Hardware-Assisted Virtualization for Neural Processing Units

Yuqi Xue
University of Illinois Urbana-Champaign

Yiqi Liu
University of Illinois Urbana-Champaign

Lifeng Nai
Google

Jian Huang
University of Illinois Urbana-Champaign

ABSTRACT

Modern cloud platforms have deployed neural processing units (NPUs) to meet the increasing demand for machine learning (ML) services. However, the current way of using NPUs in cloud platforms suffers from either low resource utilization or poor isolation between multi-tenant application services due to the lack of system virtualization support for fine-grained resource sharing.

In this paper, we investigate the system virtualization techniques for NPUs across the entire hardware and software stack. In the hardware stack, we design a hardware-assisted multi-tenant NPU for fine-grained resource sharing and isolation. It employs an operator scheduler on the NPU core to enable concurrent operator executions and flexible priority-based resource scheduling. In the software stack, we propose a flexible vNPU abstraction. We leverage this abstraction to design the vNPU allocation, mapping, and scheduling policies to maximize resource utilization while guaranteeing both performance and security isolation for vNPU instances at runtime.

1 BACKGROUND AND MOTIVATION

Machine learning (ML) workloads have seen a significant rise in modern cloud data centers [6, 7, 16, 18, 19]. They have become the foundation of many popular applications. To improve the performance of these AI applications, cloud platforms have employed neural processing units (NPUs) for deep neural networks (DNNs) [3, 10, 11, 13–15].

A typical NPU design aims to accelerate common operations in DNN models, such as matrix multiplications and convolutions. An NPU device is a peripheral board with multiple NPU chips, and each chip contains multiple NPU cores. Each NPU core usually includes systolic arrays (SAs) that exploit data reuse patterns of matrix multiplication and vector units (VUs) for generic vector operations like activations and reductions. As NPUs are nowadays the most efficient accelerators for DNN computations, they are becoming the most popular accelerators for ML workloads in the cloud [1–4, 13].

The de facto standard in the cloud to offer users NPUs is to exclusively assign one NPU device to one user VM via PCIe pass-through, preventing other users from sharing the same NPU. This inevitably leads to underutilized hardware if the single ML workload cannot fully utilize the NPU. To better utilize NPUs, modern cloud platforms implement *limited* virtualization supports for NPUs. They enable the time-sharing of an NPU device at the task level and the task preemption to allocate the NPU to prioritized users [8, 9]. However, this coarse-grained time-multiplexing on a single NPU board still suffers significant resource underutilization, because it does not support the concurrent execution of multi-tenant DNN workloads or the fine-grained resource allocation on NPU cores. They cannot leverage multi-tenant workloads to improve utilization, and none of the sharing mechanisms provide sufficient security or performance isolation in a multi-tenant cloud environment.

To understand NPU utilization in the cloud, we thoroughly investigate a real Google Cloud TPUv2 and test various DNN workloads [5, 17]. Inside the TPU core, we profile the resource utilization of its main components: the SA and the VU. We find that most DNN inference workloads significantly underutilize the compute resources on the TPU core. The reason is that many DNN inference workloads have *imbalanced demands on SAs and VUs*, as shown in Figure 1 — they are either SA-intensive or VU-intensive. As a result, SA-intensive workloads inevitably underutilize VUs and vice versa.

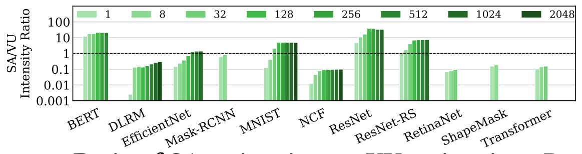


Figure 1: Ratio of SA active time vs VU active time. Deeper colors represent larger batch sizes. Some workloads with large batch sizes fail due to insufficient memory.

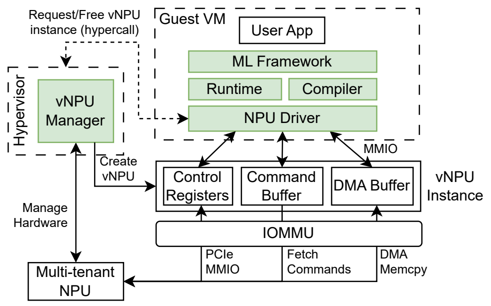


Figure 2: System overview for NPU virtualization.

2 TOWARDS NPU VIRTUALIZATION

We propose a full-stack NPU virtualization solution with hardware-software co-design as shown in Figure 2. First, we design a hardware-assisted multi-tenant NPU with fine-grained resource management for better resource utilization and performance isolation (§2.1). With the multi-tenant NPU, we propose a flexible vNPU abstraction and the system support for efficient NPU virtualization (§2.2).

