NeuralScale: A RISC-V Based Neural Processor Boosting AI Inference in Clouds

Rongkai Zhan, Xiaobo Fan
{mark,shawn}@streamcomputing.com
Stream Computing Inc.
Beijing, China

ABSTRACT

There has been a tremendous increase in AI chips for cloud computing over the last few years. Most of the specialized AI chips are designed to optimize specific applications with poor programmability and flexibility. In this paper, we present NeuralScale, a RISC-V based neural processor core architecture for AI inference in clouds that prompts programmability significantly with RISC-V vector extensions while retaining competitive performance and efficiency. Our industrial product P920, implemented with 32 NeuralScale cores, achieves 256TOPS (INT8) and 128TFLOPS (FP16) peak performance under a clock frequency of 1.0 GHz in a TSMC 12nm FinFET process technology. Evaluation results on typical inference workloads show that our processor delivers state-of-the-art throughput and power efficiency performance.

CCS CONCEPTS

• **Computer systems organization** → **Neural networks**.

KEYWORDS

RISC-V, vector, extension, processor, AI, inference

1 INTRODUCTION

AI and its diverse applications have seen significant increasing demand for AI computing in clouds over the last few years. Typical AI-enabled services include image and speech recognition, natural language processing, medical diagnosis, visual search, and personalized recommendations. AI computing in clouds includes two distinct workloads: training and inference. Training is the process of creating neural network models and fine-tuning models through a particular training algorithm with large amounts of relevant raw data. Inference is the deployment of the trained models for real-time evaluations of new data. Both training and inference are compute-intensive but they require different compute resources. The training process typically uses 32-bit single-precision floating-point or 64-bit double-precision floating-point data formats and ALUs to achieve a high level of accuracy. By contrast, the inference process can use much lower precision data formats and ALUs like 8-bit integer, 16-bit integer and 16-bit half-precision floating-point while introducing a very limited drop in prediction accuracy. It is reported that the inference workloads constitute more than 95% of AI computing workloads in clouds [20].

Currently, AI computing workloads in clouds are mainly performed on central processing units (CPUs) and graphics processing units (GPUs). Meanwhile, we have seen significant growth in FPGA-based accelerators [8] [16] [21] and application-specific integrated circuits (ASICs) [3] [4] [7] [10] for AI inference in clouds. CPUs are general-purpose processors with good programmability and almost can do anything but are not suitable for compute-intensive workloads. GPUs with massively parallel computing ability are suitable for AI computing and almost all AI models are trained using GPUs. However, the high-precision floating point ALUs are redundant for the inference workloads which lift the total cost of ownership and increase power consumption. FPGAs have flexible programmability which allow developers to create custom hardware accelerators quickly and can be reprogrammed to adapt to the evolving AI landscape. On the other hand, the reconfigurability limits the clock frequency, and hence FPGAs are less competitive than GPUs or ASICs in terms of performance and energy efficiency. ASICs with customized silicon and memory hierarchy for specific models or algorithms have shown better performance and energy efficiency than traditional GPUs in AI inference. However, most ASICs can only perform designed models or algorithms with poor or even no flexibility and programmability. As AI algorithms are still evolving, new operators and activation functions may be proposed. It will become a great challenge for ASICs when migrating to new AI models in the future.

To promote the programmability of ASICs while retaining the performance and energy efficiency, we propose a general-purpose neural processor architecture based on the RISC-V ISA [18], as RISC-V is meant to provide a basis for more specialized instruction-set extensions or customized accelerators. The sketch map in Figure 1 shows the key components of the proposed processor architecture: a RISC-V scalar core, a vector execution unit, and a matrix execution unit. The scalar core acts as the control processor, fetching and decoding all the instructions. The vector execution

