# VegasFlow: accelerating Monte Carlo simulation across multiple hardware platforms

Stefano Carrazza*, Juan M. Cruz-Martinez

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

**Abstract**

We present VegasFlow , a new software for fast evaluation of high dimensional integrals based on Monte Carlo integration techniques designed for platforms with hardware accelerators. The growing complexity of calculations and simulations in many areas of science have been accompanied by advances in the computational tools which have helped their developments. VegasFlow enables developers to delegate all complicated aspects of hardware or platform implementation to the library so they can focus on the problem at hand. This software is inspired on the Vegas algorithm, ubiquitous in the particle physics community as the driver of cross section integration, and based on Google's powerful TensorFlow library. We benchmark the performance of this library on many different consumer and professional grade GPUs and CPUs.

*Keywords:* Monte Carlo, Graphs, Integration, Machine Learning, Hardware acceleration

## PROGRAM SUMMARY

*Program Title:* VegasFlow

*Program URL:* https://github.com/N3PDF/vegasflow

*Licensing provisions:* GPLv3

*Programming language:* Python

*Nature of problem:* The solution of high dimensional integrals require the implementation of Monte Carlo algorithms such as Vegas. Monte Carlo algorithms are known to require long computation times.

*Solution method:* Implementation of the Vegas algorithm using the dataflow graph infrastructure provide by the TensorFlow framework. Extension of the algorithm to take advantage of multi-threading CPU and multi-GPU setups.

## 1. Introduction and motivation

State-of-the-art computations in High Energy Physics (HEP) require computing very complex multi-dimensional integrals numerically, as the analytical result is often not known. Monte Carlo (MC) algorithms are generally the option of choice, be it in the HEP application 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 adaptative grids.

The Vegas algorithm [1, 2] is the main driver for multi-purpose parton level event generation programs based on fixed order calculations such as MCFM[3, 4], NNLOJET [5] and also of 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 computation 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].

In this paper we present the VegasFlow library [9], where the main contribution is a novel implementation of the importance sampling algorithm used in the aforementioned event generation programs able to run both in CPUs and GPUs, enabling further acceleration of complicated integrals. The library is written using the TensorFlow [10] library hence the chosen name: VegasFlow .

With this publication we do not aim to overthrow or dethrone Vegas but rather empowering it even more by enabling the frictionless integration of complicated processes in all kinds of hardware supported by TensorFlow with little to no effort made by the user.

The importance sampling "à la" Vegas is the main al-

*Corresponding author.
*E-mail address:* stefano.carrazza@unimi.it
*Preprint number:* TIF-UNIMI-2020-8

gorithm included in `VegasFlow` but the library is designed such that new algorithms can be easily implemented. We believe this design choice together with the TensorFlow back-end will enable a much faster development cycle towards the much desired goal of a Neural Network-based integration algorithm able to surpass Vegas and further reduce computational costs. This feat is yet to come and the effort is not limited to the HEP community but it is rather multidisciplinary. One such example is the Neural Importance Sampling [11] developed in the context of image rendering whose finding have inspired new research in the field of particle physics [12, 13, 14].

## 2. Technical Implementation

The goal of this manuscript is to present a novel open-source library for Monte Carlo integration which takes advantage from hardware accelerators such as GPUs, lowering the barrier in terms of computational knowledge from the user point of view. Our motivation is primarily technical as until now there are no public available libraries which provide such features and thus we think that the scientific community may benefit from a practical implementation. Our aim is for `VegasFlow` to set a new implementation standard for future Monte Carlo calculations.

### *2.1. Acceleration paradigm*

Hardware acceleration combines the flexibility of general-purpose processors, such as CPUs, with the efficiency of fully customized hardware, such as GPUs, ASICs and FPGAs, increasing efficiency by orders of magnitude. In particular, hardware accelerators such as GPUs with large number of cores and memory are getting popular thanks to its great efficiency in deep learning applications through open-source frameworks such as TensorFlow which simplifies the development strategy by reducing the required hardware knowledge from the developer point of view. In this context, `VegasFlow` implements for the first time a Monte Carlo integration produce using TensorFlow primitives together with job scheduling for multi-GPU synchronization. The choice of TensorFlow as the back-end development framework for `VegasFlow` is motivated by its simple mechanism to write efficient python code which can be distributed to hardware accelerators without complicated installation procedures.

