PDFFlow: hardware accelerating parton density access

Marco Rossi,^{a,b,*} **Stefano Carrazza**^{a} **and Juan M. Cruz-Martinez**^{a}

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

^{b}*CERN openlab, Geneva 23, CH-1211, Switzerland*
E-mail: stefano.carrazza@mi.infn.it, juan.cruz@mi.infn.it,
marco.rossi5@unimi.it

We present PDFFlow, a new software for fast evaluation of parton distribution functions (PDFs) designed for platforms with hardware accelerators. PDFs are essential for the calculation of particle physics observables through Monte Carlo simulation techniques. The evaluation of a generic set of PDFs for quarks and gluons at a given momentum fraction and energy scale requires the implementation of interpolation algorithms as introduced for the first time by the LHAPDF project. PDFFlow extends and implements these interpolation algorithms using Google's TensorFlow library providing the possibility to perform PDF evaluations taking fully advantage of multi-threading CPU and GPU setups. We benchmark the performance of this library on multiple scenarios relevant for the particle physics community.

---

[*]Speaker

1. Introduction and motivation

Parton Distribution Functions (PDFs) are a core concept in High Energy Physics (HEP) phenomenology: they provide a description of the parton content of the proton and enter the computation of physical observables, like cross sections and differential distributions. PDFs are provided by fitting collaborations, each following different techniques and methodologies [1–3]. Nonetheless, a standard format [4] was agreed to be used by the community: PDF measurements are presented in grids of values as functions of momentum fraction x of a parton with flavor a and energy scale Q; collections of such grids are usually called PDF sets.

The state of the art tool that allows access to PDF values is represented by the LHAPDF library [5], whose algorithms ask as input a single query point in the (a, x, Q) space and output the corresponding interpolated value from the PDF grid. We identify the intrinsic sequential nature of the LHAPDF tool as the entry point of our research development: physical calculations are usually performed via Monte Carlo (MC) simulations, see section 3.1, asking to produce a huge number of independent phase space points; then, parallelizing such uncorrelated computations, may lead to massive performance improvements, like in [6].

Moreover, in recent years the decreasing costs of hardware accelerators, such as GPUs, TPUs and FPGAs, triggered a raising interest in the development of software and frameworks that target these platforms to reduce the computational burden of HEP phenomenological simulations. `VegasFlow` [7, 8] is an excellent example of such approach: written in python, it leverages the TensorFlow [9] framework to delegate the placement and management of tensors on hardware, allowing the developer and the user to focus on the actual implementation of the tool, rather than on the difficulties that may arise from programming directly into hardware specific languages.

Following the strategy introduced by `VegasFlow`, we present the `PDFFlow` library [10], where the main contribution is a novel implementation of the PDF and α_s interpolation algorithm used in LHAPDF. Able to run both in CPUs and GPUs, it enables further acceleration for Monte Carlo simulations. This paper is intended to reach users with a detailed review of `PDFFlow` usage and examples of real physical use cases, along with a presentation of main benchmark results. The paper is structured as follows: in section 2 we discuss technical implementation of the tool and its comparison with the state of the art; section 3 overviews `PDFFlow` and `VegasFlow` integration into real LHC physics simulations, sketching the idea that drives this computation and presenting two complete examples; finally, section 4 wraps up our considerations and future work directions.

2. `PDFFlow`

2.1 Technical Implementation

The idea behind `PDFFlow` implementation is to mimic the LHAPDF interpolation algorithms for PDFs evaluation, trying to parallelize as much as possible the computation and exploiting TensorFlow library to be hardware agnostic, in order to benefit from running on a wide spectrum of modern hardware. `PDFFlow` then, as opposite to LHAPDF, groups together multiple points calculations, asking as inputs three arrays: a set of flavors $\{a\}$ and two arrays of the same length, namely the fractions of momenta x_i and energy scales Q_i respectively. The output of such query is an array of PDF interpolated points f_i.

```
from pdfflow import mkPDF
pdf = mkPDF(f"{pdfset}/0")
pdf.trace() # pdf initializer

pid = [-1,21,1] # anti-down, gluon, down flavors
x = [10**i for i in range(-6,-1)] # momenta fractions
q = [10**i for i in range(1,6)] # energy scales

f = pdf.py_xfxQ(pid, x, q)


##########################################################


from pdfflow import mkPDF
pdf = mkPDF(f"{pdfset}/0")
pdf.alphas_trace() # pdf initializer

q = [10**i for i in range(1,6)] # energy scales

alphas = pdf.py_alphasQ(q)
```


