2021 IEEE 32nd International Conference on Application-specific Systems, Architectures and Processors (ASAP) | 978-1-6654-2701-2/21/$31.00 ©2021 IEEE | DOI: 10.1109/ASAP52443.2021.00046

FlexACC: A Programmable Accelerator with Application-Specific ISA for Flexible Deep Neural Network Inference

En-Yu Yang, Tianyu Jia, David Brooks, Gu-Yeon Wei
Harvard University
enyu_yang@g.harvard.edu

***Abstract*—Deep neural networks (DNN) have become ubiquitous and dominant in various application domains due to its state-of-the-art learning capabilities. To run compute and memory intensive DNN models, designing specialized hardware accelerators becomes the common choice. However, the performance improvement in accelerators comes with limitations on programmability, which has become crucial given the rapid evolution of DNN models. In this work, we first conduct workload analysis on a diverse set of DNN models, including CNN, LSTM, Transformer, and GCN to demonstrate the challenges of generalizing DNN acceleration. Next, we present a high-programmable accelerator, referred as FlexACC, with a novel application-specific ISA for flexible DNN inference. To increase the programmability, the general-purpose RISC-V instructions are tightly coupled with DNN instructions in FlexACC ISA. Compared with standalone fixed-datapath CNN and LSTM engines, FlexACC only has small latency and area overhead, while it provides much higher programmability and flexibility.**

***Index Terms*—DNN inference, ISA, Programmability**

I. Introduction

Deep neural networks (DNN) have become ubiquitous in various application domains, such as image classification, speech recognition, natural language processing (NLP), and autonomous driving, for its state-of-the-art learning capabilities. Different types of DNN models have been proposed over the years for different tasks, and some representative ones are shown in Figure 1. In early 2000s, researchers focused on data and image classification problems using multilayer perceptron (MLP) and convolutional neural networks (CNNs) [1]. Around the same time, recurrent neural networks (e.g. GRU, LSTM) have also been explored for time-series analysis [2], [3]. More recently, novel neural network models like Transformer with self-attention mechanism achieve record-breaking accuracy on natural language processing tasks [4]. Emerging DNN models are also published in a rapid rate, such as the generative adversarial network (GAN) for image generation and graph convolutional networks (GCN) for graph data structure analysis [5].

Even though DNN models have shown significant accuracy benefits, they normally consist of millions of MAC computations, which are both compute and memory intensive. Therefore, designing specialized hardware accelerators has become a common choice to accelerate the DNN computation. A variety of DNN accelerators have been developed to achieve orders of magnitude energy efficiency improvement. For example, Minerva [6], Eyeriss [7] are the first few works improving MLP and CNN performance by optimized computation dataflows, quantization, etc. SIMBA [8] is a recent scalable CNN inference accelerator with multi-chip-module chiplet solution. Beyond single DNN model acceleration, reconfigurable DNN accelerators (e.g. DNPU [9] and Thinker [10]) have been studied to support the acceleration of both recurrent neural networks and CNNs. In addition, there are a few accelerators that have been developed to accelerate emerging DNN models, such as GAN [11], NLP [12], [13], GCN [14], [15], and 3D CNN [16]. Although these accelerators achieve state-of-the-art energy efficiency, they mostly have limited programmability with the support of only one or two DNN types, leading to challenges of adapting to the rapid DNN evolution.

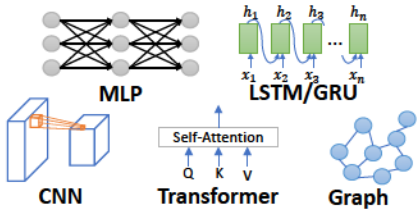

Fig. 1: The diverse types of DNN models.

To provide a flexible and programmable DNN inference solution, in this work, we first analyze the workload breakdown of a diverse set of DNN models, including CNN, LSTM, Transformer, and GCN. Next, we present a high-programmable accelerator, referred as FlexACC, with a novel application-specific ISA for flexible DNN inference. The developed application-specific ISA tightly couples the general-purpose RISC-V with customized instructions for flexible DNN acceleration. Therefore, FlexACC is able to perform flexible memory access and computation patterns with its high programmability. FlexACC is compared with two standalone fixed-datapath (also called hardwired-datapath or ASIC) CNN and LSTM engines. The experimental results show FlexACC only has small latency and area overhead, while it provides much higher programmability and flexibility.

The contributions of our work are as follows:

- A comprehensive workload analysis is conducted on a diverse set of DNN models (i.e., CNN, LSTM, Transformer, and GCN). Different hardware acceleration requirements

978-1-6654-2701-2/21/$31.00 ©2021 IEEE
DOI 10.1109/ASAP52443.2021.00046

for different DNN types are observed.

- A high-programmable accelerator, i.e. FlexACC, is developed with its application-specific ISA, in which RISC-V instructions and customized DNN acceleration instructions are tightly coupled.
- A head-to-head comparison between FlexACC and fixed-datapath designs is conducted to show the small overhead of improving programmability in FlexACC. In addition, a quantitative analysis on the tradeoffs between design spaces and hardware customization choices is provided.

II. Background & Related Works

A. DNN Acceleration Basics

The most dominant computation in DNNs is the multiply-accumulate (MAC). The massive amount of matrix (or matrix-vector) multiply makes DNN suitable for leveraging data parallelism on MAC array-based microarchitecture. However, there is also a variety of model-dependent non-MAC computations in DNNs. For example, in attention-based Transformer model, Softmax is essential and requires complex arithmetic computations, leading to significant software overhead [17]. Other examples of widely used non-MAC computations are Pooling in CNN and Sigmoid/Tanh in GRU and LSTM. To accelerate the non-MAC computations, the SIMD-typed parallelization is often required. Besides computations, DNN models also differ in terms of memory access patterns, complicating the implementation of controllers and address generators in existing works. This essentially shows the need of generalizing controller design in DNN accelerators.

