Hardware Accelerated ATLAS Workloads on the WLCG Grid

A C Forti[1], L Heinrich[2] and M Guth[3]

[1] School of Physics and Astronomy, University of Manchester, Oxford Road, Manchester, M13 9PL, UK.

[2] CERN (European Laboratory for Particle Physics), Rue de Geneve 23 CH 1211 Geneva, Switzerland.

[3] Albert Ludwigs Universität Freiburg, Friedrichstr. 39, 79085 Freiburg im Breisgau, Germany.

Abstract. In recent years the usage of machine learning techniques within data-intensive sciences in general and high-energy physics in particular has rapidly increased, in part due to the availability of large datasets on which such algorithms can be trained, as well as suitable hardware, such as graphic or tensor processing units, which greatly accelerate the training and execution of such algorithms. Within the HEP domain, the development of these techniques has so far relied on resources external to the primary computing infrastructure of the WLCG (Worldwide LHC Computing Grid). In this paper we present an integration of hardware-accelerated workloads into the Grid through the declaration of dedicated queues with access to hardware accelerators and the use of Linux container images holding a modern data science software stack. A frequent use-case in the development of machine learning algorithms is the optimization of neural networks through the tuning of their Hyper Parameters (HP). For this often a large range of network variations must be trained and compared, which for some optimization schemes can be performed in parallel – a workload well suited for Grid computing. An example of such a hyper-parameter scan on Grid resources for the case of flavor tagging within ATLAS is presented.

1. Introduction

The increase in dataset size and computing resource requirements for the HL-LHC is pushing WLCG experiments to look at Machine Learning (ML) techniques to improve the efficiency of data analysis and processing [1]. ML software is better suited to run on GPUs (Graphics Processing Unit), a type of hardware usually not available on shared resources and which ML users have difficulties accessing. Having them on the Grid would increase the ability of users and developers to test and improve the ML techniques. In this context there are three aspects we'd like to show

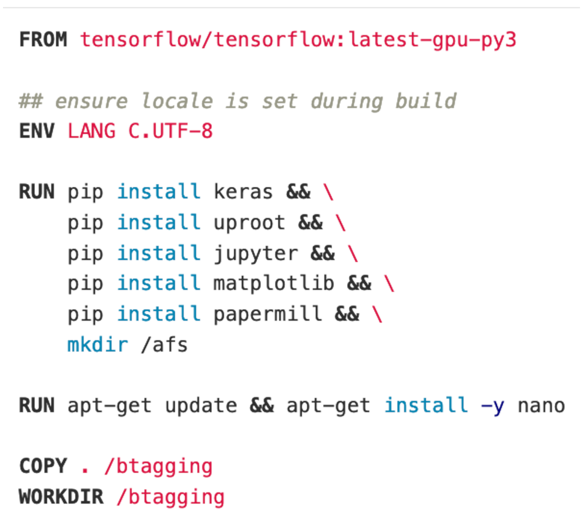
- ML software is not part of the more traditional LHC experiments software stack and is not distributed to sites. We solved this problem by using ML containers that can be easily built by the users themselves.
- Due to the usually parallel tightly coupled nature of the code running on GPUs, GPUs brokering has always been considered as an intractable problem. We can identify classes of loosely coupled ML jobs that can already be submitted using similar brokering rules to those we use already. This work can be used as an initial step towards more complicated workflows that we might be able to run on HPC resources.

- The neural network b-tagging algorithm, used by ATLAS, we present is an example of how new and improved ML algorithms can be adapted to be brokered to grid resources.

2. User containers

Running ML jobs requires custom software like Python3 and TensorFlow [2], which is not part of the default OS installation or of the usual experiment stack that is distributed to sites. This makes it difficult to run these jobs on the Grid without a large organisational effort.