### *2.2. Integration algorithms*

The main algorithm in VegasFlow is importance sampling as implemented in Vegas [1, 2], hence the name chosen for the library.

Nonetheless, the library aims to be a general purpose MonteCarlo library. We provide a `MonteCarloFlow` class from which the developer can inherit in order to construct a custom integrator algorithm. The developer has to worry just about what the integrator should do for every particular event (for instance, how to generate the random numbers) and what to do after an integration is finished (for instance, refine how the random numbers are generated). All other technicalities, GPU distribution, multithreading or vectorization of the computation will be dealt with by the library.

### *2.3. Integrands*

For a better integration with VegasFlow, integrands should be written with TensorFlow primitives in python. Written operations using TensorFlow operators allows for the usage of all the hardware TensorFlow is compatible with. The library, however, is not limited and can run integrands written in Fortran, C/C++ or even CUDA [15] through the CFFI library [1]. Alternatively, both C++ and CUDA integrands can be easily linked as TensorFlow operators. The `VegasFlow` package available in [9] contains some examples in the source code.

Features such as exporting histograms during integration can also be implemented and some examples are packaged with the source code.

## 3. Benchmark

### *3.1. Toy integrands*

As a first test and benchmark of `VegasFlow` we use several toy models for which the analytical solution is known. We start by using a spherically symmetric Gaussian as it was also the first example shown in the original Vegas paper [1].

$$I_n = \mathcal{N} \exp\left[-\frac{1}{a^2}\sum_{i=1}^{n}\left(x_i - \frac{1}{2}\right)^2\right]. \tag{1}$$

We also implement some of the integrands proposed by Genz as a test of multidimensional integration algorithms [17], in particular the Genz discontinuous function

$$I_n = \begin{cases} 0 & \text{If any } x_i \leq a_i \\ \mathcal{N}\exp\left(\sum_{i=1}^{n} x_i c_i\right) & \text{otherwise} \end{cases} \tag{2}$$

and the Genz product peak function

$$I_n = \mathcal{N}\prod_{i=1}^{n}\frac{1}{c_i^{-2} + (x_i - a_i)^2}. \tag{3}$$

In all cases the factor $\mathcal{N}$ normalizes the integrand such that it integrates exactly to one. The number of dimensions is set by $n$ and the difficulty of the integration increases with $c_i$.

[1] `https://github.com/cffi/cffi`

| Integrand | Plain MC | Vegas | VegasFlow CPU | VegasFlow GPU |
|---|---|---|---|---|
| SymGauss 8-d | 0.99 ±0.08 | 1.00002 ±0.00023 (18.7s) | 1.00005 ±0.00018 (9.87s) | 1.00008 ±0.00016 (6.21s) |
| SymGauss 20-d | - | 1.00003 ±0.00002 (38min) | 1.00002 ±0.00005 (26min) | 1.00003 ±0.00003 (5min) |
| Genz Eq.(2) 16-d | - | 0.99992 ±0.00008 (1004s) | 1.00010±0.00011 (609s) | 0.99998 ±0.00009 (86s) |
| Genz Eq.(3) 16-d | - | 0.99996 ±0.00011 (1086s) | 1.00013 ±0.00010 (468s) | 1.00026 ±0.00020 (92s) |

Table 1: Comparison of `VegasFlow` with other MC implementations. The number of events per iteration is constant for all integrators for a given integrand. The Plain MC is able to get results in a reasonable amount of time only for the Symmetric Gaussian function in 8 dimensions. The same feature is observed for all integrands where the GPU run of `VegasFlow` achieves the final result much faster than its CPU or Vegas [16] counterparts. Note that the choice of parameters (*e.g.* number of subdivisions of the grid) are arbitrarily chosen for the purposes of this benchmark to be the same across implementations but are not necessarily the optimal choices.

