System Virtualization for Neural Processing Units

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

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

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

ABSTRACT

Modern cloud platforms have been employing hardware accelerators such as neural processing units (NPUs) to meet the increasing demand for computing resources for AI-based application services. However, due to the lack of system virtualization support, the current way of using NPUs in cloud platforms suffers from either low resource utilization or poor isolation between multi-tenant application services. In this paper, we investigate the system virtualization techniques for NPUs across the entire software and hardware stack, and present our NPU virtualization solution named NeuCloud. We propose a flexible NPU abstraction named vNPU that allows fine-grained NPU virtualization and resource management. We leverage this abstraction and design the vNPU allocation, mapping, and scheduling policies to maximize the resource utilization, while achieving both performance and security isolation for vNPU instances at runtime.

CCS CONCEPTS

• **Computer systems organization** → **Systolic arrays**; **Neural networks**; • **Software and its engineering** → **Virtual machines**; *Operating systems.*

KEYWORDS

Neural Processing Unit, Accelerator Virtualization, Hardware Accelerator, Cloud Computing

ACM Reference Format:
Yuqi Xue, Yiqi Liu, and Jian Huang. 2023. System Virtualization for Neural Processing Units. In *Workshop on Hot Topics in Operating Systems (HotOS '23), June 22–24, 2023, Providence, RI, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3593856.3595912

HotOS '23, June 22–24, 2023, Providence, RI, USA

ACM ISBN 979-8-4007-0195-5/23/06…$15.00
https://doi.org/10.1145/3593856.3595912

1 INTRODUCTION

Machine learning (ML) workloads have been widely deployed in modern data centers [1, 3, 25, 29, 30]. They have become the foundation of many popular applications, including online recommendations, video analysis, language translations, and AI assistants. To improve the performance of these AI-based applications, cloud platforms have employed hardware accelerators like neural processing units (NPUs) for deep neural networks (DNNs) [4, 7, 9, 12, 14, 16].

A typical NPU design like Google Cloud TPU [12] aims to accelerate common operations in DNN models, such as matrix multiplication and convolution. 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 the most efficient accelerators for DNN computations, they are becoming the most popular accelerators for ML workloads in the cloud platforms [4, 10, 12, 15, 33].

To use NPUs on cloud platforms, the most common way is to assign *an entire NPU board* exclusively to a single virtual machine (VM) or container via PCIe pass-through techniques [29]. However, it completely disallows resource sharing and causes severe resource underutilization. For instance, our prior study [34] of running a variety of DNN workloads from MLPerf AI Benchmarks [28] and the official TPU reference models [13] disclosed that a majority of these DNN inference workloads significantly underutilize the compute resources on the TPU core. This is because many of them have imbalanced demands on SAs and VUs. They are either SA-intensive or VU-intensive, as a result, SA-intensive workloads will underutilize VUs, and VU-intensive workloads will underutilize SAs in a TPU core.

To improve the compute utilization of NPUs, modern cloud platforms implement *limited* virtualization supports for NPUs. They enable the time-sharing of an NPU device at task level, and support the task preemption for prioritized users [5, 6]. However, this coarse-grained time-multiplexing on a single NPU board still suffers from significant resource underutilization, because it does not support concurrent execution of multi-tenant DNN workloads, and the fine-grained resource allocation on NPU cores. Therefore, they cannot

leverage multiple DNN workloads to improve the NPU utilization. Furthermore, none of the sharing mechanisms provided sufficient security and performance isolation in a multi-tenant cloud environment. They either sacrifice isolation for fine-grained time-multiplexing or suffer from high preemption overhead (e.g., swapping out the entire 128GB NPU memory is expensive when switching workloads).