To ease this problem, ATLAS has worked on integrating containers in the pilot infrastructure. In particular, we introduced a mechanism to run user-defined containers [3] which can run in a standalone manner. These type of containers offer isolation from the host environment, and the advantage to the users is that they can build the images containing all the software the application needs without having to rely either on the system administrators to install it or the experiment to distribute it. The advantage of using Docker [4] to build these images is that there is a vast repository of official images already built containing ML software, and users have just to customise it and add their own application on top of it by writing a Dockerfile, an image definition file. Figure 1 shows a simple Dockerfile to add a ML user application layer on top of the TensorFlow image from the official Docker library. The ATLAS ML group is now also providing base images for ATLAS users.



```
FROM tensorflow/tensorflow:latest-gpu-py3

## ensure locale is set during build
ENV LANG C.UTF-8

RUN pip install keras && \
    pip install uproot && \
    pip install jupyter && \
    pip install matplotlib && \
    pip install papermill && \
    mkdir /afs

RUN apt-get update && apt-get install -y nano

COPY . /btagging
WORKDIR /btagging
```

Figure 1. Example of Dockerfile. Users can easily build on top of official images.

3. GPU drivers and libraries

Software like TensorFlow can run in a heterogenous environment, and even if there are GPUs on the host, it will fall back on running on CPUs if it cannot properly access the former. To run on GPUs the application in the container has to be able to access the devices and load the GPU drivers which are vendor and model dependent. Singularity [5], the container runtime of choice on the Grid, can be configured to do this automatically. In most cases it is just a matter of adding a list of libraries to a configuration file, but at some sites this might be more complicated. For example, as well as being added to the container runtime configuration, the libraries might have to be explicitly preloaded by each job before it starts.

Since each site will have a slightly different configuration and the software may silently fall back to running on CPUs, therefore wasting resources, we created some simple test containers which detect if the GPUs exist and if they are correctly configured for the containerised applications to use them. These images can be used either by the system administrators or by ATLAS operations when setting up nodes and queues to access them. In the future ATLAS will use such simple images to verify that sites are functional as part of its testing infrastructure.

4. Enabling GPUs on the WLCG Grid

The WLCG distributed resources have been built around the HTC (High Throughput Computing) paradigm that focuses on the efficient execution of a large number of loosely-coupled tasks. The environment has been traditionally uniform, not only at OS and software levels but also at the hardware level with most computing centres adopting Intel CPUs. The software and algorithms it was built for are essentially sequential and can run on commodity hardware.

GPUs, by contrast, are designed to process parallel code, possibly be used as co-processors, i.e. only part of the code is offloaded on them, and therefore might have much more complicated requirements when they are to be used by a batch job. As we will show though, there are some categories of ML workflows that run on GPUs and can be adapted to the HTC model.

5. Sites setup and brokering

Traditionally, loosely coupled tasks have been handled by splitting them into independent jobs that could run anywhere. In batch system terms, each job occupies a slot, each slot has a determined set of resources allocated to it. The easiest way to put online GPU resources so far has been by maintaining the same principle, in this case one GPU slot per node. Sites enforce such constraints either by virtualizing the worker nodes using VMs bound to a fixed amount of resources that will always be allocated when a GPU is requested, or, more recently by using cgroups, a linux kernel feature to control resources, to impose similar limitations. With this setup we could broker jobs to GPU resources at multiple sites with two simple tweaks to the system:

- Adding a selectable vendor nvidia-gpu label to select the queues. Jobs can run on any nvidia GPUs that are accessible.
- Avoid standard jobs being brokered to GPU resources. In the ATLAS workflow sites only see pilot jobs sent by the same production user. They cannot distinguish a pilot job that will request a GPU payload from a pilot that will request a standard payload. It's up to the experiment to broker the correct pilots to the GPU resources.