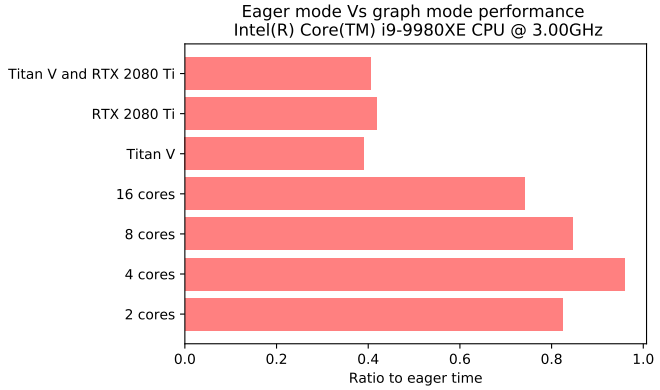


Figure 1: Comparison of performance between the eager and graph compilation TensorFlow mode. The results are shown as a ratio of the time it took the eager computation to complete one iteration. We find comparable (albeit improved) results when running the compiled graph in only CPU mode but a 3x improvement when running on the GPUs.

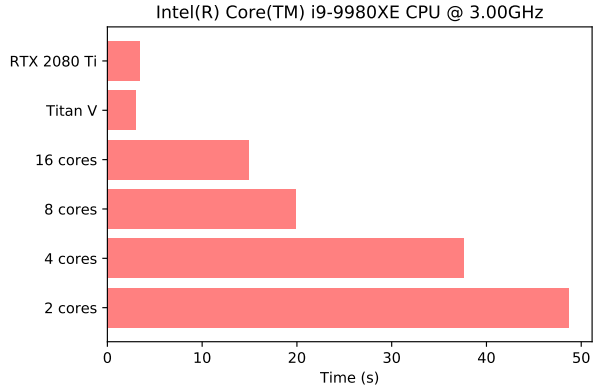


Figure 2: `VegasFlow` running the same integration in 2, 4, 8, 16 threads in a CPU and in the RTX 2080 Ti and Titan V GPUs. Less is better.

### *3.2. Eager mode vs graph mode*

In TensorFlow 2 the so called eager execution was introduced as the default behaviour. Eager execution implements the imperative programming paradigm into TensorFlow, and as a consequence, statements are executed in place instead of building a graph that is subsequently executed later in the program. In this mode, the development and debugging is simplified in exchange for an expected decreased performance.

In order to quantify the performance hit of the eager mode in comparison to the graph mode we run the same integration in both modes and show the results in Fig. 1. In such figure we compare the results of a professional-grade CPU (Intel i9-9980XE) with a consumer-grade GPU (NVIDIA RTX 2080 Ti) and a professional-grade GPU (NVIDIA Titan V). We find the greater improvement with respect to eager execution is found in highly parallel scenarios, such as multi-CPU computation or GPU runs.

It is clear that, whereas eager mode facilitates development, production runs of the code should always be run on graph compiled mode.

### *3.3. Result benchmark*

As a first test we ensure that our integrator produces the correct results for several different integrands. For this we make use of Lepage's python implementation of the Vegas algorithm [16] and a plain MC algorithm with no adaptation.

The results are shown in Table 1. It can be observed that both implementations of importance sampling produce (as one would expect) compatible result. Both Vegas and `VegasFlow` CPU are using all CPUs from an Intel(R) Core(TM) i9-9980XE CPU. In the next section we perform a more detailed benchmark of the running time of different integrators but we can already see a strong improvement due to the usage of the GPU by `VegasFlow` as the computation becomes more complicated.

### *3.4. Performance*

Arguably the main contribution from `VegasFlow` is the ability to use one single implementation across many different devices. The GPUs can take advantage of the huge parallelizability of Monte Carlo simulations reducing the time it takes to finish one computation in an order of magnitude. Furthermore, the reduction is also very apparent on the power consumption of the different devices, indeed, as seen in Table 2, running the same computation is much more slow and expensive when it is run in the CPU. The average power consumption of the CPU is comparable to the Titan V, but with a much longer computational time.