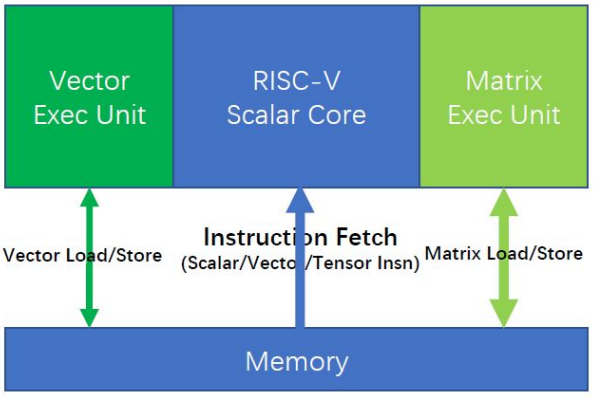

Figure 1: A sketch map of the proposed general-purpose neural processor core architecture.

unit conducts standard vector operations. The matrix execution unit conducts matrix operations, e.g. GEMM operation, which is at the heart of AI inference workloads. An industrial product configured with 32 NeuralScale cores is implemented in a TSMC 12nm FinFET process technology. Evaluation results on ResNet-50 V1.5 and BERT inference show that our processor achieves state-of-the-art throughput performance, latency performance, and energy efficiency for both CNN and NLP models.

2 NEURALSCALE ARCHITECTURE

2.1 Scalar Core

We adopt the AndesCore N25F core [12] as the scalar core, which is a 32-bit RISC-V CPU IP core with vector extension support. The scalar core has a 5-stage in-order execution pipeline and separated instruction and data caches. Features also include dynamic branch prediction for efficient branch execution. It is capable of delivering high per-MHz performance and operating at high frequencies with small gate counts. Figure 2 illustrates a high-level overview of the NeuralScale architecture. As we can see from the control flow, the scalar core fetches and decodes all instructions, and divert the instructions to the correct path based on their types. Scalar instructions are executed in order in the scalar pipeline while vector instructions flow through the scalar pipeline to the neural processor core.

2.2 Neural Processor Core

The neural processor core combines the features of vector processors and AI inference accelerators. As shown in Figure 2, the computation components include a MAC vector for executing vector operations, a MAC matrix for executing matrix operations, and a POLY module for complex arithmetics like exp, div, and sqrt computations. On-chips memory components include a vector register file (the REG Bank module) as well as three local buffers, named L1 Buffer, Data Input Buffer, and Intermediate Buffer respectively.

The neural processor core's pipeline is divided into 4 stages in concept: decode, issue, execute, and write-back. As the control flow in Figure 2 shows, the scalar core diverts the vector instructions to the neural processor core. Vector instructions are further decoded into micro-ops in the decode unit and then dispatched to the issue unit. The issue unit issues instructions to corresponding execution units based on their operation types. There are three execution units, a vector MAC engine (VME), a matrix MAC engine (MME), and a memory transmission engine (MTE) for different operation types, as will be explained. The issue unit maintains three instruction buffers tracking the state of all inflight instructions in each execution unit. A dispatched instruction from the decode unit will be buffered according to its operation type and will be removed once it's committed by the execution unit. All instructions will be issued in order, and an instruction can be issued only when there is no address overlap with inflight instructions. The issue unit can issue three instructions at most. All three execution units can work simultaneously in this case, and hence memory latency can be partially hidden by computation, which lifts overall performance. After execution, the results will be written back to vector registers or local buffers.

2.2.1 VME. VME performs all base vector extension instructions and part of customized vector extension instructions with the MAC vector and POLY module. The MAC vector consists of a vector of multiply-and-add units, which supports both 16-bit half-precision floating-point (FP16) arithmetics and 8-bit integer (INT8) arithmetics. The POLY module contains exp, div, and sqrt function units for the complex arithmetics in activation functions or classifiers during the AI inference. As we can see from the data flow, the source operands for VME may come from vector registers or local buffers, and the results may be written back to vector registers or local buffers. When performing base vector extension instructions, VME reads source operands from specified source vector registers and writes results back to the destination vector register. For customized vector extension instructions, VME reads source operands from L1 Buffer or Intermediate Buffer based on the addresses specified by the source general-purpose registers, and writes results back to L1 Buffer or Intermediate Buffer based on the address specified by the destination general-purpose register. Final results will usually be written back to the L1 Buffer.

