PROCEEDINGS OF SCIENCE

OpenForBC, the GPU partitioning framework

Stefano Bagnasco,[a] **Alessio Borriero,**[a] **Gabriele Gaetano Fronzé,**[a] **Federica Legger,**[a] **Stefano Lusso,**[a] **Daniele Monteleone**[a] **and Sara Vallero**[a]

[a] *Istituto Nazionale di Fisica Nucleare, via Pietro Giuria 1, 10125 Torino, Italy*

E-mail: federica.legger@to.infn.it

In recent years, compute performances of GPUs (Graphics Processing Units) dramatically increased, especially in comparison to those of CPUs (Central Processing Units). GPUs are nowadays the hardware of choice for scientific applications involving massive parallel operations, such as deep learning (DL) and Artificial Intelligence (AI) workflows. Large-scale computing infrastructures such as on-premises data centers, HPC (High Performance Computing) centers, and public or private clouds offer high performance GPUs to researchers. The programming paradigms for GPUs significantly vary according to the GPU model and vendor, often posing a barrier to their use in scientific applications. In addition, powerful GPUs are hardly saturated by typical computing applications. GPU partitioning may be the key to exploit GPU computing power in an efficient and affordable manner. Multiple vendors proposed custom solutions to allow for GPU partitioning, often with poor portability across different platforms and OSs (Operating Systems).

OpenForBC (Open For Better Computing) is an open source software framework that allows for effortless and unified partitioning of GPUs from different vendors in Linux KVM virtualized environments. OpenForBC supports dynamic partitioning for various configurations of the GPU, which can be used to optimize the utilization of GPU kernels from different users or different applications. For example training complex DL models may require a full GPU, but inference may only need a fraction of it, leaving free resources for multiple cloned instances or other tasks. In this contribution we describe the most common GPU partitioning options available on the market, discuss the implementation of the OpenForBC interface, and show the results of benchmark tests in typical use case scenarios.

41st International Conference on High Energy physics - ICHEP2022
6-13 July, 2022
Bologna, Italy

© Copyright owned by the author(s) under the terms of the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License (CC BY-NC-ND 4.0).

PoS(ICHEP2022)221

1. Introduction

High-end GPUs, such as those based on Nvidia Ampere or Hopper technologies, AMD Instinct and Intel Ponte Vecchio, are becoming the de-facto standard for High-Performance Computing (HPC) and Artificial Intelligence (AI) applications. Computing kernels for General-Purpose GPUs (GPGPUs) are rarely able to fully exploit powerful GPUs with large amounts of memory. Their optimization is often not trivial and time consuming. Such optimisation might need to be tailored to a specific GPU architecture, with negative effects on code portability. On the other hand data centers offering GPUs to their users have the need to maximise their usage. To overcome such issues GPU vendors offer the possibility to partition high-end GPUs, i.e. it is possible to allocate fractions of the GPU computing power and memory to different processes and users. Executing multiple copies of a poorly optimized computing kernel on a partitionable GPU can potentially lead to an improved efficiency, reducing the performances cost.

GPU partitioning technologies vary across different vendors, and sometimes across different models. For example Nvidia offers two partitioning modes in virtualised environments: virtual GPU (vGPU) or multi-instance GPU (MIG) [1]. vGPU is an older standard based on time-sharing the compute capabilities of the GPU among different partitions. The GPU memory is statically partitioned. In a given time slot, the Virtual Machine (VM) with the attached vGPU partition has full control over the GPU computing resources, and access to only a fraction of the GPU memory. In MIG partitioning mode, both compute and memory resources of the GPU are statically partitioned. In this mode, the VM may only use a fraction of the GPU cores, even if the remaining ones are idle. MIG-partitioned GPUs can be used in bare metal or in containers, whereas vGPU instances can only be used in VMs. Newer architectures such as Ampere support both modes, while on older models such as Tesla only vGPU is available. The specificities of each technology and the poor support of GPU partitioning in Linux Kernel-based VM (KVM) environments often pose a severe barrier to the wide adoption of such solutions, even when the advantages of using partitioned GPUs are obvious.

While different implementations of GPU partitioning modes appear significantly different, they are all based on the same underlying standard, Single Root Input/Output Virtualization (SR-IOV) [2]. SR-IOV is a protocol developed by the PCI (Peripheral Component Interconnect) Express Consortium that allows different VMs to share a single PCI Express hardware interface. In Section 2, we describe the development of a software layer, the OpenForBC (Open For Better Computing) framework, offering a common interface to manage different GPU partitioning technologies in the open-source Linux KVM environment. By building on top of the SR-IOV standard, we ensure support for current partitioning solutions, and possibly future ones. To test the performances of GPUs partitioned with different modes and schemas through OpenForBC, we developed a modular benchmarking framework, OpenForBC Benchmark (see Section 3). Results of several benchmarks based on various Machine Learning (ML) models are shown and discussed in Section 4.

