PROCEEDINGS OF SCIENCE

VegasFlow: accelerating Monte Carlo simulation across platforms

Juan M. Cruz-Martinez[a,*] **and Stefano Carrazza**[a]

[a] *TIF Lab, Dipartimento di Fisica, Università degli Studi di Milano and INFN Sezione di Milano*
Via Celoria 16, 20133, Milano, Italy

E-mail: juan.cruz@mi.infn.it, stefano.carrazza@mi.infn.it

In this work we demonstrate the usage of the VegasFlow library on multidevice situations: multi-GPU in one single node and multi-node in a cluster. VegasFlow is a new software for the fast evaluation of highly parallelizable integrals based on Monte Carlo integration. It is inspired by the Vegas algorithm, very often used as the driver of cross section integrations, and based on Google's powerful TensorFlow library. In this proceedings we consider a typical multi-GPU configuration to benchmark how different batch sizes can increase (or decrease) the performance on a Leading Order example integration.

40th International Conference on High Energy physics - ICHEP2020
July 28 - August 6, 2020
Prague, Czech Republic (virtual meeting)

[*] Speaker

1. Introduction

State-of-the-art computations in High Energy Physics (HEP) require computing complex multi-dimensional integrals numerically, as the analytical result is often not known. Monte Carlo (MC) algorithms are generally the option of choice for these kind of integrals, be it when considering HEP applications or elsewhere, as the error of such algorithms does not grow with the number of dimensions.

In particular, in the HEP literature, MC methods based on the idea of importance sampling are widespread as they combine the robustness of MC algorithms for high dimensional situations with the flexibility of adaptive grids.

The Vegas algorithm [1, 2] is one of the main drivers for multi-purpose parton level event generation programs based on fixed order calculations such as MCFM [3, 4] or NNLOJET [5] and is also present in more general tools such as MG5_aMC@NLO [6] and Sherpa [7]. Whereas the original implementation of the algorithm was written for a single CPU, nowadays it is usually implemented to take advantage of multi-threading CPUs and distributed computing. Indeed, MC computations are what is informally known as “embarrassingly parallel”.

However, the parallelization of a computation over multiple CPUs does not decrease the number of CPU-hours required to complete a computation and the cost of such calculations is driving the budget of big science experiments such as ATLAS or CMS [8]. With VegasFlow [9, 10] we implement the importance sampling techniques from Vegas to run both in CPUs and GPUs, enabling further acceleration of complicated integrals by leveraging the abstraction possibilities offered by TensorFlow [11].

2. A toolset for a new generation of Monte Carlo generators

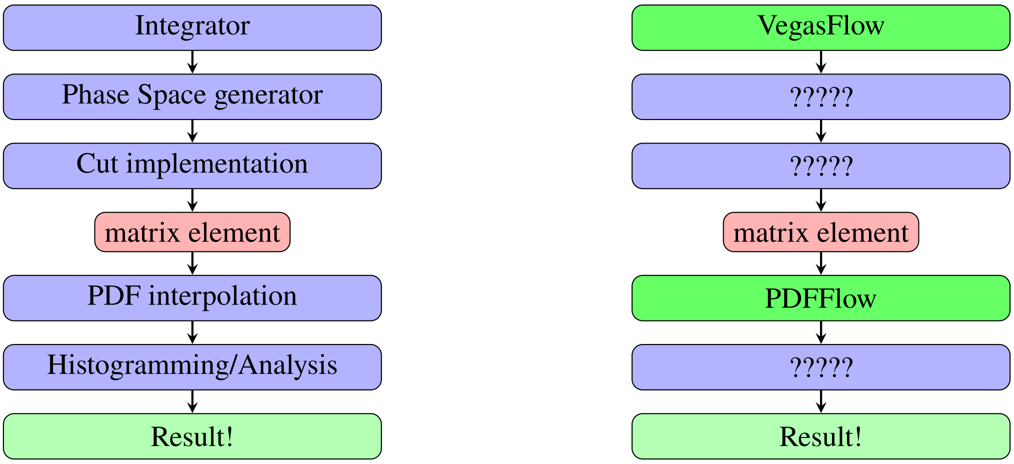
Monte Carlo generators are at the core of HEP phenomenology, and they have been for many years. As a result a plethora of tools, libraries and algorithms have been developed around these programs. Some examples of this are the LHAPDF library [12], the RAMBO [13] phase space generator or the Vegas algorithm mentioned above. This brings us to the situation in Fig. 1a where, if one would like to write a new program one can also take advantage of existing tools, greatly reducing the amount of effort necessary to produce results.

