Benchmarking Machine Learning in HEP

AUGUST 2018

AUTHOR:
Sabina Manafli

CERN IT-CF
CMS

SUPERVISORS(S):

Felice Pantaleo
Luca Atzori

CERN openlab

Abstract

The interest on machine learning workloads in the HEP community has increased exponentially in the last years, making more and more important the need of a thorough benchmarking of the most relevant/significant workloads that are going to run on the experiments. The purpose of this project is to build a set of techniques to benchmark deep neural networks on different hardware. By using different tools and methodologies we make several important observations and conclusions based on the performance of deep learning application running on GPUs which have different compute capabilities.

Contents

1. Introduction

Training new deep learning models efficiently is one of the most important topics in machine learning. Given the model, it is important to train it as fast as possible and most efficiently by exploiting the computational resources of hardware.For this we need to choose the right hardware for our models. Therefore the next step should be thoroughly done benchmark.
In this paper we present benchmark of GANs on three different GPUs. We performed performance analysis of the model to gain interesting insights.
Followings are the main contributions of this project
1. Set of techniques to benchmark deep learning models. In order to benchmark model we need to choose relevant metrics for performance analysis. To measure this metrics on different GPUs which have different compute capabilities we need infer formulas that will be presented in Methodology part of paper
2. Findings and Recommendations. Based on the result of profiling and performance analysis we make several observations and recommendations.

2. Background

2.1 Application Selection

The use of Generative Adversarial Networks (GANs) is of particular interest because of potential applications for particle physics. Based on the fact that Generative Adversarial Networks(GANs) are widely used at CERN for fast detector simulation we decided to use GANs as an application for our benchmarking experiment. WGAN specifically is chosen for the benchmark as it is one of the leading models in the area of unsupervised learning algorithms.

2.1.1 GANs

Generative Adversarial Networks is unsupervised machine learning algorithm which trains two networks competing against each other:one generator and one discriminator network. The generator is trained to generate images based on given noise input and Discriminator is trained to distinguish real and generated images.

2.2 Framework Selection

There are many open-source frameworks for DNNs, such as TensorFlow Theano,Caffe ,Torch,PyTorch *et al*.To perform efficient computation of certain layers of DNNs they apply different memory managers, specific libraries. However most of them have similar code structure and high level API. Although there is not one single and best framework, we decided to choose tensorflow as it supports single and multiple GPUs, are being evolved consistently, and has many already implemented models of GANs.

2.3 Hardware

As different GPU models has different cost, performance, and power, it is critical to understand how the key metrics are affected by different GPUs in DNN training. Therefore the model is run using three types of GPU, the NVidia V100, K40C,K20Xm.
Table 2.1 below compares the technical specifications of three GPUs that we benchmarked.

	Tesla K20Xm	Tesla K40c	Tesla V100
Architecture	Kepler	Kepler	Volta
Floating point performance	3.935 GFLOPs	4.291 GFLOPs	14.131GFLOPS
Memory Bandwidth	249 G/s	288G/s	900G/s
Memory Size	6GB	12GB	16GB
Compute Capability	3.5	3.5	7.0

Table 2.1: Hardware Specifications

2.4 Environmental Setup

We use CentOS 7.5.1804, Tensorflow v1.2.0 and Tensorflow v1.3.0 with CUDA 8.0 and CuDNN 7.1.3.

3. Methodology

In this section we explain methodology and tool chain we used to get relevant metrics to measure performance of model.

3.1 Profiling via sampling

DNNs can take hours or even days to train them which makes it impractical to profile entire process. However, as training is iterative process and all iterations have the same computation logic,we can profile only few iterations to get the desired metrics. It is important to profile when training reaches stable state and it reaches this state after few hundreds of iterations. [1].In our case we profiled only 5 iterations after 300th iteration.

