Hardware-Assisted Virtualization of Neural Processing Units for Cloud Platforms

Yuqi Xue
yuqixue2@illinois.edu
University of Illinois Urbana-Champaign

Yiqi Liu
yiqiliu2@illinois.edu
University of Illinois Urbana-Champaign

Lifeng Nai
lnai@google.com
Google

Jian Huang
jianh@illinois.edu
University of Illinois Urbana-Champaign

Abstract

Cloud platforms today have been deploying hardware accelerators like neural processing units (NPUs) for powering machine learning (ML) inference services. To maximize the resource utilization while ensuring reasonable quality of service, a natural approach is to virtualize NPUs for efficient resource sharing for multi-tenant ML services. However, virtualizing NPUs for modern cloud platforms is not easy. This is not only due to the lack of system abstraction support for NPU hardware, but also due to the lack of architectural and ISA support for enabling fine-grained dynamic operator scheduling for virtualized NPUs.

We present Neu10, a holistic NPU virtualization framework. We investigate virtualization techniques for NPUs across the entire software and hardware stack. Neu10 consists of (1) a flexible NPU abstraction called vNPU, which enables fine-grained virtualization of the heterogeneous compute units in a physical NPU (pNPU); (2) a vNPU resource allocator that enables pay-as-you-go computing model and flexible vNPU-to-pNPU mappings for improved resource utilization and cost-effectiveness; (3) an ISA extension of modern NPU architecture for facilitating fine-grained tensor operator scheduling for multiple vNPUs. We implement Neu10 based on a production-level NPU simulator. Our experiments show that Neu10 improves the throughput of ML inference services by up to 1.4× and reduces the tail latency by up to 4.6×, while improving the NPU utilization by 1.2×, compared to state-of-the-art NPU sharing approaches.

1 Background and Motivation

Machine learning (ML) is becoming the backbone for many popular ML services [3, 6, 25, 27]. To accelerate these ML services, cloud platforms have employed hardware accelerators like neural processing units (NPUs) [7, 11, 14, 15, 17, 18]. NPUs are highly specialized to accelerate the common operations in deep neural networks (DNNs), such as matrix multiplication and convolution. A typical NPU device is a peripheral board with multiple NPU chips, and each chip has multiple NPU cores. Each NPU core has matrix engines (MEs) that leverage systolic arrays to perform matrix multiplications and vector engines (VEs) for generic vector operations. A well-known example is the Google Cloud TPU [15].

The 2nd Workshop on Hot Topics in System Infrastructure (HotInfra'24), co-located with SOSP'24, November 3, 2024, Austin, TX, USA

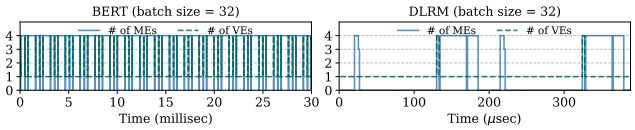

Figure 1. The number of MEs and VEs demanded by DNN inference workloads over time.

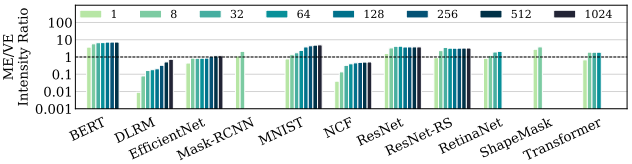

Figure 2. Intensity ratio of ME vs. VE for different inference workloads (quantified by the execution time of ME/VE).

A common approach to using NPUs in cloud platforms is to assign an entire NPU chip to a single ML application in a virtual machine (VM) or container via PCIe passthrough [27]. However, this disables resource sharing and causes severe resource underutilization of NPUs. For instance, our studies [30, 31] disclosed that many DNN inference workloads cannot fully utilize TPU cores due to their imbalanced demands on MEs and VEs. Many DNN workloads have diverse demands on the number of MEs and VEs (see Figure 1 and Figure 2). The one-size-fits-all approach is less attractive for cloud platforms.

To address the utilization challenge and ease the resource management for cloud platforms to accommodate diverse workload demands, it is desirable to virtualize hardware devices and enable resource sharing among tenants. Unfortunately, modern cloud platforms have limited virtualization support for NPUs across the software and hardware stack.

Lack of system abstraction support for NPUs. Unlike the system virtualization of multi-core processors [2, 8], NPUs have unique heterogeneous compute resources (i.e., MEs and VEs).

