. Yin, V. Berisha, C. Chakrabarti, J. Seo, An 8.93 TOPS/W LSTM recurrent neural network accelerator featuring hierarchical coarse-grain sparsity for on-device speech recognition, IEEE J. Solid-State Circuits 55 (7) (2020) 1877–1887.
[13] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, Doe Hyun Yoon, In-datacenter performance analysis of a tensor processing unit, in: Proceedings of the 44th Annual International Symposium on Computer Architecture, in: ISCA '17, Association for Computing Machinery, New York, NY, USA, 2017, pp. 1–12.
[14] Andrew Waterman, Yunsup Lee, David A. Patterson, Krste Asanović, The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Version 2.0, Technical Report UCB/EECS-2014-54, EECS Department, University of California, Berkeley, 2014.
[15] Gianna Paulin, Renzo Andri, Francesco Conti, Luca Benini, RNN-based radio resource management on multicore RISC-V accelerator architectures, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 29 (9) (2021) 1624–1637.
[16] Y. Lee, A. Waterman, R. Avizienis, H. Cook, C. Sun, V. Stojanović, K. Asanović, A 45nm 1.3GHz 16.7 double-precision GFLOPS/W RISC-V processor with vector accelerators, in: ESSCIRC 2014 - 40th European Solid State Circuits Conference (ESSCIRC), 2014, pp. 199–202.
[17] Hongshin Jun, Jinhee Cho, Kangseol Lee, Ho-Young Son, Kwiwook Kim, Hanho Jin, Keith Kim, HBM (high bandwidth memory) DRAM technology and architecture, in: 2017 IEEE International Memory Workshop (IMW), 2017, pp. 1–4.
[18] Ping Chi, Shuangchen Li, Cong Xu, Tao Zhang, Jishen Zhao, Yongpan Liu, Yu Wang, Yuan Xie, PRIME: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory, in: 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA), 2016, pp. 27–39.
[19] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, Xiaoqiang Zheng, Tensorflow: A system for large-scale machine learning, in: 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), 2016, pp. 265–283.
[20] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, Soumith Chintala, Pytorch: An imperative style, high-performance deep learning library, 2019.
[21] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, Arvind Krishnamurthy, TVM: An automated end-to-end optimizing compiler for deep learning, in: 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18), USENIX Association, Carlsbad, CA, 2018, pp. 578–594.
[22] https://www.synopsys.com/dw/ipdir.php?ds=asip-designer.
[23] A. Fauth, J. Van Praet, M. Freericks, Describing instruction set processors using nML, in: Proceedings the European Design and Test Conference. ED TC 1995, 1995, pp. 503–507.

[24] Colby R. Banbury, Vijay Janapa Reddi, Max Lam, William Fu, Amin Fazel, Jeremy Holleman, Xinyuan Huang, Robert Hurtado, David Kanter, Anton Lokhmotov, David Patterson, Danilo Pau, Jae sun Seo, Jeff Sieracki, Urmish Thakker, Marian Verhelst, Poonam Yadav, Benchmarking tinyml systems: Challenges and direction, 2021.

[25] Aakanksha Chowdhery, Pete Warden, Jonathon Shlens, Andrew Howard, Rocky Rhodes, Visual wake words dataset, 2019.

[26] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam, Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017.

[27] Beidi Chen, Tharun Medini, James Farwell, Sameh Gobriel, Charlie Tai, Anshumali Shrivastava, SLIDE : In defense of smart algorithms over hardware acceleration for large-scale deep learning systems, 2019.

[28] Alexander Matveev, Yaron Meirovitch, Hayk Saribekyan, Wiktor Jakubiuk, Tim Kaler, Gergely Odor, David Budden, Aleksandar Zlateski, Nir Shavit, A multicore path to connectomics-on-demand, in: Proceedings of the 22nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, in: PPoPP '17, Association for Computing Machinery, New York, NY, USA, 2017, pp. 267–281.

[29] Mark Kurtz, Justin Kopinsky, Rati Gelashvili, Alexander Matveev, John Carr, Michael Goin, William Leiserson, Sage Moore, Nir Shavit, Dan Alistarh, Inducing and exploiting activation sparsity for fast inference on deep neural networks, in: Hal Daumé III, Aarti Singh (Eds.), Proceedings of the 37th International Conference on Machine Learning, in: Proceedings of Machine Learning Research, vol. 119, PMLR, 2020, pp. 5533–5543.