We have also made sure `VegasFlow` can be used in a multi-GPU setting with many different brand and devices. At the moment only distribution on devices within the

| Device | Total Time | Avg. Power Consumption |
|---|---|---|
| i9 (16 cores) | 609s | 85 W |
| RTX 2080 Ti | 93s | 105 W |
| Titan V | 89s | 75 W |

Table 2: Comparison on the power consumption of different devices. The CPU power consumption is provided by the `powertop` utility while the GPU power consumption is a sum of the power draw reported by `nvidia-smi` and `powertop`.

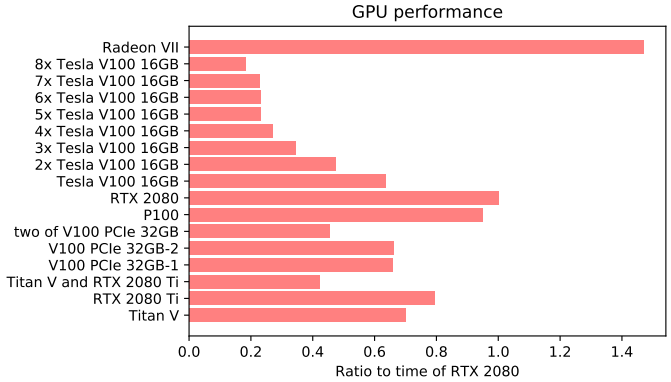


Figure 3: `VegasFlow` running in different GPU devices. We use the consumer-grade RTX 2080 as the measure of performance (less is better).

same physical machine is supported but we plan to implement distribution over different physical machines. We can observe in Fig. 3 how the hierarchy between different GPU devices corresponds to what one would expect from their technical specifications. In Fig. 3 we also observe a good scaling of the speed of the code with the number of GPUs although as the number of GPUs grow we hit a problem of diminishing returns. Similar results are also presented for different CPU models in Fig. 5.

### *3.5. Single t-quark production at leading order*

For the purposes of this benchmark we have considered the calculation of a particle physics process at the partonic level, this is, without considering the convolution with the parton density functions (PDFs). We compare our calculation with the numbers produced by MG5_aMC@NLO [6] for the single $t$-quark production (t-channel) at leading order (LO) [18] using the same physical parameters such as the $t$-quark mass, $m_t = 173.2$ GeV and centre of mass energy $\sqrt{s} = 8$ TeV.

In Fig. 4 we compare the execution time for `VegasFlow` for the single GPU, multi-GPU and multithreading CPU configurations with the equivalent fixed LO order provided by MG5_aMC@NLO 3.0.2. The stopping criteria for the total number of events relies on a target accuracy of $1.4 \cdot 10^{-2}$ pb (with no PDFs). Finally, also in this setup, we observe a great improvement in terms of execution time for the `VegasFlow` approach.

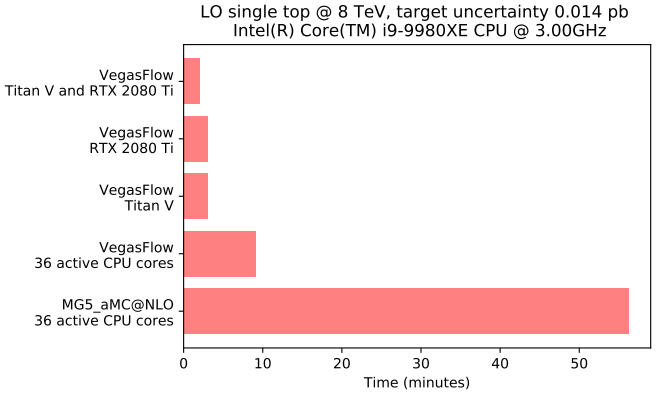


Figure 4: Comparison of a Leading Order calculation ran in both `VegasFlow` and MG5_aMC@NLO [6]. The CPU-only version of `VegasFlow` is able to improve the performance obtained by MG5_aMC@NLO for the same level of target accuracy. The usage of the GPU devices further improves the performance.

## 4. Outlook

We hope this library can accelerate research by granting to users and researchers the ability to implement with simplicity high-dimensional complex integrations without having to know about the technicalities or the difficulties of their implementation on multithreading systems or the data placement and memory management that GPU and multi-GPUs computing requires.