2.2.2 MME. MME performs customized vector extension instructions related to matrix or convolution operations with the MAC matrix. Notice that convolution operations are implemented with the GEMM algorithm. The MAC matrix is comprised of $m \times n$ multiply-and-add units, with m indicates the height of the MAC matrix and n indicates the width of the MAC matrix. Each MAC unit supports both FP16 and INT8 arithmetics. When fully utilized, the MAC matrix can compute $m \times n$ FP16 arithmetics or $2 \times m \times n$ INT8 arithmetics simultaneously. That's why the MAC matrix is the most important contributor to compute power of AI chips. MME reads source operands from Data Input Buffer and the Weight Buffer in L1 Buffer, and writes results back to Intermediate Buffer. Input buffer fetches data from the Data IO Buffer in L1 Buffer or Intermediate Buffer depends on the addresses specified by the source general-purpose register.

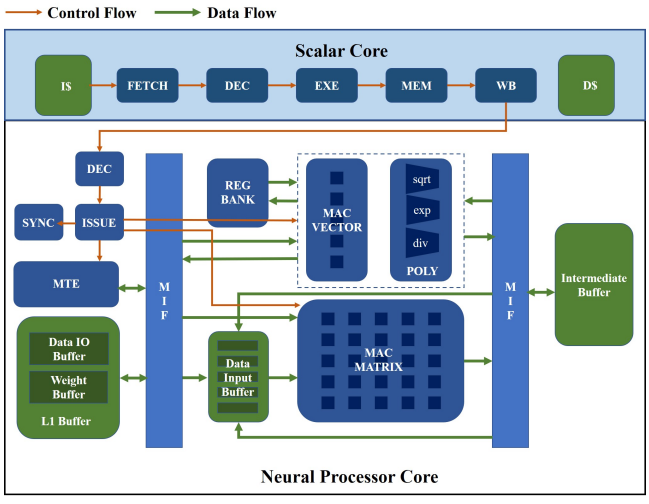

Figure 2: A high-level overview of the NeuralScale architecture.

Table 1: Customized vector CSRs

Base Address	Privilege	Register Name	Execution Units	Function Description
0x400	URW	shape_s1	VME	[31:16], width of matrix A; [15:0], height of matrix A
0x401	URW	shape_s2	VME	[31:16], width of maxrix B; [15:0], height of matrix B
0x408	URW	conv_FM_in	MME	[31:16], width of input features; [15:0], height of input features
0x409	URW	conv_Depth_in	MME	[15: 0], depth of input features
0x422	URW	mte_shape	MTE	[31:16], width of transfer matrix; [15:0], height of transfer matrix

2.2.3 MTE. MTE connects local L1 Buffer with other memory components outside the core through NoCs. Outside memory components include remote L1 Buffers in a multi-core scenario and peripheral memory like the last level buffer (LLB) or external DDR DRAMs. In the first case, MTE exchanges data between L1 Buffers in a point-to-point manner. In the second case, MTE can exchange data between L1 Buffer and LLB in a point-to-point manner, or broadcast the data in LLB to all corresponding L1 Buffers.

2.3 Instruction-Set Extension

The RISC-V Vector extension (RVV) [19] enables processor cores based on the RISC-V instruction set architecture to process data arrays, alongside traditional scalar operations to accelerate the computation of single instruction streams on large data sets. The scalar core adopted in our processor implements the RV32G. Therefore, we implement standard extensions including the base RVV extension (v0.8) and customized vector extensions with fixed-width 32-bit instruction format in our neural processor core. We use the custom-3 opcode (11111011) in the RISC-V base opcode map as the major opcode for customized vector extensions, marked as **OP-VE**. All customized vector extensions keep the source (rs1 and rs2) and destination (rd) registers at the same position as the base RISC-V ISA does to simplify decoding, as shown in Table 2.

Table 2: Format for customized vector extension

31 26	25	24 20	19 15	14	13 12	11 7	6 0
funct6	dmc	rs2	rs1	dm	opm2	rd	1111011

The **opm2** field encodes the source operand types and source locations, as listed in Table 3. For a vector or matrix operand, the general-purpose register provides the memory address of the values, marked as (rsx).