Figure 1: Interpolation example code. *Top*: PDF interpolation. *Bottom*: α_s interpolation. It's good practice to call the `trace` or `alphas_trace` initializers the first time a PDF is created.
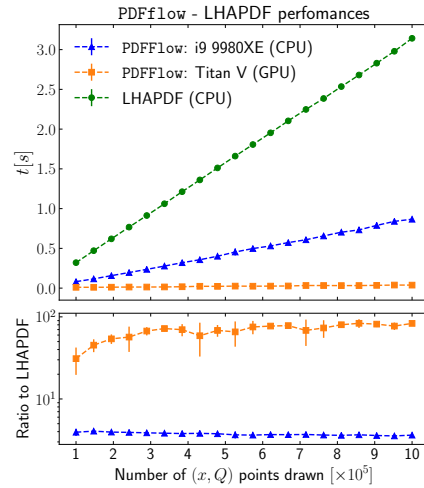
Figure 2: Performance benchmark. *Top*: absolute execution time as a function of the query array length. *Bottom*: time ratio between `PDFFlow` running on different platforms and LHAPDF. Note LHAPDF is single thread, while `PDFFlow` exploits all available cores.

PDFFlow provides a user friendly python interface, where a PDF object is istantiated via a `mkPDF` function, while the actual interpolation is called by the `py_xfxQ` PDF attribute method. Figure 1 shows how a `PDFFlow` user may start an interpolation in their code.[1] Analogous to the PDF interpolation problem is the running of the strong coupling α_s one. Like LHAPDF, we implement this algorithm, whose usage is reported also in the figure.

2.2 Benchmarks

We present here benchmark results between `PDFFlow v1.0` and `LHAPDF v6.3.0` libraries. While we refer to [11, section 3.2] for the accuracy performance, claiming a perfect agreement between the two tools outputs, here we focus on the performance comparison. In figure 2 we show the execution time as a function of the query array length. Different curves correspond to different hardware setups: `PDFFlow` provides a good speed up while running on an Intel i9 CPU (blue curve), showing a flat ratio factor varying between 3x and 4x. The behavior dramatically improves when testing our tool on a Titan V GPU (orange curve), achieving a best improvement of two orders of magnitude when we query 10^6 input points. The higher number of cores that compute parallel operations in a GPU provides tremendous improvement, even with a non-GPU specifically optimized code.

3. `VegasFlow-PDFFlow` usage and examples

3.1 Monte Carlo simulations use cases

Monte Carlo simulations represent a natural framework where a parallelized algorithm can leverage all its capabilities, given that calculations involving different integration points are com-

---

[1]A complete *"How to"* documentation is available at the N3PDF/pdfflow repository.

pletely independent from each other. In this section we show how to exploit `PDFFlow` and `VegasFlow` to compute cross sections at hadron colliders. The problem can be cast into the following form:

$$\sigma_X = \int dx_1 dx_2 \frac{\Phi}{F} \sum_{a,b} \left(\mathcal{L}_{a,b} |\mathcal{M}_{a,b\to X}|^2 \right) \quad (1)$$

where the luminosity $\mathcal{L}$ accounts for the PDFs, the matrix element $\mathcal{M}$ describes the hard interaction, while F and Φ are the flux and phase space factors, respectively. This integral can be evaluated numerically exploiting `VegasFlow` and `PDFFlow` tools. The general implementation scheme is presented in figure 3.