To circumvent this complexity, cloud platforms today expose homogeneous NPU cores to the user VMs. However, the existing abstraction at the NPU core level is too coarse-grained, as user workloads may have diverse resource requirements. We need *a flexible system abstraction that allows users to specify the ME/VE resources* following the pay-as-you-go model [28]. Such an abstraction will simplify the NPU management for cloud platforms, including NPU resource (de)allocation, resource mapping, and scheduling. Prior studies investigated the system virtualization for FPGAs [5, 22, 23, 32, 33] and GPUs [19, 29]. However, they cannot be directly applied to NPUs, as they target different architectures.

Lack of architectural support for NPU virtualization. Prior studies enabled the time-sharing of an NPU device at the task level, and support the preemption for prioritized tasks [9, 10]. However, the coarse-grained time-sharing on the shared NPU board still suffers from severe resource underutilization, due to the lack of support of concurrent execution of multi-tenant workloads. Existing NPU sharing approaches either sacrifice isolation or suffer from high preemption overhead [13]. As we move towards fine-grained NPU virtualization, we need *architectural support to achieve both improved performance isolation and NPU utilization.*

Lack of ISA support for virtualized NPUs. NPUs commonly employ VLIW-style ISAs to simplify the hardware design, and the ML compiler explicitly exploits the parallelism of the compute units [4, 20, 21]. However, this requires the number of compute units to be explicitly specified at the compilation stage, and the number cannot be changed at runtime. In this case, the VLIW ISAs unnecessarily couple control flows of the compute units (i.e., MEs). As shown in Figure 3, the original VLIW program must execute each VLIW instruction sequentially, creating false dependencies between operations on different MEs even though they do not have any true data dependencies. As the compiler explicitly specifies how many MEs are being used, the allocated MEs cannot be changed at runtime unless the DNN program is recompiled. Even though some compute units of a shared NPU become available, they cannot be utilized by the active workload (except recompiling the DNN program). This is caused by the fundamental tussle between dynamic scheduling and VLIW ISAs. As the collocated ML instances have various demands on compute units at runtime, this limitation inevitably causes either NPU underutilization or performance interference. We need to *rethink the NPU ISA design to facilitate dynamic resource scheduling for virtualized NPUs.*

2 Design and Implementation

We design Neu10 [31], a hardware-assisted system virtualization framework for NPUs to enable flexible resource sharing for improved utilization and performance isolation.

vNPU abstraction. We design the vNPU abstraction with the goals of (1) allocating NPU hardware resource to a vNPU

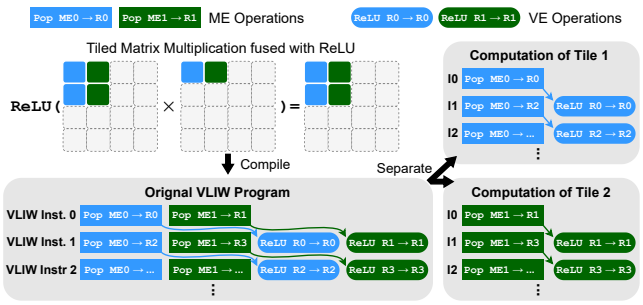

Figure 3. Execution of MEs and VEs are separable. The arrows between instructions denote data dependencies.

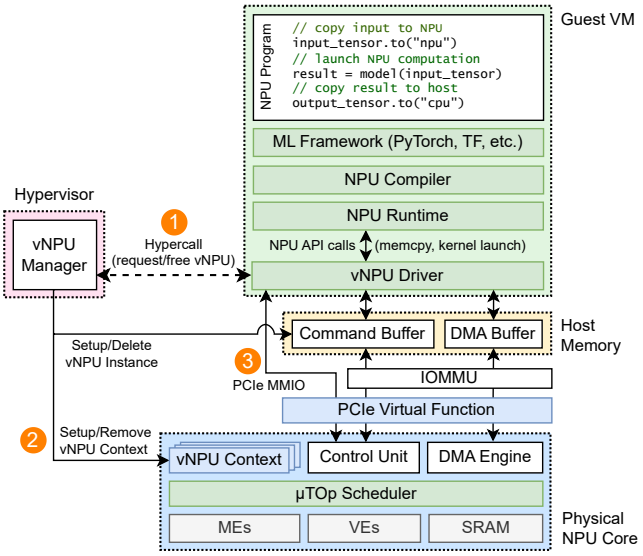

Figure 4. System architecture of Neu10.