Table 3: opm2[1:0] encoding

opm2[1:0]	source operands types	source1	source2
00	mm (matrix-matrix)	(rs1)	(rs2)
01	m (matrix)	(rs1)	null
10	mv (matrix-vector)	(rs1)	(rs2)
11	mf (matrix-scalar)	(rs1)	rs2

The matrix operation directions are encoded using the **dmc** and **dm** fields, as displayed in Table 4. Taking matrix-vector additions for example, $\{dmc, dm\} = 10$ indicates adding a matrix with a row vector while $\{dmc, dm\} = 01$ indicates adding a matrix with a column vector.

Table 4: dmc and dm encoding

dmc	dm	Operation directions
0	X	matrix operations on full elements
1	0	matrix operations vertically on row vectors
0	1	matrix operations horizontally on column vectors

The **funct6** field encodes operation types, including addition, subtraction, multiplication, accumulation, etc. Some **funct6** codes are listed in Table 5 for illustration. Typical operations such as convolutions and activation functions in AI inference workloads are all covered.

Table 5: funct6 encoding

funct6	Name	Descrition
000001	veadd	add
000010	vesub	subtract
000011	veacc	accumulate
000101	veemul	element-wise multiply
011001	memul	matrix multiply
000110	veemacc	element-wise multiply-accumulate
011010	meconv	convolution
001001	velkrelu	Leaky Relu activation function
001011	mov	transfer data with MTE

A total of 53 customized instructions are extended in addition to the base RVV extension. For many matrix-related operations, information such as height and width of the matrix cannot be encoded within the 32-bit fixed-width instruction. Therefore, 22 unprivileged vector CSRs are added to the base RVV extension. Table 1 lists several of them for illustration. Customized vector CSRs can

only be updated with CSR instructions defined in the base scalar RISC-V ISA. The values should be properly set to match application needs.

3 SOC PLATFROM IMPLEMENTATION

Based on the NeuralScale architecture, we implement an industrial SoC platform named P920 for AI inference in clouds. A complete toolchain suite including graph compiler, runtime, and driver is also released for developers.

3.1 Core Configuration

P920 consists of 32 NeuralScale cores and the configuration of each core is listed in Table 6. The scalar core has a separated L1 Data Cache and L1 Instruction Cache, each of 64KB. The neural processor core has a 1MB L1 Data IO Buffer, a 256KB L1 Weight Buffer and a 256KB Intermediate Buffer. The size of each local buffer in the neural processor core is selected based on experimental statistics of typical AI inference workloads, which helps to avoid frequently exchanging data between on-chip memory and external memory. The MAC vector in the neural processor core has 64 MAC units, and the MAC matrix in the neural processor core contains 64×32 MAC units. Each MAC unit supports both FP16 and INT8 arithmetics, which can be dynamically switched according to the operation type of each instruction.

Table 6: NeuralScale Core Configurations

feature		configuration
scalar core	L1 Data Cache	64KB
	L1 Instruction Cache	64KB
neural processor core	L1 Data IO Buffer	1MB
	L1 Weight Buffer	256KB
	Intermediate Buffer	256KB
	(VLEN, ELEN)	(1024, 16)
	MAC Vector	64 FP16 MACs
	MAC Matrix	64×32 FP16 MACs

3.2 SoC Architecture

Figure 3 shows a high-level overview of the P920 architecture. The key components include 32 NeuralScale cores, a 32MB last level buffer (LLB), a hardware synchronization (HSYNC) subsystem, two PCIe subsystems, four DDR subsystems, a peripheral subsystem, and a CPU subsystem. All components are connected through an NoC with a regular 4×6 mesh-based topology. The links between each component and an NoC router, and the links between NoC routers are all bidirectional. The NoC separates control flow and data flow to lift data transmission efficiency. The control bus is 32 bits wide in each direction and the data bus is 512 bits wide in each direction. At 1.0 GHz, each direction provides up to $64GB/s$ bandwidth or $128GB/s$ combined.