2. The OpenForBC framework

OpenForBC [3] is an open source software framework that provides a common interface to setup and use GPU partitions in Linux KVM hosts. OpenForBC is currently under development

PoS(ICHEP2022)221

and supports Nvidia GPUs compatible with both MIG and vGPU partitioning modes. OpenForBC requires that the Nvidia host driver and license are installed and available on the host system.

The OpenForBC framework is written in Python and offers its functionalities through a REST API. OpenForBC features a Command Line Interface (CLI) to execute a set of commands:

- `openforbc gpu list`: Lists the available physical GPUs that support a partitioning technology.
- `openforbc gpu types`: Lists the available virtual GPU profiles.
- `openforbc partition create`: Creates one of the available virtual GPU profiles.
- `openforbc partition get`: Gets the definition of the partition using the libvirt Virtualization API [4] XML standard. These are the information needed to instantiate a VM or container attached to the virtual GPU instance.
- `openforbc partition destroy`: Destroys the partition.

The OpenforBC framework has been tested with Nvidia cards A100 and T4, using both MIG and vGPU modes. It should be noted that graphical applications are not supported by Nvidia when the GPU is partitioned with MIG.

3. The OpenForBC Benchmark framework

To test the advantages of GPU partitioning, a set of benchmarks on promising use cases is needed. Due to the lack of a suitable option in the open-source market, we developed a benchmarking framework, OpenForBC Benchmark [5]. OpenForBC Benchmark is a modular benchmarking suite, written in Python, that allows to define set of benchmark runs based on common benchmark definitions, execute the benchmarks, and log the results. Benchmark execution can be easily steered via CLI. OpenForBC Benchmark includes our custom benchmarks, currently based on matrix multiplication and ML model training and inferences, and is compatible with the well-known Phoronix [6] and Blender [7] benchmarks. Further benchmarks can be easily added by providing the benchmark executable and its configuration parameters (for example the number of training epochs or size of the matrices) as a JSON file.

4. Results

GPU partitioning is a promising way to efficiently use high-end GPUs for a variety of workflows. For example training complex ML models is a typically computing-intensive workload, and may well occupy all GPU cores. Evaluating inference for the same model is usually much less demanding. OpenForBC Benchmark has been used to execute the following ML benchmarks in both training and inference mode:

- Teacher-Student: this benchmark implements a fully connected neural network with one hidden layer using the Teacher-student learning technique [8] in which a *student* model has to learn a dataset of input-output where the output distribution function is defined by a *teacher* network.

PoS(ICHEP2022)221

- CIFAR: this benchmark implements a convolutional neural network with two convolutional layers and one fully connected layer over the CIFAR10 dataset. CIFAR10 is a well known image recognition dataset [9].
- MNIST: this benchmark implements a fully connected neural network with three hidden layer and performs training and inference over the well-known MNIST [10] dataset of handwritten digits.

Inference is calculated on one sample at the time, i.e. with batch size equal to one. Training has been done using default parameters for the number of epochs, as reported in the OpenForBC Benchmark documentation [11].

In Figures 1 and 2, we present benchmark results of executing training and inference of the above mentioned ML models on a Nvidia A100 with 40 GB memory. Such graphical adapter can be divided to create up to seven partitions. We compare the throughput (defined as the number of processed samples per seconds) for various partitioning possibilities:

- *7 x 1/7*: the GPU is divided into seven partitions. These are the smallest partitions that can be created on a A100 card;
- *3 X 2/7*: the GPU is divided into three partitions;
- *2 x 3/7*: the GPU is divided into two partitions;
- *1 x 4/7*: the GPU is divided into one partition which has more than 50% of the GPU resources. Only one such partition can be created;
- *1 x 7/7*: the GPU is divided into one partition only. This is the largest partition that can be created on a A100 card;
- *whole GPU*: the GPU is used as a whole, without partitioning.

The fractions indicate roughly the ratio of computing cores associated to each partition with respect to the whole GPU. All partitions have been loaded with the same computation.

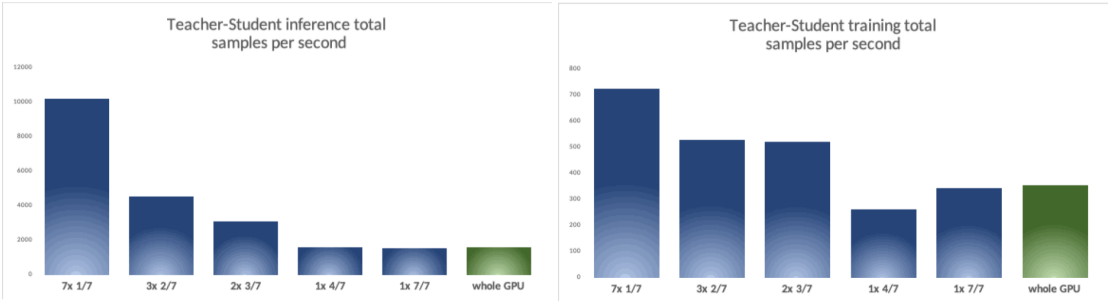

Figure 1: Throughput, defined as number of processed events per second, for training (left) and inference (right) on the Teacher-Student ML model, for several partitioning configurations. See text for details.