The user has to define an `Integrand` function that will be called by `VegasFlow` at integration time, returning the process cross section and its uncertainty. This function encodes the process-specific physical factors we highlighted in equation (1): it must take as input an array of random numbers, provided by `VegasFlow`, and output the function to be integrated, evaluated at
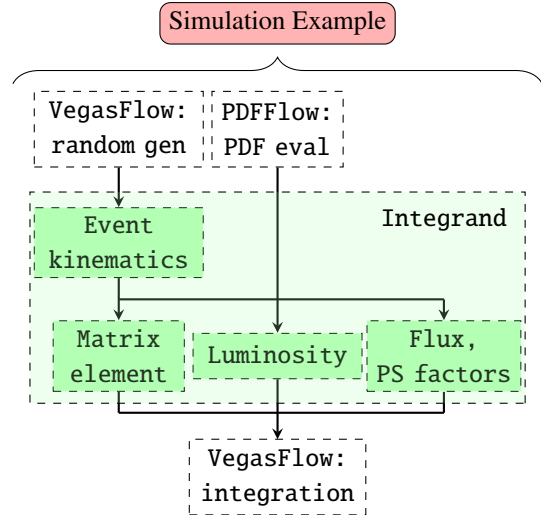

Figure 3: `PDFFlow` flowchart. Blocks are color-coded as following: white for tools implemented algorithms, green for required example-dependent user-defined functions.

those points. In order to build this function, the user must convert the plain random numbers into physical variables, like $x_{1,2}$, that describe the event kinematics. Note that, in this step, the jacobian factor of the trasformation must be taken into account to obtain the correct result. After having reconstructed the kinematics, one must compute the matrix element plus the flux and phase space factors. These ingredients in turn must be combined with the proper luminosity function, computed evaluating the PDFs at the correct (x, Q^2) points by means of `PDFFlow xfxQ2`.

3.2 Single top @ LO

As a basic example, we consider the single-top production at leading order (LO) in proton-proton collisions through the exchange of a W boson in the $t-$channel. Figure 4 instructs the fundamental steps required to build the example code[2].

In figure 5 we show the performance comparison between `VegasFlow`-`PDFFlow`, running on multiple hardware setups, against fixed order calculation at LO with MG5_aMC@NLO [12] software. We stop both the integrations when the total number of generated events allows to achieve a precision better than $2 \cdot 10^{-3}$pb (relative error of $4 \cdot 10^{-5}$) on the output cross section. The bar chart underlines how our parallelized approach grants a great speed up against the baseline algorithm. The picture is even more extreme when running on GPU.

---

[2]The entire code is publicly available inside the N3PDF/pdfflow repository.

```python
import tensorflow as tf
from vegasflow import vegas_wrapper
from pdfflow import mkPDF
...
p = mkPDF(pdfset, DIRNAME) # build PDF
...
@tf.function
def get_x1x2(xarr):
    """ Maps vegas random points to x1,x2
        Returns: partonic center of mass energy,
                 jacobian, x1, x2 """
    ...
    return shat, jac, x1, x2


@tf.function
def make_event(xarr):
    """ Returns the kinematics:
        psw, p0, p1, p2, p3, p4, x1, x2 """
    shat, jac, x1, x2 = get_x1x2(xarr) # transformation
    return build_kinematics(shat, jac, x1, x2)
```

```python
@tf.function
def build_luminosity(x1,x2):
    q2s = tf.ones_like(x1)*mt2 # eval PDFs at top mass squared
    p5x1 = pdf.xfxQ2([5], x1, q2s) # bottom PDFs
    pNx2 = pdf.xfxQ2([2, 4, -1, -3], x2, q2s) # u, c, dx, bx PDFs
    lumis = muliplyPDFs_and sum(p5x1, pNx2)
    return lumis / x1 / x2


@tf.function
def singletop(xarr):
    """ Input: xarr, vegas random points"""
    psw, flux, p0, p1, p2, p3, x1, x2 = make_event(xarr)
    wgts = evaluate_matrix_element_square(p0, p1, p2, p3)
    lumi = build_luminosity(x1,x2)
    lumi_ME = combine_lumi_ME(lumi, wgts)
    return psw * flux * lumi_ME

r = vegas_wrapper(singletop, dim, n_iter, ncalls, compilable=True)
```

Figure 4: Code for single-top production in $t-$channel at LO: `singletop` function evaluates the integrand and is passed as a parameter to `vegas_wrapper` function; PDFFlow enters in the luminosity computation.

3.3 VFH @ NLO

Monte Carlo calculations become exponentially more and more expensive in terms of computational and human efforts at higher orders in perturbation theory. This added complexity provides a perfect benchmarking ground for assessing the GPU acceleration of our models. In particular, we test `VegasFlow-PDFFlow` setup against a simplified version of the existing Fortran 95 NNLO-JET [13] implementation of the vector boson fusion for Higgs production (VFH) at NLO. In this picture we consider only the quark initiated W-boson mediated process, eventually with a gluon emitted from any of the quarks at NLO.