The 32MB LLB is split up into eight separated small LLBs of 4MB each. The small LLBs are connected to the NoC independently, providing $1TB/s$ memory bandwidth in total. Meanwhile, they are evenly distributed in the NoC so that other nodes can access an LLB within a small latency. As there are 32 NeuralScale cores in total, an HSYNC subsystem is used to manage how these cores cooperate and synchronize. NeuralScale cores can be divided into up to 16 groups by the HSYNC subsystems, and the number of cores in each group is configured by the application. That is to say, an application can be performed either on one group with 32 cores or on multiple groups with several cores in each group. The HSYNC subsystem provides great flexibility and hence an application can choose the granularity of task division according to its features to make full use of the NeuralScale cores.

P920 has two PCIe subsystems: PCIE0 and PCIE1. Each PCIe subsystem supports up to 16 lanes and can be configured as an endpoint or a root complex. PCIE0 is usually configured as an endpoint, receiving compute tasks and data from the host. PCIE1 is usually configured as a root complex for scalability, connecting to other SoC chips to construct a larger-scale compute platform. In addition, P920 has four DDR subsystems. Each subsystem has an independent channel of LPDDR4 DRAM, supporting up to 4GB memory capacity and $4266MT/s$ transfer rate. Therefore, the DDR subsystems provide 16GB memory capacity in total and a peak theoretical bandwidth of $136GB/s$ for AI inference workloads. During the AI inference process, LLBs need to fetch weights from DRAMs frequently. In order to improve the data transmission efficiency between LLBs and DRAMs, high-performance DMA controllers are integrated into the DDR subsystems. Each DMA controller connects a DDR controller and an LLB through NoCs. As there are eight LLBs and only four DDR channels, each DDR subsystem integrates two DMA controllers with independent DMA channels and data buses.

The peripheral subsystem implements many common hardware devices including UART, SPI, I2C, PWM, and RTC, which plays an important role in booting, debugging, and managing the SoC. The CPU subsystem is implemented with an ARM Cortex-A53 core [5]. It features an in-order, 8-stage, dual-issue pipeline, and supports PPI interrupts and up to 64 SPI interrupts. The CPU subsystem is mainly used to initialize a series of devices during the SoC startup, including PCIe controllers, DDR controllers, SPI controllers, and other devices. Besides, it also monitors and manages the SoC during running.

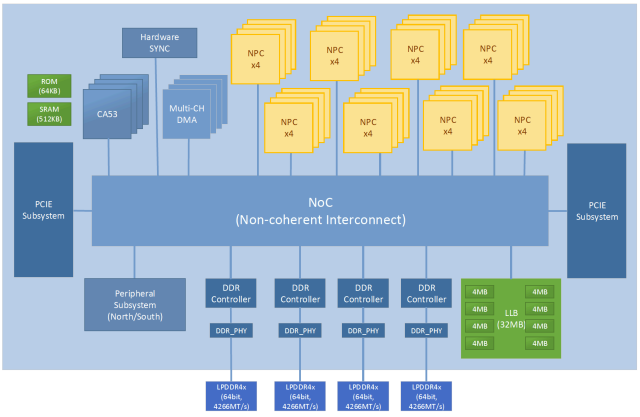

Figure 3: A high-level overview of the P920 architecture.

3.3 Toolchain

We implement an end-to-end inference stack named TensorTurbo for P920 that enables fast and efficient deployment of customers' pre-trained AI models, as shown in Figure 4. TensorTurbo is mainly comprised of a graph compiler and a heterogenous program engine (HPE). The graph compiler is based on TVM [2] and has been deeply customized for NeuralScale architecture. It provides C++ and python inference API for popular deep learning frameworks including TensorFlow, PyTorch, MxNet, and Keras. Graph intermediate representations (GIRs) from different frameworks are imported as unified TensorTurbo IRs via the inference API. The graph compiler then applies graph schedule, operators schedule, tiling strategies within an operator, among other optimizations to find the fastest implementation leveraging the hardware features at the most. The HPE provides high-level CUDA-style runtime APIs in the hardware abstraction layer (HAL), enabling functions like device management, kernel launch and management, memory management, etc. The HPE also provides utilities including GDB debug tool, performance profiling tool, and system monitor interface tool via accessing P920's debugging features (event logging, performance counters, breakpointing).