instance on demand; (2) hiding the complexity from the ML programs with minimal changes to the guest software stack for compatibility. A vNPU instance defines the number of chips, cores per chip, MEs/VEs per core, SRAM size, and HBM size, which reflects the hierarchy of a physical NPU board. Each vNPU instance is exposed to the guest VM as a PCIe device. The maximum vNPU size is capped by the physical NPU size. If a guest VM requires more resources than is available on a physical NPU board, Neu10 can allocate multiple vNPU instances to this guest VM.

vNPU lifecycle. To create a vNPU instance, a user can specify the vNPU configuration following the pay-as-you-go model [28]. Cloud providers can define various default configurations (e.g., small/medium/large vNPU cores as having 1/4/8 MEs/VEs). Neu10 can also learn an optimized vNPU configuration for a DNN workload with ML compilers. As shown in Figure 4, upon vNPU initialization, the guest driver sends a request to the hypervisor through a para-virtualized interface (1). The vNPU manager maps the vNPU instance to NPU hardware resources. Then, it initializes the vNPU

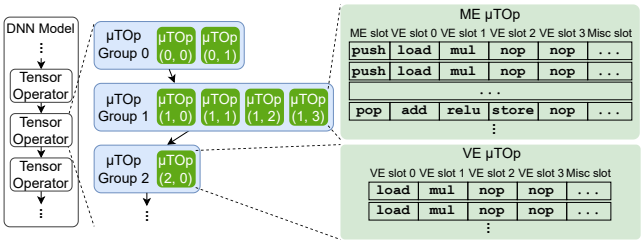

Figure 5. NeuISA programming model.

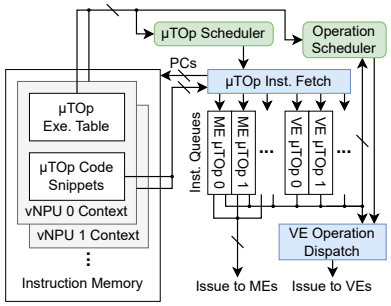

Figure 6. NPU core pipeline frontend for NeuISA.

context in the physical NPU device and creates the MMIO mappings for the guest VM to access the vNPU (2). During execution, the application issues commands such as memcpy and compute offloading through the command buffer. The NPU hardware directly fetches the commands from the host memory without the hypervisor intervention. It also has DMA access to the DMA buffer in the guest memory space via the IOMMU. The DNN program on the NPU executes asynchronously from the CPU program, and the NPU hardware schedules vNPUs independently of existing OS/hypervisor schedulers. The guest VM waits for the completion interrupt or actively polls the memory-mapped control registers for the current status of the vNPU (3). After execution, the user can free the vNPU.

vNPU allocation and mapping. For each vNPU, the user can specify the number of different types of compute units (MEs/VEs) on-demand or follow the pay-as-you-go model in cloud computing. However, as ML inference workloads have diverse ME/VE demands, specifying the number of MEs/VEs can be challenging for users who are not NPU experts. Thus, we allow them to specify the total number of execution units (EUs), which is directly related to the cost of running the vNPU instance. Neu10 uses a resource allocation mechanism that can decide the optimized vNPU configuration for different ML workloads, based on the analysis using ML compilers.

Neu10 can map vNPUs to physical compute units of NPU cores in different manners, based on the service level objectives (SLOs) of ML services. To maximize the NPU utilization while ensuring performance isolation, Neu10 enables fine-grained spatial sharing with resource harvesting. Neu10 also enables the oversubscription of NPU cores by temporally sharing the MEs and VEs among multiple vNPU instances. Therefore, the idle compute units can be opportunistically utilized by collocated workloads.

ISA extension for NPU virtualization. To support dynamic ME/VE scheduling, we develop NeuISA. Our key observation is that the execution of different MEs and VEs in a tensor operator is usually *separable*. Specifically, most DNN operators, such as matrix multiplication and convolution, are partitioned by DNN compilers [12, 34] into multiple tiles that can be computed independently, as shown in Figure 3.

NeuISA decouples the execution of independent MEs in a tensor operator by separating the control flow of each ME and VE into independent instruction sequences, called *micro-Tensor Operators (μTOps)*. To minimize changes to the existing VLIW compiler and hardware, the instruction format inside a μTOp resembles the original VLIW ISA: an instruction contains multiple slots, and each slot encodes an operation. For a physical NPU core with n_x MEs and n_y VEs, NeuISA defines two types of μTOps: (1) An *ME μTOp* contains instructions with one ME slot and n_y VE slots. An ME μTOp will only use one ME during execution, which enforces that each ME μTOp only contains the control flow of one ME. To execute an operator on multiple MEs, the compiler generates multiple ME μTOps. The VE slots in an ME μTOp enable instruction-level parallelism between MEs and VEs. They also enable operator fusions such as MatMul+ReLU. (2) A *VE μTOp* contains instructions with no ME slot and n_y VE slots. The n_y VE slots allow a VE μTOp to utilize all the VEs.