We collect performance results in figure 6, evaluating algorithms on multiple platforms. The stopping criteria for the integrators is set to per-mille precision at LO and percent at NLO in 10 iterations. Although NNLOJET code is highly optimized for CPU and CPU-cluster usage, we observe that our tool, with little to no specific GPU optimization, outperforms the baseline both for LO and NLO accuracy.

4. Conclusions

Porting PDFs to GPU is an essential step in order to accelerate Monte Carlo simulation by granting to the HEP community the ability to implement with simplicity particle physics processes without having to know about the technicalities or the difficulties of their implementation on multi-threading systems or the data placement and memory management that GPU and multi-GPUs computing requires.

PDFFlow is designed to work in synergy with the LHAPDF library, therefore it uses exactly the same PDF data folder structure, and interpolation algorithms for the PDF and α_s determination. While the current release of PDFFlow has only been tested in GPUs and CPUs showing great performance improvements, 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.
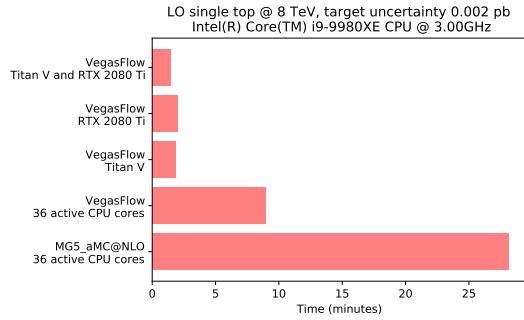
Figure 5: Total integration time comparison between `VegasFlow-PDFFlow` and MG5_aMC@NLO at LO. Software always runs on a single node, eventually exploiting multiple gpus.
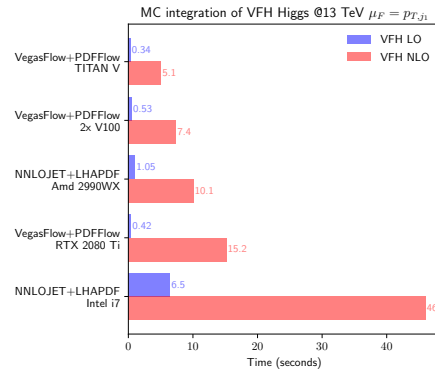

Figure 6: Time per iteration comparison of `VegasFlow-PDFFlow` and NNLOJET at NLO. NNLOJET algorithm allows parallel computing, exploiting all available CPU cores.

Acknowledgments

References

[1] NNPDF collaboration, *Parton distributions from high-precision collider data*, *Eur. Phys. J.* **C77** (2017) 663 [1706.00428].

[2] T.-J. Hou et al., *New CTEQ Global Analysis with High Precision Data from the LHC*, 1908.11238.

[3] L. A. Harland-Lang, A. D. Martin, P. Motylinski and R. S. Thorne, *Parton distributions in the LHC era: MMHT 2014 PDFs*, *Eur. Phys. J.* **C75** (2015) 204 [1412.3989].

[4] M. Whalley, D. Bourilkov and R. Group, *The Les Houches accord PDFs (LHAPDF) and LHAGLUE*, in *HERA and the LHC: A Workshop on the Implications of HERA and LHC Physics (Startup Meeting, CERN, 26-27 March 2004; Midterm Meeting, CERN, 11-13 October 2004)*, pp. 575–581, 8, 2005, hep-ph/0508110.

[5] 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].

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

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

[8] J. Cruz-Martinez and S. Carrazza, *N3pdf/vegasflow*, Feb., 2020. 10.5281/zenodo.3691926.

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

[10] J. Cruz-Martinez, M. Rossi and S. Carrazza, *N3pdf/pdfflow*, July, 2020. 10.5281/zenodo.3964191.

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

[12] 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].

[13] J. Cruz-Martinez, T. Gehrmann, E. Glover and A. Huss, *Second-order QCD effects in Higgs boson production through vector boson fusion*, *Phys. Lett. B* **781** (2018) 672 [1802.02445].