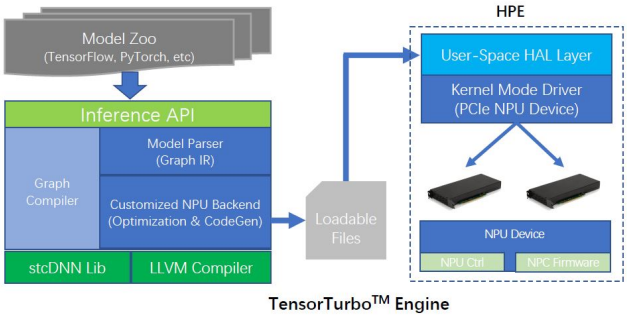

Figure 4: Deployment of pre-trained AI models onto P920 with TensorTurbo.

4 EVALUATION

P920 was fabricated using TSMC's 12nm FinFET technology and the total area is $400mm^2$. It delivers 256 TOPS (INT8) and 128 TFLOPS (FP16) peak compute performance with a thermal design power of $130W$ under 1.0 GHz working frequency. We conduct a detailed performance evaluation of P920 with two typical AI inference workloads in clouds: the ResNet-50 CNN model for vision tasks and the BERT model for NLP tasks. As a comparison, experiments are also conducted on two GPU devices (Nvidia T4 [13] and Nvidia V100 [14]) and an AI chip (Habana Goya [9]). All platforms are installed into a server host through PCIe slots. The deep learning framework used in our experiment is Tensorflow [1].

4.1 Performance on ResNet-50

The ResNet-50 model [11] is one of the most popular CNN models for visual tasks including image classification, object localization, object detection, and others. It has multiple versions and we choose the public ResNet-50 v1.5 [17] for Tensorflow to perform image classification tasks in our experiment. The Nvidia V100 GPU performs ResNet-50 inference with FP16 computations as it provides much higher FP16 performance than INT8. The other three platforms perform ResNet-50 inference with INT8 computations. Besides, the batch size is properly configured to fully exploit the compute performance of each platform: 128 for GPUs, 10 for the Habana Goya chip, and 64 for P920. The performance results include throughput, power efficiency, and latency of the four platforms are shown in Figure 5. Our P920 chip can process 14442 images per second (IPS), which is 2.98 times more powerful than the Nvidia T4 GPU, 1.85 times more powerful than the Nvidia V100 GPU, and nearly the same as Habana Goya chip. With a thermal design power of $130W$, P920's power efficiency is 110 IPS/W, which is 1.59 times of the Nvidia T4 GPU, 4.23 times of the Nvidia V100 GPU, and 1.50 times of the Habana Goya chip. In terms of latency performance, the Habana Goya chip has the shortest latency with only 0.87 ms. The latency of P920 is 4.43 ms, 5.87 times shorter than the Nvidia T4 GPU, and 3.61 times shorter than the Nvidia V100 GPU.

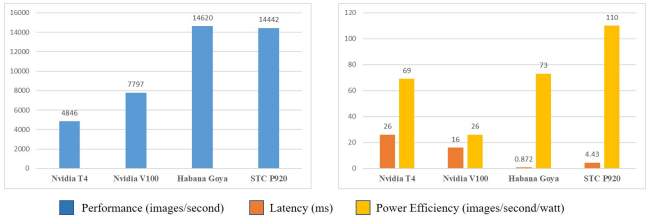

Figure 5: Performance results on ResNet-50 inference.

4.2 Performance on BERT

BERT [6] is a state-of-the-art language model for NLP tasks including text classification, question answering, natural language inference, and others. The BERT model used in our experiment is fine-tuned for sentence and sentence-pair classification tasks. Our P920 chip performs BERT inference with FP16 computations due to the poor accuracy performance of the quantized INT8 BERT model.