For GPU computations, however, the situation is much worse: many (if not most) of the necessary tools need to be written from scratch. Not only the ones we are interested in (in the example of Fig. 1b, the matrix element), but also auxiliary elements such as the integrator or the PDF interpolation library.

With VegasFlow [9] and PDFFlow [14] we provide two of these tools which we hope will kick-start a new era of truly performant HEP phenomenology where the new advances in hardware are exploited to the fullest.

3. VegasFlow

GPU computing has become ubiquitous in the world of High Performance Computing (HPC). These devices provide a way of enormously accelerating parallel computations reducing both



(a) Schematic view of the different ingredients that form a Monte Carlo generator

(b) For GPU computation, most of the ingredients need to be coded from scratch.

Figure 1: Schematic view of the ingredients necessary to write a new Monte Carlo generator for a new process. In this case the ideal situation is such we can only worry about the matrix element and use existing tool for the rest of the program (left). For GPU computing, however, this cannot yet be the case, as only some of the tools exist (right).

computing time and power consumption. With VegasFlow we aim to eliminate the technological gap between the HPC and HEP communities by providing a library that can silently offload very complicated calculation to hardware accelerators.

A paramount example is the simulation of proton-proton collisions considered above, with PDFFlow [14] we already offload the computation of the initial state to the GPU, with VegasFlow [9] the numerical Monte Carlo is also offloaded to the GPU. At this point only the actual matrix element, describing the physics of the interaction, and its phase space need to be written by the user.

A detailed description of the code, together with examples, can be found in the documentation of the library library [10].

One of the challenges of HPC is to manage multidevice computation. In this proceedings we will mention two of the use cases considered in VegasFlow: running on multiple-GPUs in one single machine and running on a cluster.

3.1 Multi-GPU

The number of events that can be run in parallel on a GPU is directly related to the memory capacity of the device. As a rule of thumb, running more events at once will reduce the latency associated with the communication between the GPU and the CPU. On the other hand, and depending on the specifics of the hardware, it might be preferable to reduce the number of points per batch. One reason to do so is to ensure that all GPUs (if there is more than one) receive a share of the data so no device is idle at any point.

In order to control the amount of points that VegasFlow will send to the GPU we provide the keyword `events_limit`. This variable becomes especially relevant for multidevice computation. It is clear from Fig. 2 that there is no one-trick value as different devices will work differently as the batch size changes. We observe that the performance of the calculation on the GPU is impacted