The commonly used compute units for DNNs are shown in Figure 2, ranging from ALU to 3D Tensor Core [18]. The conventional scalar ALU is capable of executing general-purpose scalar operations, providing the highest flexibility but the lowest performance. To parallelize ALU operations, a SIMD unit of vector size N can be utilized to achieve Nx speedup. Apart from generic compute units, 2D MAC Array and 3D Tensor Core are dedicated for MAC operations. The 2D MAC Array is commonly used to perform matrix-vector multiply to achieve a speedup of N^2 [8], [19]. As for 3D Tensor Core, although benefiting from more data reuse and N^3 speedup, it poses more limitations on the shape of tensors, i.e., a minimum batch size of 4 or 8. Thus, we deploy Scalar ALU, 1D SIMD, and 2D MAC Array but not 3D Tensor Core in our FlexACC design.

In the past, most DNN accelerators were designed with a fixed-datapath architecture with limited programmability. A fixed-datapath hardware is normally controlled by a set of configuration registers, which determines the behavior of DNN workloads, memory access patterns, execution cycles, etc. Although fixed-datapath accelerator achieves higher efficiency, tremendous engineering effort to design a complex control interface and underlying architecture is necessary to make it flexible (e.g., NVDLA [19]). In addition, as the accelerator is not directly programmable (with ISA and compiler toolchain), a separated host CPU is needed to control accelerators [8], resulting in complex heterogeneous integration. On the contrary,

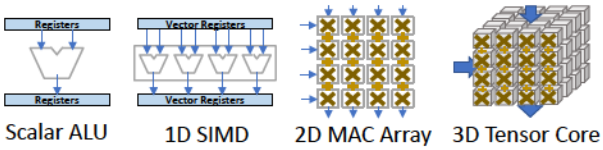

Fig. 2: Different choices of compute units.

TABLE I: Fixed-datapath vs Programmable Designs.

	Fixed-datapath	Programmable
Interface	Config Registers	Instructions
Host CPU	Yes	No
Customization	Higher	Lower
Flexibility	Lower	Higher
SoC Types	Heterogeneous	Homogeneous
Examples	NVDLA, SIMBA	Cambricon

to improve programmability of accelerators, incorporating a customized ISA is more straightforward. With a customized ISA, the tedious configuration interface in the fixed-datapath approach is replaced by application-specific instruction set with a general-purpose controller. Since both general-purpose and specialized instructions are merged into a single design, the accelerator itself is programmable and a host CPU is no longer necessary. Table I summarizes the differences between the two accelerator types.

B. Related Works

One of the earliest works investigating application-specific ISA is the H.264 processor design [20]. The work analyzes the inefficiency of general-purpose processors by comparing several hardware designs for H.264 algorithms with different degrees of specialization. With the help of application-specific instructions, the specialized processor achieves significant speedup over conventional CPUs. However, this work only focuses on the acceleration of classic H.264 algorithms.

In the era of deep learning, Cambricon [21] has a specialized ISA design for DNNs. Despite the promising performance compared to CPU/GPU, the multicycle approach suffers from additional overhead of workload scheduling on an instruction queue unit. Brainwave NPU [22] is another example of specialized ISA for DNN inference focusing more on LSTM/GRU models, but emerging models like Transformer and GCN are not evaluated. A more recent work, NCPU [23], is a programmable accelerator with a shared datapath for both general-purpose RISC-V and binary neural network (BNN). It addresses the overhead of workload offloading between the host CPU and accelerator to improve end-to-end performance, but the solution is limited to BNN inference.

In this work, we propose a high-programmable DNN acceleration solution with a microarchitecture design including RISC-V, ALU, SIMD, and MAC Array to cover a diverse range of DNN inference tasks. With the developed application-specific RISC-V ISA as well as fine-grained single cycle instructions instead of the multicycle approach used in Cambricon ISA [21] and H.264 processor [20], the proposed

FlexACC achieves the same level of performance as fixed-datapath designs. To our best knowledge, there are only a limited number of accelerators designed with application-specific ISA. None of them are able to support a wide range of DNNs as demonstrated by FlexACC.

III. DNN Workload Analysis

To understand the DNN computation requirements, we analyze key computation kernels in various DNN models, including MLP, CNN, GRU/LSTM, Transformer, and GCN. Table II summarizes the DNN workloads profiled and evaluated throughout the paper. Note that unless mentioned specifically, single batch size and output stationary flow are assumed. As DNN computations are dominated by MAC operations, we simply categorize the operations into MAC and non-MAC operations in the profiling. The operational breakdowns of CNN, LSTM, Transformer, and GCN are shown in Figure 3. It is interesting to observe the MAC percentage differences among DNN kernels, ranging from 99% MAC in Conv2D to 86% MAC in GCN. In addition, the detailed requirements on non-MAC computations are quite different. We further explain each workload in the following paragraphs.

MLP and CNN are the most MAC-centric DNN models. Although there are nonlinear operations like ReLU and batch normalization, they are removable during inference [24]. Therefore, in addition to MAC computations, MLP and CNN only require SIMD ALU to perform re-quantization on accumulation values. The convolution operation (Conv2D) can be carried out with loops of matrix-vector multiply as demonstrated in [8], [19].