2.1 Hardware-Assisted NPU Multi-tenancy

We develop V10, a hardware-assisted multi-tenant NPU [21]. It enables fine-grained concurrent execution of ML workloads with an operator scheduler in the NPU, which exploits the idle cycles caused by the imbalanced use of SAs and VUs in an ML kernel and enables concurrent execution of operators from different ML workloads.

Tensor Operator Scheduler. Figure 3 shows the architecture of the operator scheduler. It is located at the front end of the NPU pipeline, between the instruction memory and the functional units (FUs) such as SAs and VUs, to control the instruction fetch and issue logic. It selects independent operators from different DNN workloads and dispatches them to multiple FUs. The scheduler minimizes hardware modifications by leveraging the existing hardware capability to dispatch SA and VU operations in parallel.

The scheduler maintains a *workload context table* to track the execution states of each workload and their operators. The *priority-based scheduling policy* uses the context table to decide which operator will be executed next. Periodically, a *preemption timer* will trigger the scheduler to examine whether an operator should be preempted. If so, the *preemption module* generates instructions to save the context for the preempted operator. Once an operator starts execution, it will

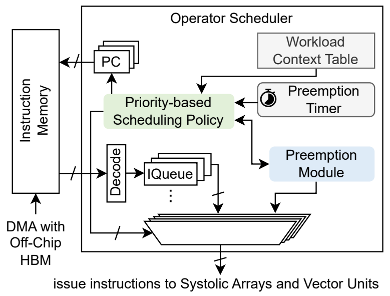


Figure 3: Architecture of the tensor operator scheduler.

occupy an FU until it finishes or is preempted. Then, the scheduling policy will be invoked to assign the next operator to the free FU.

Priority-based Operator Scheduling. V10's hardware scheduler offers the flexibility for enforcing different priorities to satisfy different service-level agreements (SLAs) for ML services. The scheduler guarantees that a workload should spend computation cycles proportional to its relative priority [12]. The scheduler will prioritize the workload that currently suffers from the lowest share of resources with respect to its priority. With this scheduling policy, V10 dynamically controls the resource allocation to each workload.

Tensor Operator Preemption. Since the sizes of tensor operators vary, different operators require different amounts of hardware resources, which causes unfairness and starvation. For example, large operators will block small operators of the collocated ML workload. This not only causes unfairness but also limits potential opportunities for overlapping the execution of SA and VU.

We propose a lightweight operator preemption mechanism for NPUs. Instead of re-executing the entire operator after a context switch, we enable low-overhead recomputation for the SA by asynchronously checkpointing input data and overlapping the switching of two operators. At the cycle when preemption is invoked, we keep the SA running while saving all further inputs into the on-chip SRAM. No cycles are wasted so far as the SA is still popping valid outputs. After all partial sums depending on earlier inputs are popped, we pause the execution. Then, we save the data of the preempted operator and simultaneously start restoring the data of the next operator into the SA. With our approach, the context switch overhead is negligible compared to the average operator length. The storage overhead is also trivial compared to the on-chip SRAM capacity.

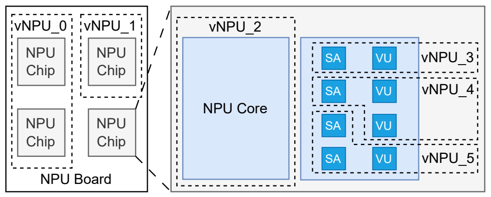
2.2 System Support for NPU Virtualization

Based on the hardware-assisted multi-tenant NPU, we propose NeuCloud, a full-stack NPU virtualization solution with fine-grained hardware resource management with the vNPU abstraction [20].

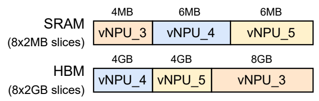
vNPU Abstraction. For optimal resource efficiency, we must allocate different numbers of SAs and VUs to a DNN workload based on its demands. The vNPU abstraction must provide flexibility for the user to customize a vNPU. To minimize changes to guest software stacks, a vNPU has the same structure as a physical NPU board.

As Figure 4(a) shows, a vNPU can encapsulate different resource configurations, and a physical NPU board can host multiple vNPU instances with different configurations. A vNPU can be collocated with other vNPUs on the same NPU core if the total number of SAs and VUs does not exceed hardware limitations. If the DNN workload requires more resources than is available on one core, vNPUs occupying an entire chip or more can be created. The maximum size of one vNPU is capped by the real physical hardware. If a guest VM requires more resources than is available on a physical NPU board, NeuCloud can allocate multiple vNPUs to it.

A vNPU can also customize its in-core SRAM size and HBM size. As shown in Figure 4(b), the SRAM is evenly divided into 8 slices and



(a) Abstraction of NPU compute resources.



(b) Abstraction of memory resources of a single NPU core.