To support a fused operator, NeuISA organizes the μTOps into a sequence of *μTOp groups* to express the dependencies between μTOps. Each group contains up to n_x ME μTOps, allowing the operator to utilize all MEs, and up to one VE μTOp, as one VE μTOp can already utilize all the VEs. All μTOps in one μTOp group may execute concurrently, but each group must execute sequentially to preserve data dependency. As NeuISA inherits the VLIW semantic inside each μTOp, it intrinsically supports branches and loops inside a μTOp. To support branches across μTOp groups, NeuISA defines special instructions that can be invoked in each μTOp.

Architectural support for NeuISA. Figure 6 shows the pipeline design for fetching and scheduling μTOps. The NPU core maintains the contexts of multiple vNPUs, including the PC pointers to the program and the vNPU configurations. Each time a new μTOp is ready or an existing μTOp finishes, the *μTOp scheduler* selects the μTOps to be executed next. For each vNPU, the μTOp scheduler retrieves the number of allocated MEs and the number of ready ME μTOps from the vNPU context. It selects a set of ready μTOps, and fetch their instructions to the instruction queues.

Next, the *operation scheduler* selects which operations from the instruction queues will be executed at every cycle. The ME operations from the ME μTOp instruction queues are directly issued to the corresponding MEs. For the VE

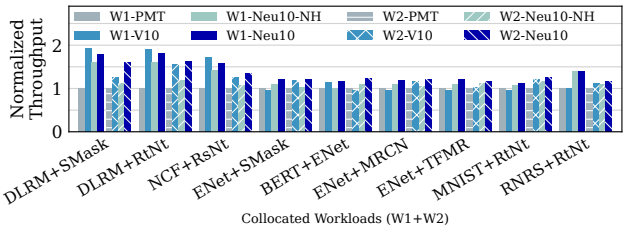

Figure 7. Throughput of Neu10 (normalized to PMT).

operations, the scheduler selects which operations to issue from all VE μTOp instruction queues. To reclaim a harvested ME, Neu10 performs a context switch to preempt the harvesting μTOp. Upon a context switch, the register file and the intermediate data in the MEs are saved to SRAM, which incurs negligible overhead compared to the length of an operator. The number of instruction queues should be large enough to support simultaneous execution of all MEs/VEs.

Neu10 Implementation. We implement Neu10 with a production-level event-driven NPU simulator. We obtain the operator execution traces on real Google Cloud TPUs. We use the tiling information to generate μTOps and replay the generated μTOp traces in our simulator. We also prototype the hardware scheduler for NeuISA in Verilog and synthesize it using FreePDK-15nm library [1]. The hardware area overhead of Neu10 is 0.04% on a TPUv4 chip. The power overhead of this small area is negligible over the entire chip.

Performance of Neu10. We evaluate Neu10 using DNN workloads from MLPerf [26] and the official TPU reference models [16]. We compare the following designs: (1) PMT [13]: temporal-sharing of the entire NPU core among multiple vNPUs. (2) V10 [30]: temporal-sharing of all MEs and VEs among the vNPUs, with a priority-based preemptive policy. The workload is compiled with the traditional VLIW-style ISA. (3) Neu10-NoHarvest: spatial-isolated vNPUs with dedicated MEs/VEs without dynamic scheduling. This resembles static partitioning techniques such as NVIDIA MIG [24]. (4) Neu10: spatial-isolated vNPUs with dynamic resource scheduling and harvesting enabled by NeuISA. As shown in Figure 7, Neu10 improves the throughput of ML inference services by up to 1.4× compared to state-of-the-art NPU sharing approaches. Neu10 also reduces the tail latency by up to 4.6×, while improving the NPU utilization by 1.2×. We presented more sensitivity analyses in the full paper[1] [31].

3 Conclusion

We identify the key challenges of virtualizing NPUs for cloud platforms. We present a holistic solution Neu10 for enabling NPU virtualization. It improves both NPU utilization and performance isolation for multi-tenant ML services.

[1]Our full paper will appear in the proceedings of the 57th IEEE/ACM International Symposium on Microarchitecture (MICRO'24).