To this end, we propose NeuCloud, a system virtualization solution for NPUs using a hardware-software co-design approach. In NeuCloud, we present the necessary architectural supports for enabling NPU virtualization for achieving improved end-to-end performance for DNN inference workloads (§3.1). We propose an abstraction named vNPU to support fine-grained SA/VU allocation (§3.2). We also propose an algorithm to identify an optimized vNPU configuration for workloads with different SA and VU demands (§3.3). To maximize the NPU utilization at scale, we enable fine-grained spatial sharing by designing an efficient mapping policy that can dynamically assign different vNPU instances to multiple physical NPU cores (§3.4). When a vNPU instance is idle, we enable NPU core oversubscription by temporally sharing the SAs and VUs among multiple vNPUs, therefore, the idle compute units can be used by other workloads (§3.5). NeuCloud also provides security and performance isolation of vNPUs via the partitioning of compute units and memory address space (§3.6). In order to facilitate the NeuCloud deployment in practice, we also discuss the possible ways of integrating NeuCloud into state-of-the-art cloud infrastructures (§3.7).

2 BACKGROUND AND MOTIVATION

In this section, we first introduce the generic NPU system architecture. After that, we present our study on the resource utilization of NPUs in the cloud, which serves as the motivation of NeuCloud design.

2.1 NPU System Architecture

Without loss of generality, we use an NPU system derived from the state-of-the-art NPUs in production, such as Google Cloud TPUs [19]. An NPU core consists of a command processor that fetches commands from the host memory, a direct memory access (DMA) engine for copying data between host and NPU, and a compute engine that includes an on-chip scratchpad memory (SRAM), systolic arrays (SAs), and vector units (VUs). The on-chip SRAM exchanges data with off-chip high-bandwidth memory (HBM) via DMA operations executing in parallel with the computations. The VU uses multiple SIMD units to perform vector operations. The SA uses a set of processing elements (e.g., 128×128 PEs) to exploit data reuse and parallelism in matrix multiplication and convolution operations.

Table 1: DNN models used in our experiments.

DNN Model Names	Category
BERT, Transformer	Natural Language Processing
DLRM, NCF	Recommendation
Mask-RCNN, RetinaNet, ShapeMask	Object Detection & Segmentation
MNIST, ResNet, ResNet-RS, EfficientNet	Image Classification

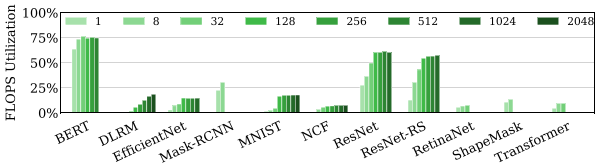


Figure 1: Overall FLOPS utilization for DNN inference workloads (deeper color represents a larger batch size). Note that some workloads with large batch sizes fail due to insufficient memory.

To execute a DNN model on the NPU, the model is first compiled by ML frameworks into a stream of tensor operators [8, 26, 31] representing the DNN execution graph. The operators are then translated by vendor-specific libraries and compilers into machine instructions for the NPU core.

2.2 NPU Resource Underutilization

The de facto standard for the cloud to offer users NPUs is exclusively assigning one NPU device to one VM or container via PCIe pass-through, preventing other users to share the same NPU. This inevitably leads to underutilized hardware, if the single ML workload cannot fully utilize the NPU. To investigate the NPU utilization on the cloud, we run various DNN inference tasks from MLPerf benchmarks [28] and official TPU reference models [13] (see Table 1) on a real Google TPUv2 board with 8 cores. Each core has one SA and one VU. We profile the resource utilization with performance counters on the TPU. We vary the inference batch size to demonstrate the impact of different computational intensities on resource utilization. We report the utilization of the core components in a TPU core: the matrix multiplication unit (i.e., SA), the vector processing unit (i.e., VU), and the HBM memory. We present the profiling results of one representative TPU core, as all the cores perform identical computations with data parallelism.

Low NPU utilization for a single ML workload. As shown in Figure 1, we present the compute resource utilization (measured in FLoating Point OPerations per Second) when running one ML workload on a TPU. Most ML workloads utilize less than half of the total FLOPS of a TPU core. Increasing batch sizes has limited impact on the compute utilization. For example, the most compute-intensive model BERT still wastes 25% of the maximum FLOPS.