A few general remarks can be made. None of the tested models can fully saturate the computing resources of the GPU. This represents the ideal testing ground for GPU partitioning. The largest

PoS(ICHEP2022)221

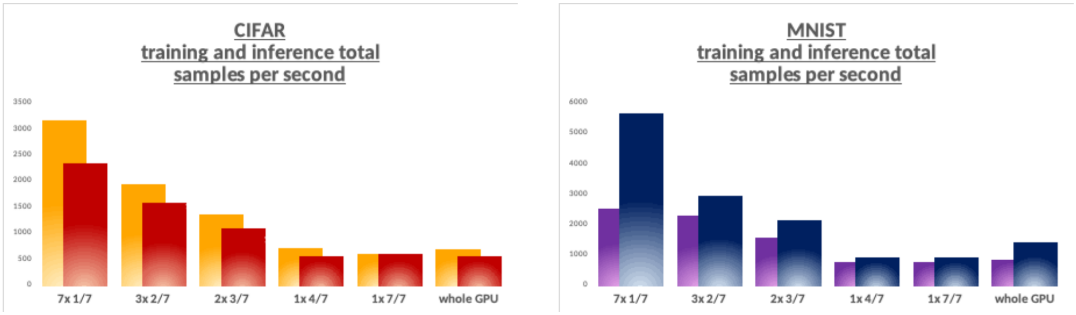

Figure 2: Throughput, defined as number of processed events per second, for training and inference on the CIFAR (left) and MNIST (right) ML models, for several partitioning configurations. See text for details.

speed-up (defined as increase in throughput) with respect to the unpartitioned GPU is generally reached by the partitioning schema involving seven small partitions. The speed-up varies from 105% in the case of Teacher-Student model training to 655% for Teacher Student model inference. For the CIFAR model the speedup during training and inference is similar, 461% and 428% respectively. For the MNIST model we measured a speedup of 305% in training and 409% in inference.

The overhead of GPU partitioning is negligible, as can be seen by comparing the throughput of the 1x7/7 partition with that of the whole GPU. The GPU power consumption merely rises from 130 W when using the unpartitioned GPU to 225 W when executing computations in the 7x1/7 partitioning mode. To summarise, these first tests clearly show the advantage of partitioning the GPU for workflows that cannot saturate the GPU. In the future we plan to expand the number of tested use cases, to probe the boundary when GPU partitioning may not be as efficient.

5. Conclusion

GPU partitioning may be the key to use efficiently high-end GPUs such as those found in modern data centers. GPU partitioning technologies significantly vary across different vendors, and even across different GPU models from the same vendor. We developed OpenForBC, a framework that offers a uniform interface to create and manage GPU partitions. This development paves the way for the wide adoption of GPU partitioning for both research and industry applications.

To test the advantages of GPU partitioning under various configurations we measured the performances of executing training and inference of several ML models, and found a significant speedup when using the fully partitioned GPU. The benchmarks were executed using OpenForBC Benchmark, a modular and open-source benchmarking framework developed in the context of this project.

OpenForBC currently supports Nvidia GPUs in MIG and vGPU partitioning modes. Future developments will include adding support for AMD and Intel GPUs, and further expansion of the benchmark tests.

References

[1] *NVIDIA Multi-Instance GPU User Guide*, https://docs.nvidia.com/datacenter/tesla/mig-user-guide/index.html (2022), accessed: 2022-08-05

PoS(ICHEP2022)221

[2] *Single Root I/O Virtualization*, https://pcisig.com/specifications/iov/single_root (2022), accessed: 2022-08-05

[3] *The OpenForBC Github repository*, https://github.com/Open-ForBC/OpenForBC (2022), accessed: 2022-08-05

[4] *Libvirt Virtualization API*, https://libvirt.org/ (2022), accessed: 2022-08-05

[5] The OpenForBC benchmark Tensorflow benchmarks, https://github.com/Open-ForBC/OpenForBC-Benchmark, accessed: 2022-08-08

[6] The Phoronix Test Suite, https://www.phoronix-test-suite.com/, accessed: 2022-08-09

[7] Blender Benchmark, https://opendata.blender.org/, accessed: 2022-08-09

[8] G. Hinton, O. Vinyals, and J. Dean, *Distilling the knowledge in a neural network*, (2015) arXiv:1503.02531

[9] A. Krizhevsky, *Learning Multiple Layers of Features from Tiny Images*, (2009) Technical Report, Canadian Institute For Advanced Research

[10] Y. LeCun, Courant Institute, NYU; C. Cortes, Google Labs, New York; C. J.C. Burges, Microsoft Research, Redmond, *The MNIST Database of handwritten digits*, http://yann.lecun.com/exdb/mnist/, accessed: 2022-08-08

[11] The OpenForBC benchmark Tensorflow benchmarks, https://github.com/Open-ForBC/OpenForBC-Benchmark/tree/main/benchmarks/tensorflow_benchmark, accessed: 2022-08-08

PoS(ICHEP2022)221