Acknowledgements

We thank Haoyang Zhang for his insightful discussion on the NeuISA design. This work was partially supported by NSF grant CCF-1919044, NSF CAREER Award CNS-2144796, and the Hybrid Cloud and AI program at the IBM-Illinois Discovery Accelerator Institute (IIDAI).

References

[1] [n. d.]. FreePDK15. https://eda.ncsu.edu/freepdk15/

[2] Keith Adams and Ole Agesen. 2006. A Comparison of Software and Hardware Techniques for X86 Virtualization. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'06)*. San Jose, California, USA.

[3] Altexsoft. 2021. Comparing Machine Learning as a Service: Amazon, Microsoft Azure, Google Cloud AI, IBM Watson. https://www.altexsoft.com/blog/datascience/comparing-machine-learning-as-a-service-amazon-microsoft-azure-google-cloud-ai-ibm-watson/

[4] AMD. 2023. AI Engine: Meeting the Compute Demands of Next-Generation Applications. https://www.xilinx.com/products/technology/ai-engine.html

[5] AWS. 2022. Amazon EC2 F1 Instances. https://aws.amazon.com/ec2/instance-types/f1/

[6] Amazon AWS. 2022. Machine Learning on AWS Innovate faster with the most comprehensive set of AI and ML services. https://aws.amazon.com/machine-learning/

[7] Amazon AWS. 2023. AWS Inferentia. https://aws.amazon.com/machine-learning/inferentia/

[8] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. 2003. Xen and the Art of Virtualization. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles (SOSP'03)*. Bolton Landing, NY, USA.

[9] Quan Chen, Hailong Yang, Minyi Guo, Ram Srivatsa Kannan, Jason Mars, and Lingjia Tang. 2017. Prophet: Precise QoS Prediction on Non-Preemptive Accelerators to Improve Utilization in Warehouse-Scale Computers. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'17)*. Xi'an, China.

[10] Quan Chen, Hailong Yang, Jason Mars, and Lingjia Tang. 2016. Baymax: QoS Awareness and Increased Utilization for Non-Preemptive Accelerators in Warehouse Scale Computers. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'16)*. Atlanta, GA.

[11] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. 2014. DianNao: A Small-Footprint High-Throughput Accelerator for Ubiquitous Machine-Learning. In *Proceedings of the 20th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'14)*. Salt Lake City, UT.

[12] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, 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 Symposium on Operating Systems Design and Implementation (OSDI'18)*. Carlsbad, CA.

[13] Yujeong Choi and Minsoo Rhu. 2020. PREMA: A Predictive Multi-Task Scheduling Algorithm For Preemptible Neural Processing Units. In *Proceedings of the 2020 IEEE International Symposium on High Performance Computer Architecture (HPCA'20)*. San Diego, USA.

[14] Eric Chung, Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Adrian Caulfield, Todd Massengil, Ming Liu, Daniel Lo, Shlomi Alkalay, Michael Haselman, Christian Boehn, Oren Firestein, Alessandro Forin, Kang Su Gatlin, Mahdi Ghandi, Stephen Heil, Kyle Holohan, Tamas Juhasz, Ratna Kumar Kovvuri, Sitaram Lanka, Friedel van Megen, Dima Mukhortov, Prerak Patel, Steve Reinhardt, Adam Sapek, Raja Seera, Balaji Sridharan, Lisa Woods, Phillip Yi-Xiao, Ritchie Zhao, and Doug Burger. 2017. Accelerating Persistent Neural Networks at Datacenter Scale. In *Proceedings of HotChips'17*. Cupertino, CA.

[15] Google. 2022. System Architecture - Cloud TPU. https://cloud.google.com/tpu/docs/system-architecture-tpu-vm

[16] Google. 2023. Supported reference models. https://cloud.google.com/tpu/docs/tutorials/supported-models

[17] Graphcore. 2022. Graphcore IPU Overview. https://www.graphcore.ai/products/ipu

[18] Linley Gwennap. 2020. Tenstorrent Scales AI Performance: New Multicore Architecture Leads in Data-Center Power Efficiency. https://www.linleygroup.com/mpr/article.php?id=12287

[19] Mingcong Han, Hanze Zhang, Rong Chen, and Haibo Chen. 2022. Microsecond-scale Preemption for Concurrent GPU-accelerated DNN Inferences. In *Proccedings of the 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI'22)*. Carlsbad, CA.