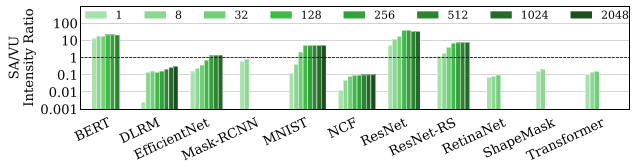


Figure 2: Ratio of SA active time vs. VU active time.

The major reason for the low NPU utilization is that the SA, which provides the most FLOPS on TPUs, is temporally underutilized. As many DNN operators, such as pooling and activation, can only be executed on the VU, SA usually becomes idle when there are no matrix multiplications (MM) or convolutions. We show the ratio between SA and VU computation time in Figure 2. For many DNN workloads, the time spent on the VU is 10× more than that on the SA, causing significant SA idleness. In contrast, for some DNN workloads, the time spent on the SA is 10× more than that on the VU. These patterns are determined by the DNN model structure. For example, BERT and ResNet models are MM or convolution-intensive, they involve more SA operators, while DLRM and ShapeMask models are bottlenecked by vector operations that execute on VUs.

The FLOPS utilization is not further increased as the batch size increases beyond 128. Although larger batch sizes can reduce SA padding overhead, it also increases the burden on the VU. Therefore, it will take a longer time to process the larger batch. As a result, the utilization of SA and VU remains imbalanced regardless of larger batch sizes.

Imbalanced demands on SAs and VUs. To scale up the performance of an ML workload, a common way is to increase the number of SAs and VUs in an NPU core. For example, Google TPUv3 doubles the number of SAs, compared to TPUv2. However, not all ML workloads are able to exploit the full benefit of having more SAs, which inevitably leads to a waste of resources. This is an expected outcome based on the analysis with Amdahl's Law: since increasing the number of SAs only accelerates the SA portion of a workload, if the workload is VU-intensive, adding more SAs will have limited improvement and waste even more hardware resources. Similarly, an SA-intensive workload will not benefit significantly from having more VUs, which will be mostly idle anyway. Therefore, the numbers of SAs and VUs assigned to a workload should be flexible according to its demands. Adding more SAs or VUs without considering workload patterns will cause suboptimal resource utilization.

3 TOWARDS NPU VIRTUALIZATION

We present the system overview of NeuCloud in Figure 3. In this section, we first discuss the hardware support for NPU virtualization. And then, we introduce the major components of NeuCloud respectively.

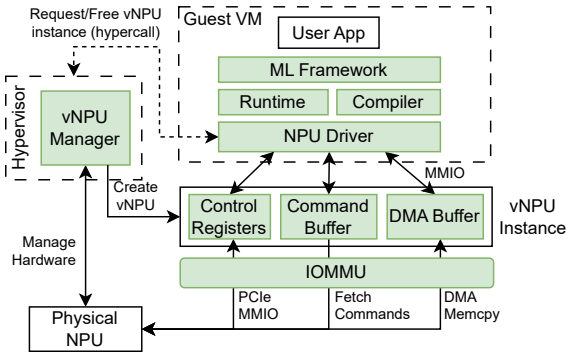


Figure 3: System overview of NPU virtualization.

3.1 Architectural Support for Virtualization

The most essential hardware support for virtualizing NPU cores is the ability to partition the core at the execution unit (EU) granularity. This allows the EUs, including SAs and VUs, to execute independently in different vNPU instances. The hypervisor partitions a physical NPU core, and allows different EUs to execute instructions from different vNPUs. To facilitate this execution mode, the EUs must follow a multi-instruction multi-data (MIMD) computing paradigm. This feature is the foundation for fine-grained resource allocation and isolation between vNPUs.

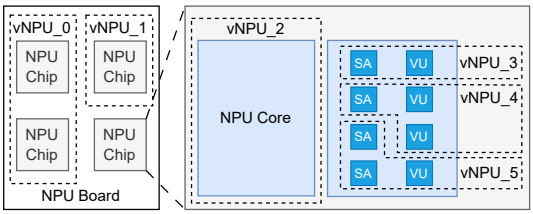
In fact, MIMD across EUs is already supported by many NPU designs. For example, Graphcore IPU has 1472 EUs on each chip, each EU has its own instruction space and execution pipeline [14]. Tenstorrent GraySkull has one vector engine (VU) and one matrix engine (SA) in each of its 120 Tensix Cores [16], both engines can execute independently.