GRU and LSTM involve matrix-vector multiply similar to MLP with more complex activation functions (e.g. Sigmoid, Tanh) and vector operations. In addition, the re-quantization of accumulation values must be performed before each nonlinear function. Therefore, the breakdown of LSTM shows lower MAC percentage comparing to CNN. To compute GRU and LSTM, acceleration of Sigmoid is necessary ($\tanh(x) = 2\sigma(2x) - 1$). Furthermore, the recurrent data dependency also complicates memory address generation.

Transformers mainly involves two types of DNN kernels, MLP and Self-Attention. While MLP is MAC-centric, the profiling of Self-Attention shows a larger portion of Softmax-related non-MAC computations. Softmax requires complicated exponential-sum and scalar division operations, which are both computation costly and challenging for hardware implementation [17]. Furthermore, Self-Attention requires accelerator to support matrix multiply between two activations, posing another difficulty to datapath and memory access design.

GCN and other graph-based DNNs are designed to process graph-based data structure with edges and vertices (nodes). The basic computation of a GCN layer involves two steps, (1) matrix multiply between weights and each (input) vertex vector and (2) aggregation of neighboring vertices (weighted sum of activation vectors) according to the graph topology [15]. The aggregation step needs to be carried out by SIMD, leading to a large amount of non-MAC operations. Moreover, as the neighboring vertex is determined at runtime with a highly sparse adjacency matrix, the aggregation step results in a costly calculation of irregular memory addresses.

TABLE II: Summary of DNN software test cases.

Workloads	Input	Weight	Output	Sizes
MatMul (MLP)	X[T][Cin]	W[Cout][Cin]	Y[T][Cout]	T=320 Cin=Cout=320
Conv2D (CNN)	X[Hin][Win][Cin]	W[Cout][H][W][Cin]	Y[Hout][Wout][Cin]	H=W=3 Hin=Win=28 Hout=Wout=28 Cin=128, Cout=64
GRU	X[T][Cin]	Wx[3*Cout][Cin] Wh[3*Cout][Cout]	Hidden[T][Cout]	T=160 Cin=Cout=160
LSTM	X[T][Cin]	Wx[4*Cout][Cin] Wh[4*Cout][Cout]	Hidden[T][Cout] Cell[T][Cout]	T=160 Cin=Cout=160
Attention (Transformer)	Q[Head][T][Cin] K[Head][T][Cin] V[Head][T][Cin]	N/A	O[Head][T][Cin]	Head=12 T=64 Cin=64
GCN	X[Node][Cin]	W[Cout][Cin]	Y[Node][Cin]	Node=800 Cin=Cout=64 #Edge=3200

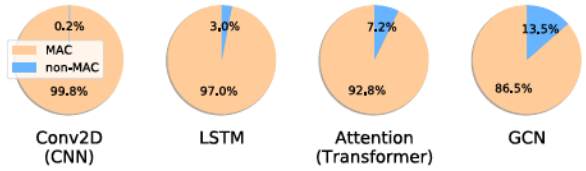

Fig. 3: Percentage of MAC and Non-MAC operations.

Based on the DNN workload study, it is obvious that different DNN models have different non-MAC operations. Although the non-MAC operations only account for a small portion of the entire workload, they require sufficient programmability and flexibility. For MLP and CNN, the data are mostly accessed regularly based on incremental addresses in weight stationary or output stationary dataflows. However, in regard to Transformer and GCN, the memory access shows some irregularity that raises challenges for the accelerator design. It can be anticipated that the memory access will become more sophisticated in future models. Therefore, development of highly flexible DNN accelerators is important to make hardware adaptable to software changes.

IV. FlexACC Accelerator Design

A. Architecture Overview

To develop a highly programmable accelerator for supporting a diverse set of DNN models as well as avoiding the "one-time" design problem in prior fixed-datapath designs, we propose FlexACC with application-specific RISC-V ISA. Figure 4 gives the overview of FlexACC architecture. In the design, a general-purpose RISC-V pipeline is tightly coupled to a variety of DNN acceleration units, and a low-power 3-stage pipeline (IF, ID, EX) is used to implement all 32b RISC-V Base ISA [25] and application-specific instructions.

During computation, the processor control unit (PCU) fetches 64b instructions from the program memory (PM) to issue control signals for both scalar ALU and DNN acceleration units (SIMD and MAC Array). Control signals are issued in fine granularity (i.e. cycle-level) to manage DNN

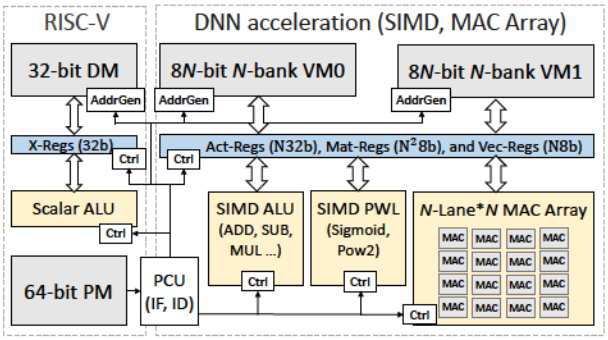

Fig. 4: Overview of FlexACC architecture.