Figure 4: vNPU abstraction.

allocated at 2MB granularity. By default, a 2MB slice will be allocated to each of the vNPU's EU, since the SRAM is used as a buffer to hide HBM latencies and should be large enough to match the compute throughput of the EUs. The off-chip HBM is also allocated in slices.

A vNPU instance reflects the hierarchy of a physical NPU board to minimize changes to existing compiler/driver stacks. Each vNPU is exposed to the guest VM as a PCIe device that resembles an NPU board. The guest NPU driver can query the hierarchy of the emulated vNPU, such as the number of chips, cores per chip, etc. The guest ML framework can handle data distribution according to the vNPU configuration, the same as how it handles a bare-metal NPU board.

vNPU Lifecycle. Before creating the vNPU instance, **1)** the user specifies how many NPU cores it needs and the core size. The cloud service provider can also define some presets of vNPU configurations. **2)** Upon vNPU initialization, the guest driver sends a request to the hypervisor through a para-virtualized interface. **3)** The vNPU manager finds a piece of NPU hardware to allocate the vNPU instance and creates the MMIO mappings for the guest VM to access the vNPU. **4)** During execution, the user application issues memcpy and compute offload commands through the command buffer. The NPU hardware directly fetches and executes the commands from the host memory without hypervisor intervention and accesses the guest memory space via the IOMMU. The guest VM can wait for the command completion interrupt or actively poll the control registers for the current status of the vNPU instance.

vNPU Mapping. NeuCloud leverages V10's priority-based scheduling policy to map multiple vNPUs to the same physical NPU core at fine granularity. The user first chooses a vNPU configuration that satisfies the SLOs. Then, the priority of this vNPU is set by the expected resource consumption of the vNPU on a physical NPU core, based on the temporal utilization of each FU. For example, if the vNPU has 4 FUs and the average FU utilization is 50%, this vNPU is expected to consume 200%/800% = 25% of resources on a physical NPU core with 8 FUs. NeuCloud leverages the SA/VU intensity of the targeted DNN workload to estimate the FU utilization of the vNPU. At runtime, V10's operator scheduler treats all SAs/VUs as a single large SA/VU and time-multiplexes multiple operators on all FUs. The priority setting described above provides a convenient way to map vNPU instances to V10's NPU cores.

3 CONCLUSION AND FUTURE WORK

We identify resource underutilization of NPUs in the cloud and propose NPU virtualization. We identify key challenges of virtualizing NPUs, such as the need for fine-grained resource management and isolation between vNPUs. We propose a hardware-software co-design for efficient NPU virtualization, which improves both resource utilization and performance isolation in a multi-tenant cloud. As future work, we plan to develop a full system prototype of NeuCloud and integrate it into the open-source container platforms.

REFERENCES

[1] 2019. Alibaba Unveils AI Chip to Enhance Cloud Computing Power. https://www.alibabacloud.com/blog/alibaba-unveils-ai-chip-to-enhance-cloud-computing-power_595409

[2] 2019. HUAWEI CLOUD Enables More Intelligence with Its AI Chips. https://www.huaweicloud.com/intl/en-us/cloudplus/thirdphase/detail_12.html

[3] 2023. AWS Inferentia. https://aws.amazon.com/machine-learning/inferentia/

[4] 2023. Graphcloud: Cloud-based Machine Intelligence. https://www.graphcore.ai/graphcloud

[5] 2023. Supported reference models. https://cloud.google.com/tpu/docs/tutorials/supported-models

[6] 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/

[7] 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/

[8] 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.

[9] 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, Georgia, USA.

[10] 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.

[11] 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.

[12] Stijn Eyerman and Lieven Eeckhout. 2008. System-Level Performance Metrics for Multiprogram Workloads. *IEEE Micro* 28, 3 (2008), 42–53. https://doi.org/10.1109/MM.2008.44

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

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

[15] 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

[16] 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

[17] 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. 2019. MLPerf Inference Benchmarks. *arXiv* 1911.02549 (2019).

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

[19] Alexander Spiridonov. 2021. New Cloud TPU VMs make training your ML models on TPUs easier than ever. https://cloud.google.com/blog/products/compute/introducing-cloud-tpu-vms

[20] Yuqi Xue, Yiqi Liu, and Jian Huang. 2023. System Virtualization for Neural Processing Units. In *The 19th Workshop on Hot Topics in Operating Systems (HotOS '23)*. https://doi.org/10.1145/3593856.3595912

[21] Yuqi Xue, Yiqi Liu, Lifeng Nai, and Jian Huang. 2023. V10: Hardware-Assisted NPU Multi-tenancy for Improved Resource Utilization and Fairness. In *ISCA-50: 50th International Symposium on Computer Architecture (ISCA '23)*. https://doi.org/10.1145/3579371.3589089