A handful of other hardware features are not necessary for virtualization, but they enable a more efficient implementation compared to the software-based approaches. For example, the hardware-based memory address translation enables memory isolation for vNPUs with near-zero overhead [17]. Many existing hardware designs physically partition their entire on-chip memory into multiple slices, each of which serves one or two EU(s) [14, 16]. Alternatively, without hardware support, memory isolation is still possible via hypervisor with a higher overhead (see §3.6).

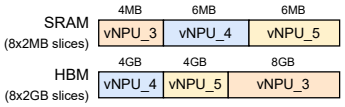
3.2 vNPU Abstraction

For improved resource efficiency, we can allocate different numbers of SAs and VUs to a DNN workload based on its demands. Thus, the vNPU abstraction must provide the flexibility for a workload to customize its vNPU. Also, a vNPU instance maintains the same structure as a physical NPU board to minimize the changes to the guest software stack.

Fined-grained resource abstraction with vNPU. As shown in Figure 4(a), a vNPU can encapsulate different compute



(a) Abstraction of NPU compute resources.



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

Figure 4: vNPU abstraction.

configurations, and a physical NPU board can host multiple vNPU instances with different configurations. The smallest vNPU (`vNPU_3`) contains one SA and one VU, and it can be collocated with other vNPUs (`vNPU_4` and `vNPU_5`) on the same NPU core, if the total number of SAs and VUs does not exceed the hardware limitation. If the DNN workload requires more resources than available on one NPU core, vNPUs occupying an entire chip with two cores (`vNPU_1`) or more chips (`vNPU_0`) can be created.

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 allocated at 2MB granularity. By default, a 2MB slice will be allocated to each compute unit of the vNPU, since the SRAM is used as a buffer to hide the HBM access latency, and should be large enough to match the compute throughput of EUs. In addition, the off-chip HBM is also allocated at the slice granularity.

vNPU hierarchy. A vNPU instance reflects the hierarchy of a physical NPU board to minimize the changes to existing compiler/driver stacks. Listing 1 shows all customizable parameters of a vNPU. Each vNPU is exposed to the VM as a PCIe device that resembles a small NPU board. The guest NPU driver can query the hierarchy of the emulated vNPU, such as the number of chips, cores per chip, HBM size, and others. The guest ML framework can handle the data distribution across multiple vNPU cores in the same way as that on physical NPUs. As the TPU compiler already supports TPUv2 (1 SA, 1 VU) and TPUv3 (2 SAs, 1 VU), and the TensorFlow framework can handle data parallelism across physical NPU devices, a vNPU instance with different configurations can smoothly work with the guest ML framework.

The maximum resource that can be allocated to one vNPU is capped by the actual physical hardware. If a guest VM requires more resources than is available on a physical NPU board, NeuCloud can allocate multiple vNPU instances to it.

vNPU lifecycle. Before creating a vNPU instance, **(1)** a user specifies the number of NPU cores it needs as well as the

Listing 1: vNPU parameters.

```
struct vNPU_Board {
    size_t num_chips;
    size_t num_cores_per_chip;
    size_t num_SAs_per_core
    size_t num_VUs_per_core;
    size_t sram_size_per_core;
    size_t mem_size_per_core;
}
```

core size. The cloud service provider can define, for example, small, medium, and large NPU cores as having a total of 1, 4, and 8 SAs/VUs for simplicity. **(2)** Optionally, the user can enable the provided compiler and profiling toolchain to learn an optimized combination of SAs and VUs for any specific DNN workload (§3.3). **(3)** Upon vNPU initialization, the guest driver sends a request to the hypervisor through a para-virtualized interface. **(4)** The vNPU manager identifies the available NPU hardware resources to allocate the vNPU instance (§3.7), and creates the MMIO mappings for the guest VM to access the vNPU. **(5)** During execution, the user application issues memcpy and compute offloading commands through the command buffer. The NPU hardware directly fetches and executes the commands from the host memory without the hypervisor intervention. It also has DMA access to 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 (Figure 3). **(6)** The user can detach and free the vNPU instance or request a different vNPU configuration.