3.2 Relevant Metrics

GPU Compute Utilization: The GPU is the main unit behind DNN training and our aim is to keep the GPU busy all the time. Therefore it is important to examine the utilization of GPU. We measure GPU Compute utilization as the fraction of time when GPU is busy.

$$GPU\ Utilization = GPU\ Active\ time\ * 100\ /total\ elapsed\ time$$

We can get this metric by simply running the code using *nvprof* commannd as below

```
nvprof -o output.nvvp --print-gpu-summary python yourcode.py
```

By importing the output file *output.nvvp* to Nvidia Visual Profiler we can get numbers for GPU Compute Utilization.

FP32 Utilization: The training on DNNs is perfomed
Single precision floating points are number formats that takes 32 bit computer memory. In deep learning training calculations are performed using floating point operations.Therefore it is important to see how the FP32 resources of GPU are being utilized while it is active:

$$FP32\ Utilization\ = actual\ flop\ count\ during\ T\ *100\ /max\ number\ of\ FLOPS\ *total\ elapsed\ time$$

Profiling FP32 Utilization means is looking at the GPU Utilization from different angle.
For GPUs with compute capability 7.0 , V100 in our case, it is easier to get FP32 utilization metrics. To get this particular metrics we need to specify

```
single_precision_fu_utilization
```

as an option when running *nvprof* command as shown below

```
nvprof --profile-from-start off --metrics single_precision_fu_utilization -o output
    .nvvp  --print-gpu-summary python yourcode.py
```

After we get the output file we can import it to Nvidia Visual Profiler to visualize our result. What we see in profiler is FP32 utilization of individual kernels in the scale of 1 to 10. In order get the average utilization we need to find the average of all values.
GPUs with compute capability 3.5 does not support use of

```
single_precision_fu_utilization
```

metrics. We need to find which metrics are used to calculate single precision utilization.
According to the answer in this forum we need to know following measures in order to calculate single precision utilization:

- number of instructions executed on multiprocessor function unit
- active cycles
- maximum number of instructions that can be executed in single cycle on multiprocessing function unit

Following metrics can be used to get first two above listed measures: inst_count,-e active_cycles. Number of instructions that can be executed in single cycle on multiprocessing function unit can be obtained from Floating point performance of GPU. In our case it has been mentioned in Table 2.1

4. Results

As previously mentioned, this analysis will focus on a following key metrics: GPU compute uilization and FP32 utilization The focus of project is to observe how these metrics change as we change mini-batch size on different GPUs. More specifically, the effect of key hyper parameter,mini-batch size, on mentioned metrics will be studied.

4.1 GPU Compute Utilization

Figure 4.1 shows the GPU compute utilization on three different GPUs as we change mini batch-size. Following observations have been made. *Observation 1:* The mini-batch size should be large enough to keep the GPU busy. Larger mini batch size means GPU spends more time doing computations rather than invoking and finishing small kernels.[1]