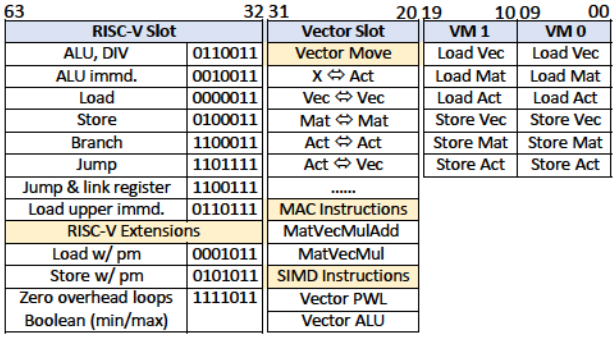

Fig. 5: Developed FlexACC ISA with 4 slots in 64b VLIW.

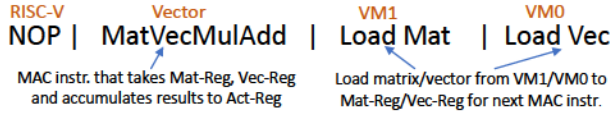

Fig. 6: Simultaneous MAC and VM accesses via ILP.

operations, providing higher flexibility than the prior works with multicycle instructions [21].

Scalar data is loaded from the 32b data memory (DM) into general purpose registers (X-Regs) for scalar operations in scalar ALU. As for accelerating DNN inference, two identical N-bank vector memories (VM0, VM1) are used to store vectorized DNN weights and activations, which can be loaded into vector registers (Vec-Regs), matrix registers (Mat-Regs), or activation vector registers (Act-Regs) for MAC or other vector operations using MAC Array or SIMD units.

B. Application-specific RISC-V ISA

We design an application-specific ISA in a very long instruction word (VLIW) format combining the general-purpose RISC-V ISA with the DNN acceleration instructions, as illustrated in Figure 5. In addition to the RISC-V slot, there are 3 slots (i.e., Vector, VM1, VM0) specifically implemented for computations or memory accesses in DNNs.

Several techniques are leveraged to improve the overall performance. First, instruction level parallelism (ILP) (Figure 6) enables simultaneous DNN acceleration and memory accesses. With ILP, RISC-V instructions are issued in parallel with accelerator instructions, so the overhead of the scalar or control operation can be hidden. Load/store instructions with address postmodify (hardware-based address increment) is another technique used to achieve zero delay during sequential data accesses from DM, VM0, or VM1. Finally, to avoid branch latency, branch instructions are replaced by zero overhead loop (ZLP) for the innermost software loops. The combination of ILP, address postmodify, and ZLP ensures continuous dataflow and zero delay during sequential MAC operations.

The instructions in the Vector slot can be grouped into three subcategories based on their functionality: Vector Move, MAC, and SIMD. Each instruction has two or three operands with 2 or 3 bits of encoding for an operand.

- Vector Move instructions are for data movements between different register instances.
- MAC instructions take an entry of Vec-Reg and Mat-Reg to perform tiled matrix-vector multiply in a cycle. The result is accumulated or stored in an entry of Act-Reg.
- SIMD instructions contain vector-vector and scalar-vector operations. For the later case, the scalar variable is accessed directly from X-Regs.

Lastly, for VM1 and VM0 slots, the Vector Memory instructions give the flexibility to load/store $N^2$8b matrices, N8b vectors or N32b activation vectors from N banks, 1 bank or 4 banks of memory respectively. Since VM0 and VM1 are identical, software programmers have the flexibility to adjust the storage locations for different weights, activations, or other software parameters.

C. 2D MAC Array and SIMD Unit

In FlexACC, an N-lane MAC Array, with N MACs in each lane, is deployed to accelerate DNN workloads by an order of N^2, as shown in Figure 7. During MAC computation, each column of a tiled matrix is mapped onto each lane while a tiled vector is broadcasted to every lane of the MAC Array. MAC performs 8b multiplication, 32b addition and writes the results to Act-Reg in 32b.

For N-lane SIMD, scalar arithmetic is carried out in each lane in parallel so that computations are accelerated by an order of N. The SIMD is able to perform most basic ALU operations, such as multiply (Mul), rational operation, MIN/MAX values, etc. All vector SIMD computations are performed in 32b wide except the vector 24b Mul for reducing hardware area.

Furthermore, to accelerate Sigmoid in LSTM and Softmax in Transformer, a piecewise linear (PWL) SIMD unit is deployed to support the approximation of complex maths via lookup tables (LUT). In the LUT-based implementation, the function output is derived from the base and fractional bit segments of the input, as explained by the following equations where LUT_{base} is the base output and LUT_{frac} refers to the slope.

$$idx = in[11:10] \qquad frac = in[9:0] \tag{1}$$

$$out = \text{LUT}_{base}[idx] + (\text{LUT}_{frac}[idx] * frac >> 10) \tag{2}$$

Take Sigmoid as an example, the input range of $[0, 5)$ is partitioned into 10 segments with negative input values computed as $\sigma(x) = 1 - \sigma(-x)$ in the hardware. The PWL-based

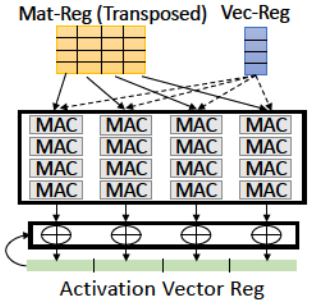

Fig. 7: MAC Array (N=4).

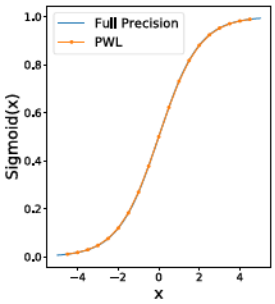

Fig. 8: PWL Sigmoid.

and full-precision Sigmoid is compared in Figure 8, showing an indistinguishable difference between two approaches.