3.3 vNPU Allocation

To optimize NPU utilization while guaranteeing service level objectives (SLOs), the allocator should assign a proper combination of SAs, VUs, and SRAM/HBM to a vNPU instance.

As discussed in §2.2, DNN workloads have imbalanced SA/VU demands. To improve the NPU utilization, NeuCloud decides the ratio between the numbers of SAs and VUs as follows. The SA/VU demands of a DNN workload can be reflected by how it runs on 1 SA and 1 VU, and we denote its active runtime on the SA as $\boldsymbol{x}$, and that on the VU as $\boldsymbol{y}$. These numbers can be obtained via profiling at the compilation stage. In reality, tensor operators have to execute sequentially due to data dependency, so the SA/VU will be idle while waiting for the execution of previous VU/SA operators. Thus, the total execution time of the workload on one SA and one VU is $x + y$. With Amdahl's Law, the new execution time on $\boldsymbol{n_x}$ SAs and $\boldsymbol{n_y}$ VUs will be $\frac{x}{n_x} + \frac{y}{n_y}$. Therefore, the expected speedup is

$$S = (x + y) / \left(\frac{x}{n_x} + \frac{y}{n_y} \right). \quad (1)$$

Let $S_h = n_x + n_y$ be the hypothetical speedup regardless of EU types, which means an EU can execute both SA and

VU operators. Compared to real cases where each EU must respect data dependencies and operator types, the hypothetical speedup assumes all $n_x + n_y$ EUs are always busy and 100% utilized. The utilization of running an ML workload on n_x SAs and n_y VUs can be quantified as the ratio between the expected and hypothetical speedups:

$$\boldsymbol{U} = \frac{S}{S_h} = \frac{(x+y)n_x n_y}{(xn_y + yn_x)(n_x + n_y)}. \quad (2)$$

To isolate the impact of total SA and VU quantity, we simplify the function by letting $\boldsymbol{a} = x/y$ be the SA/VU intensity ratio of the given workload, and $\boldsymbol{k} = n_x/n_y$ be the ratio between the numbers of SAs and VUs. Then, we can simplify Equation (2) with WolframAlpha [21]:

$$U = \frac{k(1+a)}{(k+a)(k+1)}, k > 0, a > 0. \quad (3)$$

To find the value of k that maximizes U, we compute the value of k where $\frac{\mathrm{d}U}{\mathrm{d}k} = 0$. Thus, when $k = \sqrt{a}$, we have the maximum value of U. In this case, for workloads with SA/VU intensity ratio a, we can maximize its EU utilization by allocating approximately $\sqrt{a}$ times more SAs than VUs.

The total quantity of EUs is determined by the smallest possible value that satisfies the SLO for the ML workload running in the guest VM. For the workloads that do not have specified SLO, they will get only one SA and one VU, as small instances can obtain idle EUs more easily. As for the memory allocation for an instance, NeuCloud relies on the DNN compiler to estimate the total amount of memory needed by the DNN workload. The ML compilers will compile the NPU program using the given SA/VU configuration [32].

3.4 vNPU Mapping

Initial mapping. To allocate a new vNPU, the vNPU manager takes the SA/VU configuration and requested memory size as the input. It aims to fit as many vNPUs as possible on a physical NPU to maximize the utilization of both the EU and memory resources.

To maximize the EU utilization, the vNPU manager in NeuCloud groups vNPUs, such that the total number of EUs of all vNPUs is as large as possible without exceeding the number of available EUs on an NPU core. For example, for an NPU with 4 SAs and 4 VUs, NeuCloud can map one vNPU demanding 2 SAs and 3 VUs with another demanding 2 SAs and 1 VU, so all EUs are utilized.