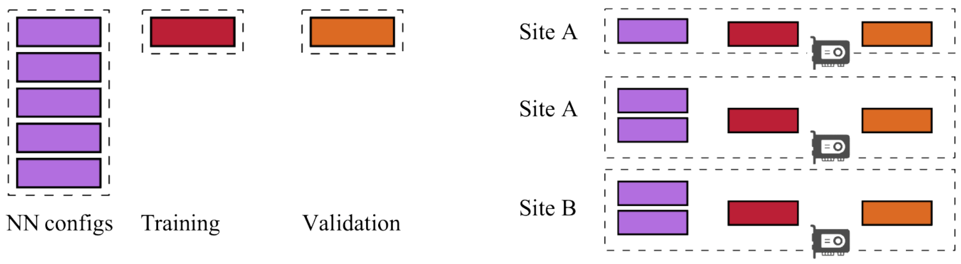


Figure 2. Neural net input and job splitting over multiple Grid sites

6. b-tagging Hyper Parameter optimization

Some workloads are more suited than others to run on HTC environment. The Hyper Parameter (HP) scan, used by ATLAS for b-tagging [6], is an embarrassingly parallel workload and can be split in several independent jobs each running on a GPU. The optimisation presented here is set up to scan 800 combinations spanning 6 Hyper Parameters dimensions (3 layers, learning rate, batch size and activation functions). The workload has been split in 10 jobs with 80 combinations each. Each job ran on the same training and validation data. The input files, small JSON files containing the configuration for each combination, were replicated to the sites with GPUs. Figure 2 shows how the task was split.

The output were also small JSON files containing the optimization values for the validation loss for each HP combination. The smaller the validation loss the better the combination. The results are illustrated in a parallel coordinates plot in Figure 3. Results were then verified by

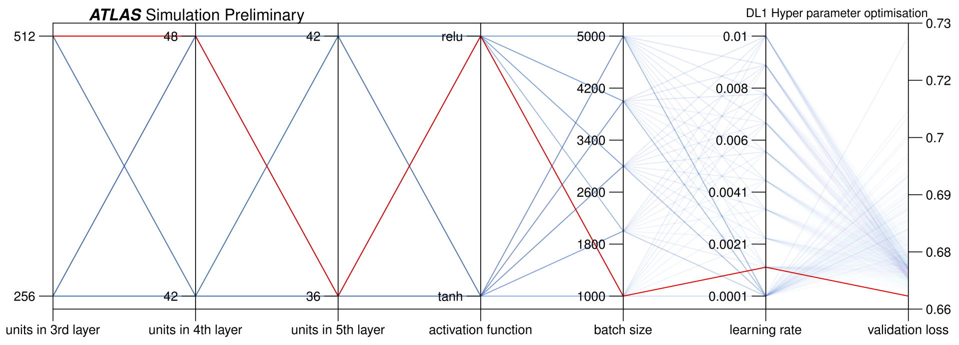


Figure 3. Parallel coordinates plot for 800 different Hyper Parameter combinations. The lines show different combinations of configurations represented in each axis. The last axis shows the neural network loss in the validation sample for a given configuration. The red line shows the Hyper Parameter configuration with the smallest validation loss [7].

running again the training on best, medium and worst validation loss combinations of HPs to produce a ROC (Receiver operating characteristic) plot. The ROC curves as shown in figure 4 are calculated using the following equation:

$$\mathrm{DL1}_{\text{b-score}}(f_c) = \ln\left(\frac{p_b}{f_c \cdot p_c + (1 - f_c) \cdot p_{\text{light-flavour}}}\right), \tag{1}$$

where p_b is the b-jet probability, p_c the c-jet probability and $p_{\text{light-flavour}}$ - the light flavour jet probability, all obtained from the network evaluation, and f_c is the c-jet fraction. The fact that the neural network has an output layer with three categories allows, besides b-jet-tagging, also c-jet-tagging without retraining by redefining the likelihood from equation (1).

Comparing the results from the Hyper Parameter optimisation with the results from the BDT based tagger MV2c10rnn [8], marked as red star in figure 4, we observe a great improvement already with this simplified approach where the simplification is mainly the high batch size in order to speed up the single trainings.