It is worth to mention that although it is possible to implement every nonlinear function with customized instructions, it will complicate the ISA and microarchitecture design. In FlexACC, only the two most common nonlinear functions, Sigmoid and Pow2 (power-of-2), are implemented with customized instructions. Other nonlinear operations are handled by a generic LUT instruction (using an operand of Act-Reg as LUT, and another for parallel indexing). The performance difference between customized instruction and generic LUT will be studied in Section VII-D.

V. Software Mapping & Programming

In this section, the software mapping and programming of FlexACC is explained from three perspectives, including tensor tiling, computation mapping, and sequential and irregular memory accesses.

Tensor tiling is an essential step to map vectors and matrices to vector memories (VM1 and VM0) so that they can be accessed during MAC or non-MAC SIMD operations. Tensor tiling is based on the vector size N of FlexACC. Take matrix-vector multiply as an example, the vector is tiled along one dimension and the matrix is tiled along two dimensions. As shown in Figure 9, a vector or matrix tile can be mapped to 1 bank or N banks of vector memory, respectively. This basic concept can be applied to more complicated DNN kernels such as Conv2D as well as matrix multiply in Attention and GCN.

Computation mapping is performed during the compilation of C code from software to hardware functional units. Figure 10 shows three examples illustrating how low-level C code is mapped to application-specific FlexACC instructions. From the controlling aspect, for-loop can be mapped to ZLP (do-loop) instructions, and while-loop (e.g., for finding neighboring nodes with an index array in GCN) is compiled into jump and branch instructions. On the other hand, to accelerate DNN computations, primitive functions of FlexACC are leveraged in the C code. For instance, the acceleration of Softmax is achieved by utilizing SIMD, Pow2 and Vsum (sum of elements in an Act-Reg variable) primitive functions.

Sequential and irregular memory accesses are processed in different manners in FlexACC. For sequential memory access from scalar DM, vector VM1 and VM0, there are

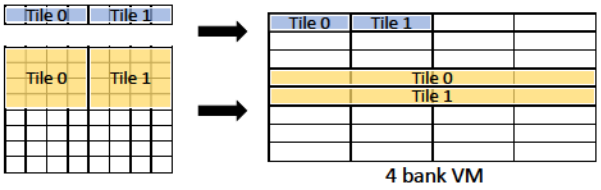

Fig. 9: Tiling and Vector Memory mapping.

Conv2D (Inner-loops)

```
for (h = h_st; h < h_ed; h++)                do (loop) x19, 6
  for (w = w_st; w < w_ed; w++)              do (loop) x23, 3
    for (ic = 0; ic < Cin/N; ic++)           vmac a0, m0, v0
      acc_vec = MatVecMulAdd(……);
```

Attention (Softmax)

```
for (j = 0; j < T/N; j++){                   do (loop) x26, 31
  act = pow2(act);                           pow2 (exp) a0, a0
  sum += vsum(act);                          vsum (sum) x6, a0
}                                            add  x11, x9, x11
inv_sum = 65536 / sum                        div  x11, x5, x11
```

GCN (Aggregation)

```
while (is_end != false) {                    do (loop) x24, 33
  // Aggregate next node                     j (jump) 10
  for (k = 0; k < Cout/N; k++){              ……
    ……                                       vadd a1, a0
  }                                          ……
}                                            bne (branch) x2,x6,-10
```

Fig. 10: Three examples illustrating the mapping of C code (left) to application-specific FlexACC instructions (right). Detailed memory access is omitted for clarity.

hardware-based addresses generators to increment addresses by constant offsets. However, irregular access patterns can only be managed in a software-based approach with additional scalar or control instructions. Therefore, in order to efficiently carry out the computations, the loop structures of a DNN workload are specially arranged such that the memory access of the inner-most loop is sequential. As an example, the input channel dimension can be mapped to the inner-most loop of Conv2D (in an output stationary dataflow). And for the computation of Softmax in Attention and Aggregation (mainly involving weighted sum of activation vectors) in GCN with more irregular memory access patterns, the inner-most loops can still be arranged to be sequential memory access as they are essentially vector-based arithmetic.

VI. Evaluation Methodology

To evaluate the performance of FlexACC, we implement it with ASIP Designer [26], a commercial tool for application-specific processor design, which also provides a software compilation toolchain for simulation with complied C code. The FlexACC microarchitecture is experimented with different vector size choices from 4 to 32 as summarized in Table III. To obtain area and energy results, the RTL of FlexACC is synthesized and evaluated in a commercial 12nm FinFET technology with a commercial 12nm SRAM compiler for memories. The design is also placed and routed in the same 12nm technology.

We also implement the RTL of two standalone fixed-datapath accelerators for Conv2D and LSTM (Fixed-Conv2D

TABLE III: Design parameters of FlexACC.

Vector Size (N)	4	8	16	32
# MACs	16	64	256	1024
# SIMD Lanes	4	8	16	32
VM bit-width	32b	64b	128b	256b
VM entries/bank	16384	4096	1024	256
VM # banks	4	8	16	32
VM total size	256kB	256kB	256kB	256kB

MAC bit-width	8b MUL and 32b ADD
SIMD bit-width	32b ADD/SHIFT..., 24b MUL, 10b PWL
CPU Core	3-Stage RISC-V with extensions
Program Memory	4096x64b
Data Memory	4096x32b
Technology node	12nm
Frequency	500MHz
Operating Condition	TT Corner, 25C, 0.8V

and Fixed-LSTM) to make fair head-to-head comparisons. Similar to the FlexACC microarchitecture, the fixed-datapath designs have a controller unit (with multiple address generators), MAC array, and SIMD unit. However, all components are specialized to compute only one dedicated DNN workload. A host CPU is used for writing the attributes of target workload, such as number of channels, to the configuration registers via 32-bit AXI channel. The area and energy metrics of the baseline accelerators are also collected with the same technology node.