The vNPU manager also attempts to balance the number of allocated EUs and the size of allocated memory. This minimizes the chance that all EUs on one core are allocated but a large portion of its memory is not allocated, or vice versa. Therefore, vNPUs with many EUs and small memory will be collocated with vNPUs with few EUs and large memory.

For the case that all EUs have been allocated but there are still vNPUs to be mapped, NeuCloud provides the flexibility to allow vNPU instances to oversubscribe an NPU core. The scheduler will manage the vNPU execution at runtime and perform context switches between vNPUs (§3.5).

Dynamic remapping. After the initial mapping, the workload patterns, including the SA/VU intensity ratio and memory usage, may change over time. When a certain amount of vNPUs on the cloud are not running with optimized configurations, the NeuCloud scheduler will adjust the vNPU configurations and remap them to the physical NPUs.

The key challenge of vNPU remapping is to avoid moving workloads between physical cores, as it requires moving a large amount of data in the HBM across NPU cores or even across the low-bandwidth PCIe link. Thus, the scheduler will avoid vNPU migration across cores by remapping most vNPUs to the same NPU core they are originally on. In this way, the data remain stationary in HBM, and only the EU configuration needs to be updated.

3.5 vNPU Scheduling

After assigning vNPUs to different EUs on an NPU core, these vNPUs may become idle when there is no offloaded workload. This creates temporal underutilization. To address this issue, NeuCloud allows the oversubscription of EUs by mapping more vNPUs to the core. Therefore, when a vNPU is idle, another vNPU can utilize the EUs. When multiple vNPU instances require the same EU simultaneously, the vNPU scheduler needs to decide which vNPU can be executed.

There are two cases for NPU oversubscription. First, NeuCloud only allocates vNPUs that have the same SA/VU configurations to the same NPU cores. In this case, the NPU core is considered by the scheduler as multiple SA/VU partitions, and each partition is temporally shared by a set of vNPUs with identical SA/VU configurations. Similar to scheduling vGPUs on an NVIDIA multi-instance GPU [24], NeuCloud scheduler only needs to schedule vNPUs in each partition independently. Second, a more complicated case is that the vNPUs can have arbitrary SA/VU configurations. This means the scheduler must also consider dynamic vNPU mappings. We wish to explore this as future work.

3.6 vNPU Isolation

As multiple application instances share the same NPUs, we need to enforce runtime isolation between vNPU instances. This includes security isolation to prevent malicious attacks, and performance isolation to provide SLO guarantees.

Security isolation on an NPU core is enforced by memory address space isolation for the on-chip SRAM and off-chip HBM. For performance reasons, this can be achieved by using a hardware memory management unit (MMU) to

perform address translations and permission checks, such as the MMU in CPUs or GPUs [27]. One option is to support page-level address mapping on NPUs. However, since NPUs are intrinsically designed to support DNN models with moderate to large sizes, it could be sufficient to have coarse-grained memory segmentation rather than the page-level address mapping. This reduces the hardware complexity and performance overhead of address translation.

Given the deterministic data access patterns of DNN workloads, even though the hardware MMU is unavailable, address space isolation is still possible with static address translation in the hypervisor. Before offloading the ML workload, the hypervisor parses the instructions by conducting the boundary checking for each address. This is feasible for NPUs, since the NPU programming model has separate instruction and data address spaces, and it prohibits self-modifying codes and dynamic code generation.

Performance isolation can be enforced collaboratively with hardware and software techniques. For example, static partitioning of EUs already provides compute resource isolation. Memory bandwidth isolation can also be enforced by allocating an HBM channel to a vNPU. In addition, the vNPU scheduler can prioritize vNPUs with stricter SLOs, and avoid collocating vNPUs that are likely to have severe performance interference [5, 22, 23].

3.7 Integration with Cloud Infrastructures

In previous sections, we discussed how to virtualize a single NPU device and maximize its resource utilization. In reality, a data center consists of thousands of NPUs deployed on different host machines. To achieve improved resource utilization, a VM/container orchestration framework decides which VM/container is placed on which physical machine, according to its scheduling policies. In this section, we describe how to integrate NeuCloud into current cloud platforms.