`VegasFlow` is also aimed to developers of new integration methods which can focus on the algorithm technicalities and reduce to a minimum the implementation effort required to adapt the computation into different hardware platforms.

The current release of `VegasFlow` has only been tested in GPUs and CPUs, however we believe that investigation about new hardware accelerators such as Field Programmable Gate Arrays (FPGA) and Tensor Processing Units (TPUs) could provide even more impressive results in terms of performance and power consumption results.

## Acknowledgements

We thank Stefano Forte for a careful reading of the manuscript. We thank Durham University's IPPP for the access to the P100 and V100 32 GB GPUs used in order to benchmark this code. We also acknowledge the NVIDIA Corporation for the donation of a Titan V GPU used for this research. This project is supported by the European Research Council under the European Unions Horizon 2020 research and innovation Programme (grant agreement number 740006) and by the UNIMI Linea2A project "New hardware for HEP".

## Bibliography

[1] G. P. Lepage, A New Algorithm for Adaptive Multidimensional Integration, J. Comput. Phys. 27 (1978) 192. doi:10.1016/0021-9991(78)90004-9.

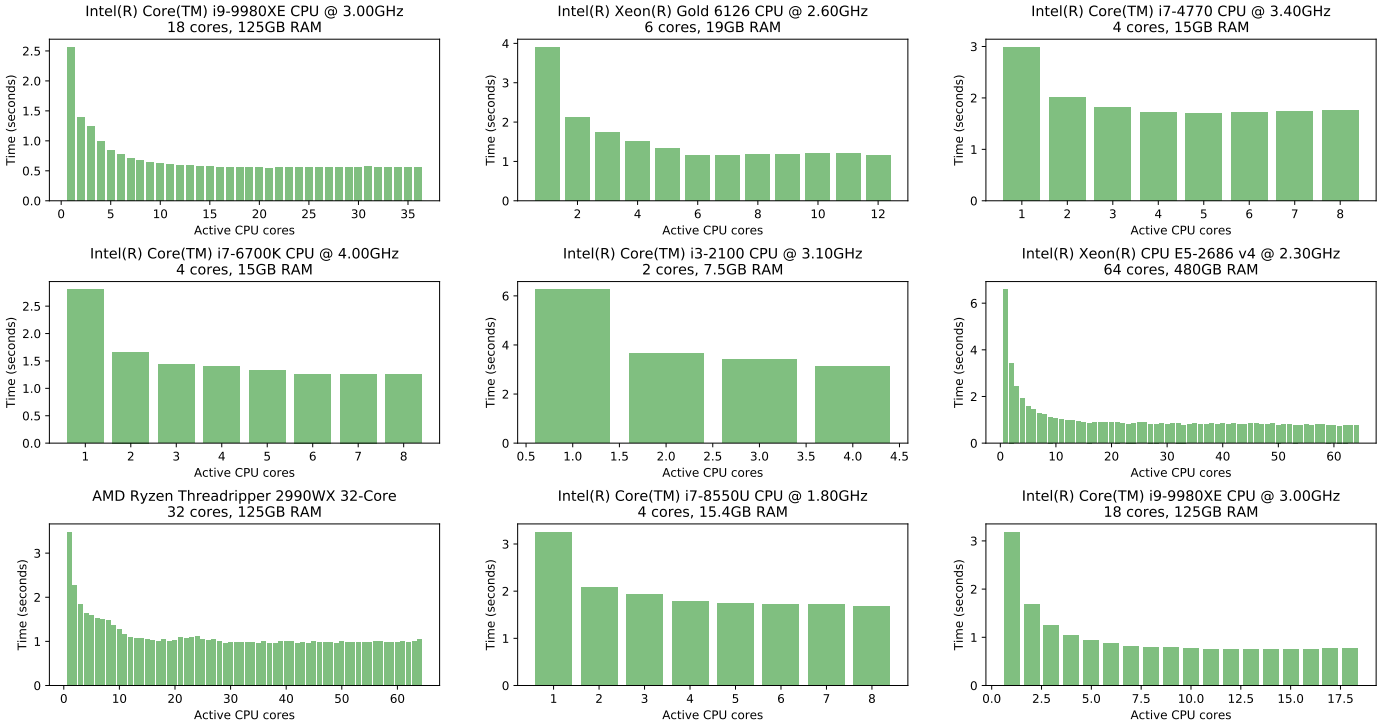


Figure 5: Benchmark of `VegasFlow` on CPU. We observe an improvement of the performance as the number of allowed cores grows until the number of allowed cores is of the same order of the number of physical cores in the machine. The control of the threads for each run was left to TensorFlow with default values, while the binding of the process to the desired number of cores was done with `taskset`.


[2] G. P. Lepage, VEGAS: AN ADAPTIVE MULTIDIMENSIONAL INTEGRATION PROGRAM (1980).
URL https://lib-extopc.kek.jp/preprints/PDF/1980/8006/8006210.pdf

[3] J. M. Campbell, R. K. Ellis, W. T. Giele, A Multi-Threaded Version of MCFM, Eur. Phys. J. C75 (6) (2015) 246. arXiv:1503.06182, doi:10.1140/epjc/s10052-015-3461-2.

[4] J. Campbell, T. Neumann, Precision Phenomenology with MCFM, JHEP 12 (2019) 034. arXiv:1909.09117, doi:10.1007/JHEP12(2019)034.