In this work, we focus on accelerator-level analysis and assume instructions and data are preloaded into accelerator memories. We leave system-level analysis, such as multi-core solutions and data movements between accelerators and DRAM, to our future work.

VII. Experimental Results

A. Design Space Exploration

Figure 11 shows the area breakdown of the FlexACC accelerator with different vector sizes ($N = 4, 8, 16, 32$). The total number of multipliers in MAC Array is N^2, so the area of MAC Array becomes more significant when N is increased. On the contrary, the area of SIMD unit only increases linearly with N, and maintains a relative small area percentage for large vector sizes. Although the total memory capacity remains constant, the memory area changes due to different memory structures generated by SRAM compiler. A physical implementation of FlexACC prototype (vector size $N = 8$) is displayed in Figure 12.

Figure 13 (top) shows the performance of test benchmark workloads on FlexACC with different vector sizes. It is observed that MatMul and Conv2D have the highest MAC utilization since the SIMD Unit only needs to process quantization operations. Due to the activation functions (e.g. Sigmoid) and element-wise operations, the ratio of SIMD in GRU/LSTM is larger than CNN. For Attention and GCN models, a large number of SIMD runtime cycles is observed for processing Softmax and vertex aggregation operations. When

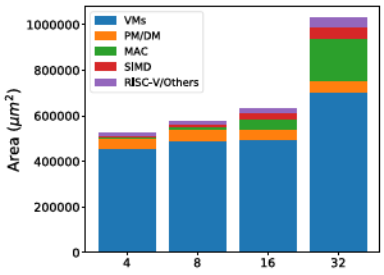

Fig. 11: Area breakdown of FlexACC.

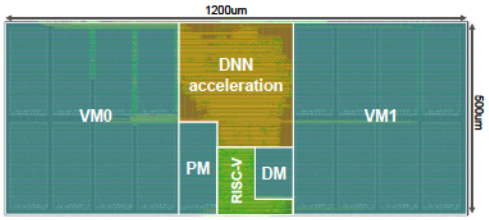

Fig. 12: Physical layout of FlexACC (N=8).

the vector size of both SIMD and MAC array is increased from 4 to 32, it is clearly observed that the latency of MAC computation is significantly reduced while the SIMD and data movement operations become the performance bottleneck. In the experiment, the MAC utilization of 84%, 71%, and 36% is achieved ($N = 8$) for Conv2D, LSTM and Attention, respectively.

Figure 13 (bottom) provides the energy analysis for different test cases. In FlexACC, the output stationary dataflow is adopted, where a vector and a matrix (of size N8b, $N^2$8b) are loaded from VMs for MAC contributing to a large portion of total energy. However, for larger vector size designs, the energy of VMs decreases as the SRAM becomes more energy efficient with a larger word size, and input activation vectors are reused more through broadcasting in MAC Array.

In summary, for MAC-centric DNN kernels like Matmul, Conv2D, GRU and LSTM, more performance and energy benefits are obtained by using a larger vector size ($N = 16, 32$) to accelerate MAC operations. While the speedup and energy reduction are limited for Attention and GCN when sweeping the vector size from 4 to 32, as there is more non-MAC operations. With configurable vector size N, FlexACC is adjustable for different computational requirements.

B. Comparison with Fixed-Datapath Designs

In Figure 14, we compare the FlexACC area ($N = 8$) with a standalone RISC-V pipeline and two fixed-datapath accelerators. The area of RISC-V core (with only standard RISC-V instructions) is also shown to illustrate the area cost of having a separated host CPU in a heterogeneous SoC. Although the core area of Fixed-Conv2D is much smaller (36% less) than FlexACC for Fixed-Conv2D does not have SIMD PWL functions, FlexACC only has a small area overhead (17%) compared to Fixed-LSTM. It is also interesting to observe that the RISC-V (and others) area of FlexACC is only 20%

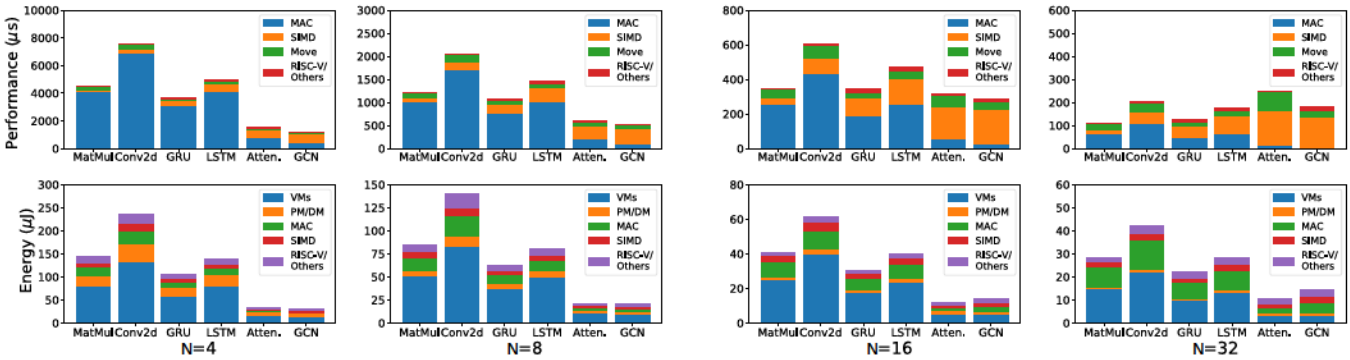

Fig. 13: Performances (top) and energy (bottom) breakdown of FlexACC on different test cases.