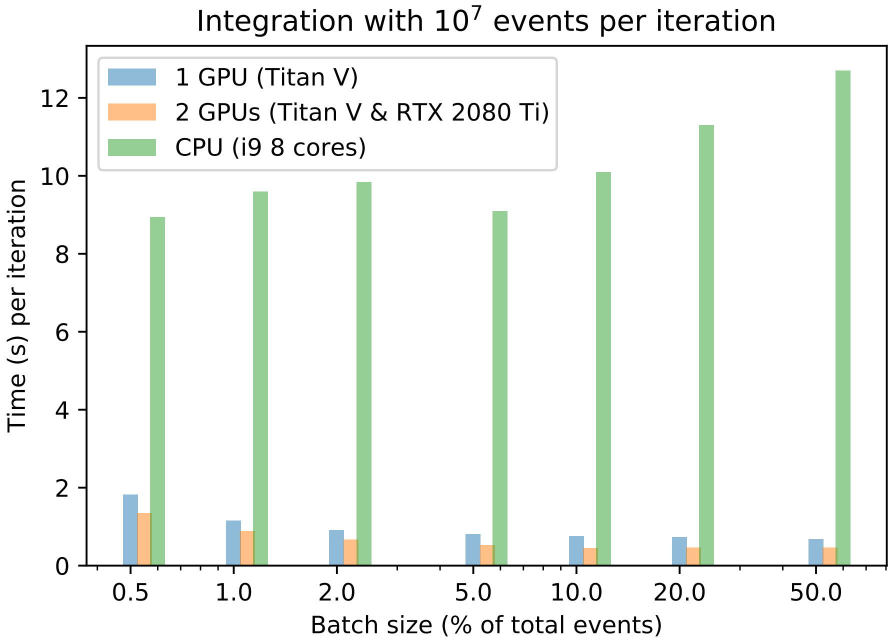


Figure 2: Example of the effect of running an integration on different devices with different batch sizes. Greater batch sizes can improve the performance of the GPU by minimizing I/O latency. For CPU computation, where this issue doesn't exist, a greater batch size actually hurts performance. The CPU used for this example is a Intel i9-9980XE. The GPUs are NVIDIA's Titan V (working alone in 1 GPU mode) and GeForce RTX 2080Ti. The integration corresponds to the single top example shown in [9].

when the number of events per batch is too small. This can be easily understood, as for a batch size of 50% of the events, there are only two transfers of data from the CPU to the GPU, while for a batch size of 2% there will be 50 data transferring operations. We also observe that, once a certain threshold is reached, increasing the size of the batch does not impact the performance. This threshold depends on the hardware (how fast is the data transfer) and the calculation (for how long will the GPU be computing) and it is the point at which the transfer time is offset by the calculation itself.

Below we show an example of how to set the batch size using `events_limit` in actual code with a small toy integrand:

```
>>> from VegasFlow import VegasFlow
>>>
>>> def complicated_integrand(xarr, **kwargs):
>>>     return tf.reduce_sum(xarr, axis=1)
>>>
>>> n_dim = 10
>>> n_events = int(1e6)
>>> integrator = VegasFlow(n_dim, n_events, events_limit = int(1e5))
>>> integrator.compile(complicated_integrand)
>>> res = integrator.run_integration(n_iter = 5, log_time = True)
```

Finally, one other interesting aspect on Fig. 2 is that for the CPU the behaviour is opposite to that of the GPU. In this case there is no transfer of data to be performed and so an increase of the batch size has a negative impact as VegasFlow needs to allocate a bigger chunk of RAM.

3.2 Cluster

A common use case for any parallelizable Monte Carlo algorithm is cluster computation. In the simplest scenario one would just send a separate instance of the program to each node of the cluster, combining the results afterwards in a consistent way.

This, however, presents a challenge for adaptable Monte Carlo algorithm such as VegasFlow, as every iteration informs the subsequent run. We overcome this problem by implementing an interface to the Dask [15] library.

The main advantage of implementing an interface to Dask is that it means automatic support for all Dask-supported backends. The authors must warn, however, that it has not been possible to test it beyond the systems available to them, so only the SLURM [16] system can be guaranteed to work.

Below we extend the previous example where a `SLURMCluster` instance from Dask is configured and passed to VegasFlow. After calling the method `set_distribute` with the reference to the chosen cluster, the method `run_integration` will send the job to the cluster instead of running them in the local computer. An example on how VegasFlow scales is given in Table 1.

```
>>> from dask_jobqueue import SLURMCluster
>>>
>>> cluster = SLURMCluster(queue="<q>", project="<p>", cores=4, memory="2g")
>>>
>>> integrator.set_distribute(cluster)
>>> res = integrator.run_integration(n_iter)
```

	1 node	2 nodes	3 nodes
Time per iteration (s)	59.7s	33.3s	22.3s

Table 1: How an integration scales in a SLURM cluster as a function of the number of nodes active. Using 4 cores per node. Note that time does not scale linearly with the number of nodes as a greater number of nodes also means a greater number of data transferring operations.