As a case study, we can implement NeuCloud with KubeVirt/Kubernetes [11]. First, we integrate the vNPU manager into the KVM hypervisor, which exposes the vNPU instances as mediated PCIe devices to the guest VM [18]. The guest NPU driver also needs modifications to be aware of the paravirtualized interface like KVM hypercalls [20]. As KVM supports hot plugging of PCIe devices, it provides the system support for vNPU allocation and (re)mapping.

We can also extend the Kubernetes scheduler [2] to implement vNPU mapping policies. The `kube-scheduler` will assign a score to each available NPU node, and then rank all the nodes. After that, the VM is assigned to the node with the highest score. We can extend the scoring mechanism to rank the nodes by their remaining NPU hardware resources, such as the amount of free cores, EUs, and memory.

4 CONCLUSION AND FUTURE WORK

In this paper, we discuss system virtualization techniques for NPUs. We discuss the necessary hardware support to make NPU virtualization practical. We identify the key challenges of virtualizing NPUs, such as the need for fine-grained resource allocation, as well as security and performance isolation between vNPUs. As future work, we plan to develop a real system prototype of NeuCloud, and demonstrate its benefits for machine learning services.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their comments. This work was partially supported by NSF grant CCF-1919044, NSF CAREER Award 2144796, and the Hybrid Cloud and AI program at the IBM-Illinois Discovery Accelerator Institute.

REFERENCES

[1] 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/

[2] The Kubernetes Authors. 2023. Kubernetes Scheduler. https://kubernetes.io/docs/concepts/scheduling-eviction/kube-scheduler/

[3] 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/

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

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] Alibaba Clouder. 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

[11] The KubeVirt Contributors. 2023. KubeVirt.io. https://kubevirt.io/

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

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

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

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

[16] 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

[17] Bongjoon Hyun, Youngeun Kwon, Yujeong Choi, John Kim, and Minsoo Rhu. 2020. NeuMMU: Architectural Support for Efficient Address Translations in Neural Processing Units. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'20)*. Lausanne, Switzerland.

[18] Neo Jia and Kirti Wankhede. 2023. VFIO Mediated devices. https://docs.kernel.org/driver-api/vfio-mediated-device.html

[19] Norman P. Jouppi, Doe Hyun Yoon, George Kurian, Sheng Li, Nishant Patil, James Laudon, Cliff Young, and David Patterson. 2020. A Domain-Specific Supercomputer for Training Deep Neural Networks. *Commun. ACM* 63, 7 (June 2020).

[20] The kernel development community. 2023. Linux KVM Hypercall. https://docs.kernel.org/virt/kvm/x86/hypercalls.html

[21] Wolfram Alpha LLC. 2023. WolframAlpha: Computational Intelligence. https://www.wolframalpha.com/

[22] David Lo, Liqun Cheng, Rama Govindaraju, Parthasarathy Ranganathan, and Christos Kozyrakis. 2015. Heracles: Improving Resource Efficiency at Scale. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture (ISCA'15)*. Portland, OR.

[23] Jason Mars, Lingjia Tang, Robert Hundt, Kevin Skadron, and Mary Lou Soffa. 2011. Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'11)*. Porto Alegre, Brazil.

[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] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic Differentiation in PyTorch. In *Proceedings of the 30th International Conference on Neural Information Processing Systems (NIPS'17)*. Long Beach, CA.

[27] Bharath Pichai, Lisa Hsu, and Abhishek Bhattacharjee. 2014. Architectural Support for Address Translation on GPUs: Designing Memory Management Units for CPU/GPUs with Unified Address Spaces. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'14)*. Salt Lake City, UT.

[28] 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

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

[30] 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

[31] Google TensorFlow. 2023. Create production-grade machine learning models with TensorFlow. https://www.tensorflow.org/

[32] Google TensorFlow. 2023. XLA: Optimizing Compiler for Machine Learning. https://www.tensorflow.org/xla

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

[34] 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 International Symposium on Computer Architecture (ISCA'23)*. Orlando, FL.