[20] Jonathan Hui. 2020. AI Chips Technology Trends and Landscapes (Mobile SoC, Intel, Asian AI Chips, Low-Power Inference Chips). https://jonathan-hui.medium.com/ai-chips-technology-trends-landscape-mobile-soc-intel-asian-ai-chips-low-power-inference-4db701dbe85d

[21] 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, and Doe Hyun Yoon. 2017. In-Datacenter Performance Analysis of a Tensor Processing Unit. In *Proceedings of the 44th International Symposium on Computer Architecture (ISCA'17)*. Toronto, Canada.

[22] Ahmed Khawaja, Joshua Landgraf, Rohith Prakash, Michael Wei, Eric Schkufza, and Christopher J. Rossbach. 2018. Sharing, Protection, and Compatibility for Reconfigurable Fabric with AmorphOS. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. Carlsbad, CA.

[23] Joshua Landgraf, Tiffany Yang, Will Lin, Christopher J. Rossbach, and Eric Schkufza. 2021. Compiler-Driven FPGA Virtualization with SYNERGY. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (Virtual, USA). https://doi.org/10.1145/3445814.3446755

[24] Nvidia. 2022. Multi-Instance GPU User Guide. https://docs.nvidia.com/datacenter/tesla/mig-user-guide/

[25] Ejiro Onose. 2022. Machine Learning as a Service: What It Is, When to Use It and What Are the Best Tools Out There. https://neptune.ai/blog/machine-learning-as-a-service-what-it-is-when-to-use-it-and-what-are-the-best-tools-out-there

[26] Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maximilien Breughe, Mark Charlebois, William Chou, Ramesh Chukka, Cody Coleman, Sam Davis, Pan Deng, Greg Diamos, Jared Duke, Dave Fick, J. Scott Gardner, Itay Hubara, Sachin Idgunji, Thomas B. Jablin, Jeff Jiao, Tom St. John, Pankaj Kanwar, David Lee, Jeffery Liao, Anton Lokhmotov, Francisco Massa, Peng Meng, Paulius Micikevicius, Colin Osborne, Gennady Pekhimenko, Arun Tejusve Raghunath Rajan, Dilip Sequeira, Ashish Sirasao, Fei Sun, Hanlin Tang, Michael Thomson, Frank Wei, Ephrem Wu, Lingjie Xu, Koichi Yamada, Bing Yu, George Yuan, Aaron Zhong, Peizhao Zhang, and Yuchen Zhou. 2020. MLPerf Inference Benchmark. arXiv:1911.02549

[27] RUN:AI. 2022. Google TPU Architecture and Performance Best Practices. https://www.run.ai/guides/cloud-deep-learning/google-tpu

[28] Stephen J. Bigelow. 2022. pay-as-you-go cloud computing (PAYG cloud computing), https://www.techtarget.com/searchstorage/definition/pay-as-you-go-cloud-computing-PAYG-cloud-computing.

[29] Kyle Wiggers. 2022. Microsoft and NVIDIA team up to build new Azure-hosted AI supercomputer. https://techcrunch.com/2022/11/16/microsoft-and-nvidia-team-up-to-build-new-azure-hosted-ai-supercomputer/

[30] Yuqi Xue, Yiqi Liu, Lifeng Nai, and Jian Huang. 2023. V10: Hardware-Assisted NPU Multi-Tenancy for Improved Resource Utilization and Fairness. In *Proceedings of the 50th Annual International Symposium on Computer Architecture (ISCA'23)*. Orlando, FL, USA.

[31] Yuqi Xue, Yiqi Liu, Lifeng Nai, and Jian Huang. 2024. Hardware-Assisted Virtualization of Neural Processing Units for Cloud Platforms. https://arxiv.org/abs/2408.04104

[32] Yue Zha and Jing Li. 2020. *Virtualizing FPGAs in the Cloud*. Association for Computing Machinery, New York, NY, USA, 845–858. https://doi.org/10.1145/3373376.3378491

[33] Yue Zha and Jing Li. 2021. When Application-Specific ISA Meets FPGAs: A Multi-Layer Virtualization Framework for Heterogeneous Cloud FPGAs. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (Virtual, USA). https://doi.org/10.1145/3445814.3446699

[34] Hongyu Zhu, Ruofan Wu, Yijia Diao, Shanbin Ke, Haoyu Li, Chen Zhang, Jilong Xue, Lingxiao Ma, Yuqing Xia, Wei Cui, Fan Yang, Mao Yang, Lidong Zhou, Asaf Cidon, and Gennady Pekhimenko. 2022. ROLLER: Fast and Efficient Tensor Compilation for Deep Learning. In *Proceedings of 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. Carlsbad, CA.