4. Conclusions

We have demonstrated the performance gains that can be obtained from hardware accelerators (in a single node or in a computing cluster) by using the VegasFlow library. We show how the improvement is sensitive to the distribution strategy. Indeed, in a fashion similar to machine learning techniques, finding the right batch size can make a notable difference on the performance.

Acknowledgments

The authors are supported by the European Research Council under the European Union's Horizon 2020 research and innovation Programme (grant agreement number 740006) and by the UNIMI Linea 2A grant "New hardware for HEP".

References

[1] G. P. Lepage, *A New Algorithm for Adaptive Multidimensional Integration*, *J. Comput. Phys.* **27** (1978) 192.

[2] G. P. Lepage, *VEGAS: AN ADAPTIVE MULTIDIMENSIONAL INTEGRATION PROGRAM*, 1980.

[3] J. M. Campbell, R. K. Ellis and W. T. Giele, *A Multi-Threaded Version of MCFM*, *Eur. Phys. J.* **C75** (2015) 246 [1503.06182].

[4] J. Campbell and T. Neumann, *Precision Phenomenology with MCFM*, *JHEP* **12** (2019) 034 [1909.09117].

[5] T. Gehrmann et al., *Jet cross sections and transverse momentum distributions with NNLOJET*, *PoS* **RADCOR2017** (2018) 074 [1801.06415].

[6] J. Alwall, R. Frederix, S. Frixione, V. Hirschi, F. Maltoni, O. Mattelaer et al., *The automated computation of tree-level and next-to-leading order differential cross sections, and their matching to parton shower simulations*, *JHEP* **07** (2014) 079 [1405.0301].

[7] T. Gleisberg, S. Hoeche, F. Krauss, M. Schonherr, S. Schumann, F. Siegert et al., *Event generation with SHERPA 1.1*, *JHEP* **02** (2009) 007 [0811.4622].

[8] A. Buckley, *Computational challenges for MC event generation*, in *19th International Workshop on Advanced Computing and Analysis Techniques in Physics Research: Empowering the revolution: Bringing Machine Learning to High Performance Computing (ACAT 2019) Saas-Fee, Switzerland, March 11-15, 2019*, 2019, 1908.00167.

[9] S. Carrazza and J. M. Cruz-Martinez, *VegasFlow: accelerating Monte Carlo simulation across multiple hardware platforms*, *Comput. Phys. Commun.* **254** (2020) 107376 [2002.12921].

[10] J. Cruz-Martinez and S. Carrazza, *N3pdf/vegasflow* `VegasFlow.readthedocs.io`, Feb., 2020. 10.5281/zenodo.3691926.

[11] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro et al., *TensorFlow: Large-scale machine learning on heterogeneous systems*, 2015.

[12] A. Buckley, J. Ferrando, S. Lloyd, K. Nordström, B. Page, M. Rüfenacht et al., *LHAPDF6: parton density access in the LHC precision era*, *Eur. Phys. J. C* **75** (2015) 132 [1412.7420].

[13] R. Kleiss, W. Stirling and S. Ellis, *A New Monte Carlo Treatment of Multiparticle Phase Space at High-energies*, *Comput. Phys. Commun.* **40** (1986) 359.

[14] S. Carrazza, J. M. Cruz-Martinez and M. Rossi, *PDFFlow: parton distribution functions on GPU*, 2009.06635.

[15] Dask Development Team, *Dask: Library for dynamic task scheduling*, 2016.

[16] A. B. Yoo, M. A. Jette and M. Grondona, *Slurm: Simple linux utility for resource management*, in *Job Scheduling Strategies for Parallel Processing*, D. Feitelson, L. Rudolph and U. Schwiegelshohn, eds., (Berlin, Heidelberg), pp. 44–60, Springer Berlin Heidelberg, 2003.