The GPUs use INT8 mixed-precision computations and the Habana Goya chip uses INT16 computations. We select a batch size of 128 for GPUs, 12 for the Habana Goya chip, and 32 for P920. The performance results are shown in Figure 6. Our P920 chip can process 4192 sentences per second, which is 2.31 times more powerful than the Nvidia T4 GPU, 1.31 times more powerful than the Nvidia

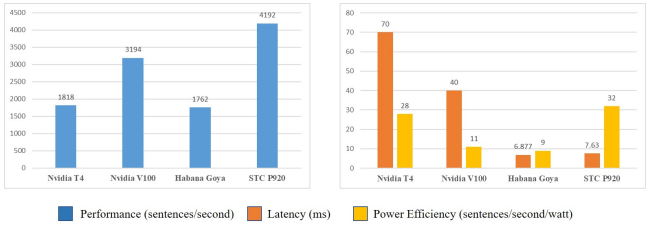

Figure 6: Performance results on BERT inference.

V100 GPU, and 2.37 times more powerful than the Habana Goya chip. The power efficiency of P920 is 32 sentences per second per watt, which is 1.14 times of the Nvidia T4 GPU, 2.91 times of the Nvidia V100 GPU, and 3.56 times of the Habana Goya chip. The latency of P920 is 7.63 ms, 9.17 times shorter than the Nvidia T4 GPU, 5.24 times shorter than the Nvidia V100 GPU, and very close to the latency of the Habana Goya chip.

4.3 Trace Analysis

The runtime tool traces the instructions and performance counters for further profiling using P920's debugging features. Figure 7 illustrates the profiling results of P920's performance on BERT inference. During P920's running, scalar cores process scalar instructions for control, neural processor cores process compute-intensive vector instructions, and DMAs transfer data between LLBs and external DRAMs. As expected, processing in neural processor cores takes up most of the total time, 95%. DMAs work in parallel with scalar cores or neural processor cores in 96% of its total time, showing that external memory transfers are very well overlapped with computations in NeuralScale cores.

Inside of the neural processor cores, MME units take up 78% of the total cycles. MTE units work in parallel with MME units, VME units or both in 92% of its total time, showing that on-chip memory transfers between LLBs and L1 Buffers are also well overlapped with computations. VME units work in serial with MTE units and MME units in 45% of its total time mainly due to data dependencies with MME units, which is the focus of our future optimization.

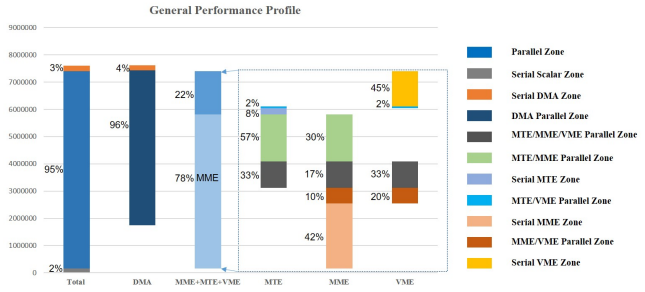

Figure 7: BERT Performance Profile

5 CONCLUSION

In this paper, we present NeuralScale, a neural processor core architecture based on RISC-V ISA for AI inference in clouds. NeuralScale takes advantage of customized RISC-V vector extensions to improve programmability and performance. Evaluations on our industrial product P920 demonstrate that our processor can achieve state-of-the-art inference performance on both CNN and NLP tasks. Optimizations will be done in future work to further lift overall performance, including replacing the in-order scalar core with an out-of-order alternative [15] [22] and adjusting instruction granularity or local buffer design to lift the overlapping ratios of VME and MME units.

REFERENCES

[1] 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, and Xiaoqiang Zheng. 2016. TensorFlow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 265–283. https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf

[2] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Meghan Cowan, Haichen Shen, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. 2018. TVM: An Automated End-to-End Optimizing Compiler for Deep Learning. In *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation* (Carlsbad, CA, USA) *(OSDI'18)*. USENIX Association, USA, 579–594.