7. Conclusions

The use of GPUs in ATLAS and more generally in WLCG may increase due to the introduction of ML and resources coming online at sites, particularly, but not only, at HPC centres. For loosely coupled workloads, like the b-tagging HP scan presented here, users can access GPUs at standard Grid sites already with minimal changes to their client commands. In addition, there is ongoing work to run this workflow on larger resources at 3 HPC sites and to implement a more sophisticated brokering to run workloads via the production system. Additionally, based on the results presented here, work is ongoing to enable the deployment of more complex workloads,

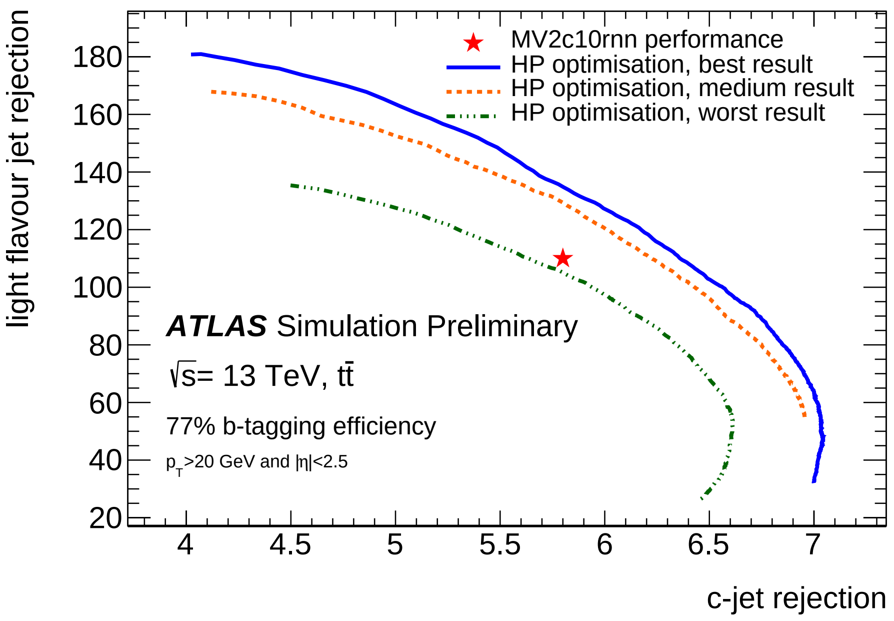


Figure 4. Three ROC curves are shown for different Hyper Parameter combinations (best, medium and worst performing). The evaluation is done for 77% b-tagging efficiency varying the c-fraction parameter in the calculation of the c and light flavour jet rejection. The MV2c10rnn performance is extracted from [8], where MV2c10rnn utilises the same inputs as DL1 presented here and additional inputs related to soft muons [7].

such as the training of distributed generative adversarial networks (GANs)[9], developed in the context of fast calorimeter simulation, on the ATLAS computing infrastructure.

References

[1] Albertsson K et al. 2018 Machine Learning in High Energy Physics Community White Paper, *J. Phys. Conf. Ser.* **1085**, no. 2, 022008

[2] Abadi M, Agarwal A et al. 2016 Tensorflow: Large-scale machine learning on heterogeneous distributed systems *Preprint* arXiv:1603.04467

[3] Heinrich L, Forti AC, Nilsson P and Maeno T 2019 Continuous Analysis in ATLAS: Running User-Defined Container Images on the Grid URL `https://indico.cern.ch/event/708041/contributions/3276174/`

[4] Docker URL `https://www.docker.com/resources/what-container`

[5] Kurtzer GM, Sochat V and Bauer MW 2017 Singularity: Scientific containers for mobility of compute. *PLoS ONE* 12(5): e0177459

[6] Paganini M and ATLAS Collaboration 2018 Machine Learning Algorithms for b-Jet Tagging at the ATLAS Experiment *J. Phys. Conf. Series* **1085** 042031

[7] ATLAS Collaboration 2019 Hyper Parameter Scan with the Deep Learning Heavy Flavour Tagger (DL1) URL `http://atlas.web.cern.ch/Atlas/GROUPS/PHYSICS/PLOTS/FTAG-2019-001/`

[8] ATLAS Collaboration 2017 Optimisation and performance studies of the ATLAS b-tagging algorithms for the 2017-18 LHC run ATL-PHYS-PUB-2017-013 URL `https://cds.cern.ch/record/2273281`

[9] Salamani D et al 2018 Deep Generative Models for Fast Shower Simulation in ATLAS *IEEE 14th International Conference on e-Science* Amsterdam, pp. 348-348.