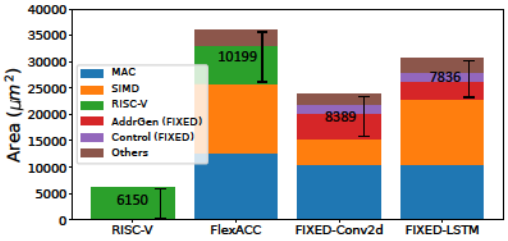

Fig. 14: Core area comparison of FlexACC and Fixed-Datapath engines with annotation on control units sizes (N=8).

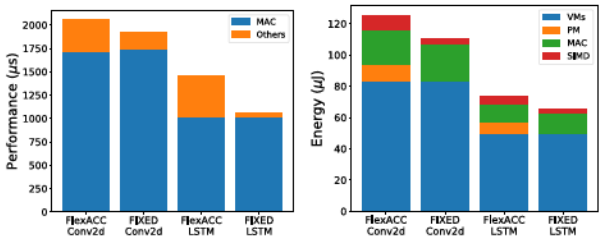

Fig. 15: Performance (left) and energy (right) comparisons of Programmable and Fixed-Datapath (N=8).

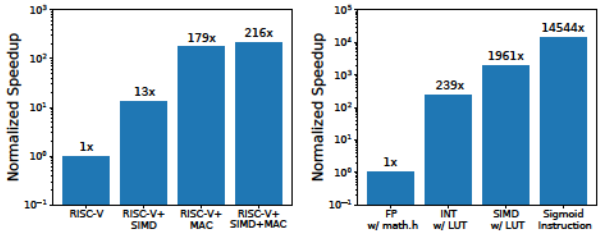

Fig. 16: Normalized speedup. (left) Conv2D with different compute units; (right) Different methods for Sigmoid. (N=8).

and 30% larger than the combined area of address generator (AddrGen) and Controller in fixed-datapath designs, indicating that customized memory access and control units are hardware costly. Through the proposed programmable FlexACC, conventional heterogeneous integration with a separate host CPU is not needed, making FlexACC potentially more hardware efficient at SoC integration level.

The performance and energy comparisons are presented in Figure 15. Compared with fixed-datapath accelerators, FlexACC only has 10% and 30% latency overhead. The performance gap, is mainly related to not parallelizing MAC and SIMD computations in FlexACC, which is another design space to be explored in the future. The energy breakdown shows that the main difference between the two types of accelerators comes from the instruction fetch of PM. The total energy consumption of FlexACC is only 15% and 11% higher than that of fixed-datapath Conv2D and LSTM designs.

In summary, FlexACC can achieve the same level of performance and energy-efficiency as fixed-datapath accelerators with moderate cost in terms of hardware area to achieve the benefit of high programmability.

C. Dataflow and Weight Reuse

To improve weight reuse and reduce total memory access, the "for-loops" of a workload can be rearranged from stationary (OS) into a weight stationary (WS) dataflow. Contrary to OS, WS accesses different input vectors and output activations while keeping the same weight matrix in Mat-Reg. The main cost of WS dataflow is that activation vectors need to be reloaded for accumulation. By replacing OS with WS, the memory accesses corresponding to each MAC instruction changes from $N^2$8b+N8b (OS) to N8b+N32b+N32b (WS), meaning that the total number of VM accesses can be reduced when $N > 8$. Simulation results show that WS dataflow can reduce VM accesses to 0.67x and 0.38x for $N = 16$ and 32 at the cost of 2.0x and 1.7x latency compared to OS dataflow. With fine-grained instructions and programmable datapaths, FlexACC has the flexibility to support different dataflows.

D. Customization and Speedup Analysis

To explain how each acceleration hardware unit impacts the overall performance improvement, we further compare the performance of Conv2D with the following settings: (1) Standalone RISC-V, (2) RISC-V+SIMD, (3) RISC-V+MAC, and (4) RISC-V+SIMD+MAC. Figure 16 (left) shows an overall of 216x speedup over RISC-V when all acceleration components (RISC-V+SIMD+MAC) are in place. It is software dependent to determine the contributions of SIMD and MAC Array to the overall performance, but the analysis shows the importance of having both SIMD and MAC Array in the design.

Careful hardware implementation is often needed to accelerate complex math functions in DNNs [17], but over-

customization may result in area overhead and reduction of flexibility. Figure 16 (right) compares the performance of Sigmoid with different choices of customization: (1) floating point on CPU, (2) integer approximation on CPU, (3) SIMD with generic LUT instructions, and (4) SIMD with customized instruction. It is observed that substantial speedup can be obtained through both software and hardware customization. However, there is a tradeoff between performance and flexibility in choosing implementation methods. FlexACC finds a good balance for the tradeoff by using SIMD with customized instructions for the two most common nonlinear functions, Sigmoid and Pow2, and other nonlinear functions can be handled by a generic LUT instruction on SIMD.