[3] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, and Olivier Temam. 2014. DaDianNao: A Machine-Learning Supercomputer. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*. 609–622. https://doi.org/10.1109/MICRO.2014.58

[4] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. 2016. Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. 367–379. https://doi.org/10.1109/ISCA.2016.40

[5] CortexA53 2012. *ARM Cortex-A53.* Retrieved April 29, 2021 from https://developer.arm.com/ip-products/processors/cortex-a/cortex-a53

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Association for Computational Linguistics, Minneapolis, Minnesota, 4171–4186. https://doi.org/10.18653/v1/N19-1423

[7] Clément Farabet, Berin Martini, Benoit Corda, Polina Akselrod, Eugenio Culurciello, and Yann LeCun. 2011. NeuFlow: A runtime reconfigurable dataflow processor for vision. In *CVPR 2011 WORKSHOPS*. 109–116. https://doi.org/10.1109/CVPRW.2011.5981829

[8] Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Todd Massengill, Ming Liu, Daniel Lo, Shlomi Alkalay, Michael Haselman, Logan Adams, Mahdi Ghandi, Stephen Heil, Prerak Patel, Adam Sapek, Gabriel Weisz, Lisa Woods, Sitaram Lanka, Steven K. Reinhardt, Adrian M. Caulfield, Eric S. Chung, and Doug Burger. 2018. A Configurable Cloud-Scale DNN Processor for Real-Time AI. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. 1–14. https://doi.org/10.1109/ISCA.2018.00012

[9] HabanaGoya [n.d.]. *Habana GOYA.* Retrieved April 29, 2021 from https://habana.ai/inference/

[10] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A. Horowitz, and William J. Dally. 2016. EIE: Efficient Inference Engine on Compressed Deep Neural Network. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. 243–254. https://doi.org/10.1109/ISCA.2016.30

[11] K. He, X. Zhang, S. Ren, and J. Sun. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, Los Alamitos, CA, USA, 770–778. https://doi.org/10.1109/CVPR.2016.90

[12] N25F [n.d.]. *AndesCore N25F.* Retrieved April 29, 2021 from http://www.andestech.com/en/products-solutions/andescore-processors/riscv-n25f/

[13] NVIDIA-T4 [n.d.]. *NVIDIA T4 Tensor Core GPU for AI Inference.* Retrieved April 29, 2021 from https://www.nvidia.com/en-us/data-center/tesla-t4/

[14] NVIDIA-V100 [n.d.]. *NVIDIA V100 Tensor Core GPU.* Retrieved April 29, 2021 from https://www.nvidia.com/en-us/data-center/v100/

[15] Karyofyllis Patsidis, Dimitris Konstantinou, Chrysostomos Nicopoulos, and Giorgos Dimitrakopoulos. 2018. A low-cost synthesizable RISC-V dual-issue processor core leveraging the compressed Instruction Set Extension. *Microprocessors and Microsystems* 61 (2018), 1–10. https://doi.org/10.1016/j.micpro.2018.05.007

[16] Abhinav Podili, Chi Zhang, and Viktor Prasanna. 2017. Fast and efficient implementation of Convolutional Neural Networks on FPGA. In *2017 IEEE 28th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. 11–18. https://doi.org/10.1109/ASAP.2017.7995253

[17] ResNetV15 [n.d.]. *ResNet-50 V1.5 Model for Tensorflow.* Retrieved April 29, 2021 from https://github.com/IntelAI/models/blob/master/benchmarks/image_recognition/tensorflow/resnet50v1_5/README.md

[18] RISC-V [n.d.]. *RISC-V ISA Specifications.* Retrieved April 29, 2021 from https://riscv.org/technical/specifications/

[19] RVV [n.d.]. *RISC-V Vector Extension Specification.* Retrieved April 29, 2021 from https://github.com/riscv/riscv-v-spec

[20] TIRIAS 2019. *Why Your AI infrastructure Needs Both Training and Inference.* Retrieved April 29, 2021 from https://www.ibm.com/downloads/cas/QM4BYOPP

[21] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. 2015. Optimizing FPGA-Based Accelerator Design for Deep Convolutional Neural Networks. In *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (Monterey, California, USA) *(FPGA '15)*. Association for Computing Machinery, New York, NY, USA, 161–170. https://doi.org/10.1145/2684746.2689060

[22] Jerry Zhao, Ben Korpan, Abraham Gonzalez, and Krste Asanovic. 2020. SonicBOOM: The 3rd Generation Berkeley Out-of-Order Machine. (May 2020).