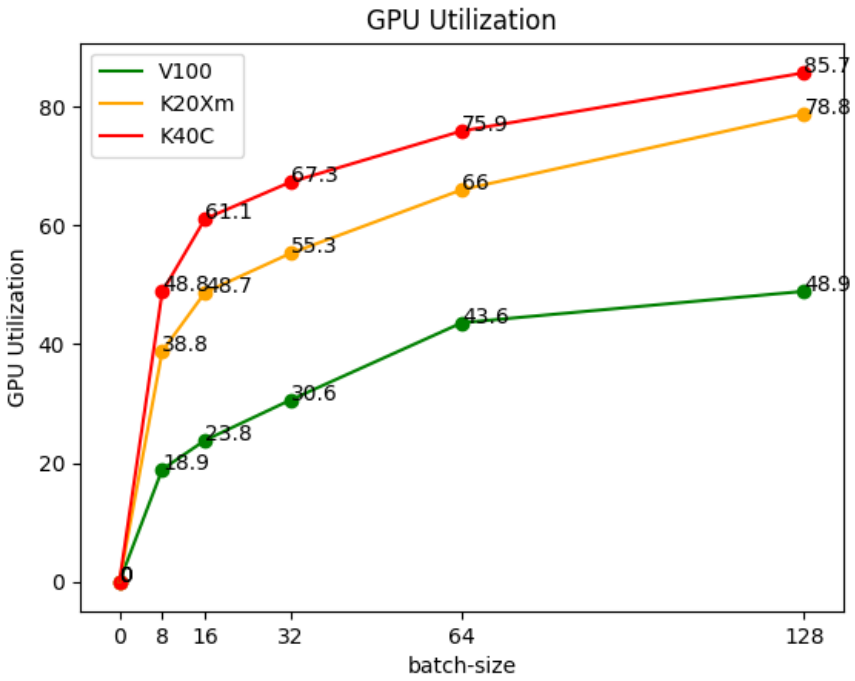

Figure 4.1

Observation 2: More advanced GPUs should be accompanied by better systems designs and more efficient low-level libraries. V100 usually helps improving training time and throughput,however the computation power of V100 is not well and fully utilized. Hence it can be concluded that, although V100 is computationally more powerful the proper utilization of these re-

sources requires a more careful design of algorithms that can efficiently exploit these resource.

4.2 FP32 Utilization

Figure 4.2shows the FP32 utilization on three different GPUs as we change mini batch-size. K40C achieves 85.7% utilization while it is 48.9% in V100 GPU.
We make the following observations Observation 1: As expected the mini-batch size improved FP32 utilization.
Observation 2: As observed in previous section, FP32 is not utilized in V100.

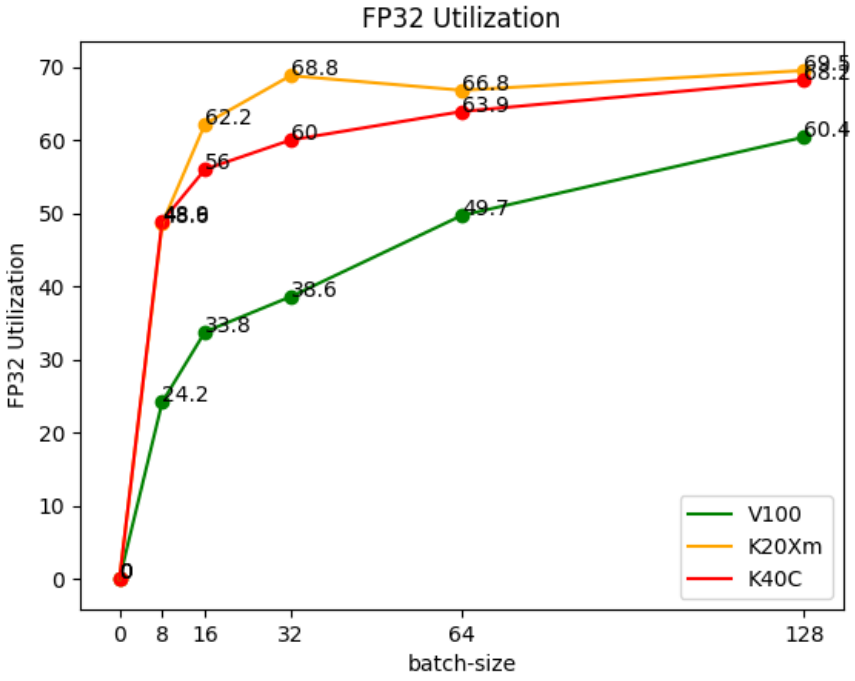

Figure 4.2

5. Future Work

This project set the foundation of techniques to benchmark given deep learning application on GPUs. This project can be further developed by benchmarking more machine learning applications on more GPUs.

6. References

Bibliography

[1] Zhu, Hongyu and Akrout, Mohamed and Zheng, Bojian and Pelegris, Andrew and Phanishayee, Amar and Schroeder, Bianca and Pekhimenko, Gennady *TBD: Benchmarking and Analyzing Deep Neural Network Training* 4, 6

[2] Nvidia visual profilers user guide
`https://docs.nvidia.com/cuda/profiler-users-guide/index.html`

[3] Machine Learning
`https://github.com/David-Levinthal/machine-learning`