VIII. Conclusion

Conventional fixed-datapath DNN accelerators often suffer from limited programmability. In this work, we propose and implement FlexACC accelerator with an application-specific ISA for DNN inferences. To increase the accelerator programmability, the proposed application-specific ISA tightly couples the general-purpose RISC-V with customized DNN instructions. The experimental results affirm FlexACC is capable of performing a wide range of DNN inferences with decent performance and MAC utilization. A head-to-head comparison to fixed-datapath baselines further reveals that FlexACC has moderate overhead to achieve high programmability.

Acknowledgements

We thank David Florez (from Synopsys) for his advice in hardware implementation with ASIP Designer. This work is supported in part by the Application Driving Architectures (ADA) Research Center, a JUMP Center cosponsored by SRC and DARPA, DSSoC programs, and the NSF under CCF-1704834 and CCF-1718160.

References

[1] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[2] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[3] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *arXiv preprint arXiv:1706.03762*, 2017.

[5] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[6] B. Reagen, P. Whatmough, R. Adolf, S. Rama, H. Lee, S. K. Lee, J. M. Hernández-Lobato, G.-Y. Wei, and D. Brooks, "Minerva: Enabling low-power, highly-accurate deep neural network accelerators," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2016, pp. 267–278.

[7] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2016, pp. 367–379.

[8] Y. S. Shao, J. Clemons, R. Venkatesan, B. Zimmer, M. Fojtik, N. Jiang, B. Keller, A. Klinefelter, N. Pinckney, P. Raina *et al.*, "Simba: Scaling deep-learning inference with multi-chip-module-based architecture," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 14–27.

[9] D. Shin, J. Lee, J. Lee, and H.-J. Yoo, "14.2 dnpu: An 8.1 tops/w reconfigurable cnn-rnn processor for general-purpose deep neural networks," in *2017 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2017, pp. 240–241.

[10] S. Yin, P. Ouyang, S. Tang, F. Tu, X. Li, L. Liu, and S. Wei, "A 1.06-to-5.09 tops/w reconfigurable hybrid-neural-network processor for deep learning applications," in *2017 Symposium on VLSI Circuits*. IEEE, 2017, pp. C26–C27.

[11] S. Kang, D. Han, J. Lee, D. Im, S. Kim, S. Kim, and H.-J. Yoo, "7.4 ganpu: A 135tflops/w multi-dnn training processor for gans with speculative dual-sparsity exploitation," in *2020 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 2020, pp. 140–142.

[12] T. Tambe, E.-Y. Yang, G. G. Ko, Y. Chai, C. Hooper, M. Donato, P. N. Whatmough, A. M. Rush, D. Brooks, and G.-Y. Wei, "9.8 a 25mm 2 soc for iot devices with 18ms noise-robust speech-to-text latency via bayesian speech denoising and attention-based sequence-to-sequence dnn speech recognition in 16nm finfet," in *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 64. IEEE, 2021, pp. 158–160.

[13] T. J. Ham, S. J. Jung, S. Kim, Y. H. Oh, Y. Park, Y. Song, J.-H. Park, S. Lee, K. Park, J. W. Lee *et al.*, "A^ 3: Accelerating attention mechanisms in neural networks with approximation," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020, pp. 328–341.

[14] B. Zhang, H. Zeng, and V. Prasanna, "Hardware acceleration of large scale gcn inference," in *2020 IEEE 31st International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 2020, pp. 61–68.

[15] M. Yan, L. Deng, X. Hu, L. Liang, Y. Feng, X. Ye, Z. Zhang, D. Fan, and Y. Xie, "Hygcn: A gcn accelerator with hybrid architecture," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020, pp. 15–29.

[16] H. Fan, C. Luo, C. Zeng, M. Ferianc, Z. Que, S. Liu, X. Niu, and W. Luk, "F-e3d: Fpga-based acceleration of an efficient 3d convolutional neural network for human action recognition," in *2019 IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, vol. 2160. IEEE, 2019, pp. 1–8.

[17] Z. Wei, A. Arora, P. Patel, and L. John, "Design space exploration for softmax implementations," in *2020 IEEE 31st International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 2020, pp. 45–52.

[18] Z. Jia, M. Maggioni, B. Staiger, and D. P. Scarpazza, "Dissecting the nvidia volta gpu architecture via microbenchmarking," *arXiv preprint arXiv:1804.06826*, 2018.

[19] NVIDIA, "Nvidia deep learning accelerator (NVDLA)," 2018.

[20] R. Hameed, W. Qadeer, M. Wachs, O. Azizi, A. Solomatnikov, B. C. Lee, S. Richardson, C. Kozyrakis, and M. Horowitz, "Understanding sources of inefficiency in general-purpose chips," in *Proceedings of the 37th annual international symposium on Computer architecture*, 2010, pp. 37–47.

[21] S. Liu, Z. Du, J. Tao, D. Han, T. Luo, Y. Xie, Y. Chen, and T. Chen, "Cambricon: An instruction set architecture for neural networks," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2016, pp. 393–405.

[22] J. Fowers, K. Ovtcharov, M. Papamichael, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, L. Adams, M. Ghandi *et al.*, "A configurable cloud-scale dnn processor for real-time ai," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 1–14.

[23] T. Jia, Y. Ju, R. Joseph, and J. Gu, "Ncpu: An embedded neural cpu architecture on resource-constrained low power devices for real-time end-to-end performance," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 1097–1109.

[24] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2704–2713.

[25] A. Waterman, Y. Lee, D. A. Patterson, and K. Asanović, "The risc-v instruction set manual, volume i: User-level isa, version 2.1," 2016.

[26] Synopsys, "ASIP Designer," https://www.synopsys.com/designware-ip/processor-solutions/asips-tools.html.