[5] T. Gehrmann, et al., Jet cross sections and transverse momentum distributions with NNLOJET, PoS RADCOR2017 (2018) 074. arXiv:1801.06415, doi:10.22323/1.290.0074.

[6] J. Alwall, R. Frederix, S. Frixione, V. Hirschi, F. Maltoni, O. Mattelaer, H. S. Shao, T. Stelzer, P. Torrielli, M. Zaro, 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. arXiv:1405.0301, doi:10.1007/JHEP07(2014)079.

[7] T. Gleisberg, S. Hoeche, F. Krauss, M. Schonherr, S. Schumann, F. Siegert, J. Winter, Event generation with SHERPA 1.1, JHEP 02 (2009) 007. arXiv:0811.4622, doi:10.1088/1126-6708/2009/02/007.

[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. arXiv:1908.00167.

[9] J. Cruz-Martinez, S. Carrazza, N3pdf/vegasflow (Feb. 2020). doi:10.5281/zenodo.3691926.
URL https://doi.org/10.5281/zenodo.3691926

[10] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, TensorFlow: Large-scale machine learning on heterogeneous systems, software available from tensorflow.org (2015).
URL http://tensorflow.org/

[11] T. Müller, B. McWilliams, F. Rousselle, M. Gross, J. Novák, Neural importance sampling, CoRR abs/1808.03856. arXiv:1808.03856.
URL http://arxiv.org/abs/1808.03856

[12] C. Gao, J. Isaacson, C. Krause, i-flow: High-Dimensional Integration and Sampling with Normalizing FlowsarXiv:2001.05486.

[13] C. Gao, S. Hoeche, J. Isaacson, C. Krause, H. Schulz, Event Generation with Normalizing FlowsarXiv:2001.10028.

[14] E. Bothmann, T. Janen, M. Knobbe, T. Schmale, S. Schumann, Exploring phase space with Neural Importance SamplingarXiv:2001.05478.

[15] J. Nickolls, I. Buck, M. Garland, K. Skadron, Scalable parallel programming with cuda, Queue 6 (2) (2008) 40–53.

[16] P. Lepage, gplepage/vegas: vegas version 3.4.2 (Feb. 2020). doi:10.5281/zenodo.592154.
URL https://doi.org/10.5281/zenodo.592154

[17] A. Genz, Testing multidimensional integration routines, in: Proc. of International Conference on Tools, Methods and Languages for Scientific and Engineering Computation, Elsevier North-Holland, Inc., USA, 1984, p. 8194.

[18] M. Brucherseifer, F. Caola, K. Melnikov, On the NNLO QCD corrections to single-top production at the LHC, Phys. Lett. B736 (2014) 58–63. arXiv:1404.7116, doi:10.1016/j.physletb.2014.06.075.