VegasFlow: accelerating Monte Carlo simulation across platforms using TensorFlow

Juan M Cruz-Martinez
in collaboration with: S. Carrazza

comp-ph/2002.12921 10.1016/j.cpc.2020.107376

N3PDF
Machine Learning • PDFs • QCD

40th International Conference on High Energy Physics
Prague - 2020

European Research Council
Established by the European Commission

This project has received funding from the EU's Horizon 2020 research and innovation programme under grant agreement No 740006.

Outline

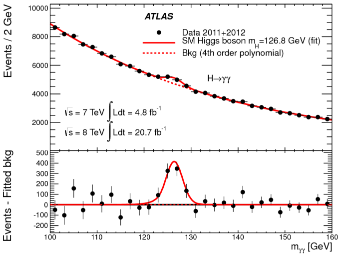
1. Motivation
   - Introduction, hep-ph
   - High Energy (consuming) Physics (phenomenology)
   - How can we do better
2. VegasFlow
   - What is VegasFlow?
   - How to use the code
   - Example of results
3. Conclusions

Parton-level Monte Carlo generators

Behind most predictions for LHC phenomenology lies the numerical computation of the following integral:

$$\int \mathrm{d}x_1\,\mathrm{d}x_2\, f_1(x_1,q^2) f_2(x_2,q^2) |M(\{p_n\})|^2 \mathcal{J}^n_m(\{p_n\})$$

- → $f(x,q)$: Parton Distribution Function
- → $|M|$: Matrix element of the process
- → $\{p_n\}$: Phase space for n particles.
- → $\mathcal{J}$: Jet function for n particles to m.

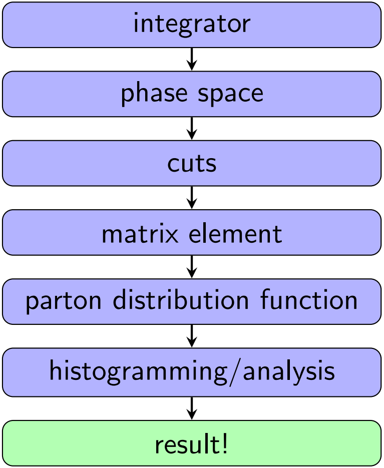


Parton-level Monte Carlo generators ingredients:

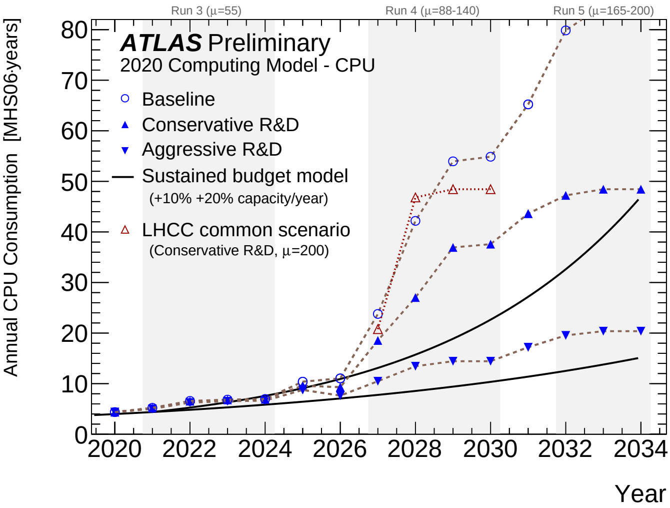
$$\int \mathrm{d}x_1\,\mathrm{d}x_2\, f_1(x_1,q^2) f_2(x_2,q^2) |M(\{p_n\})|^2 \mathcal{J}_m^n(\{p_n\})$$

proton-proton $\rightarrow$ jets

The integrals are usually be computed numerically using CPU-expensive Monte Carlo methods.

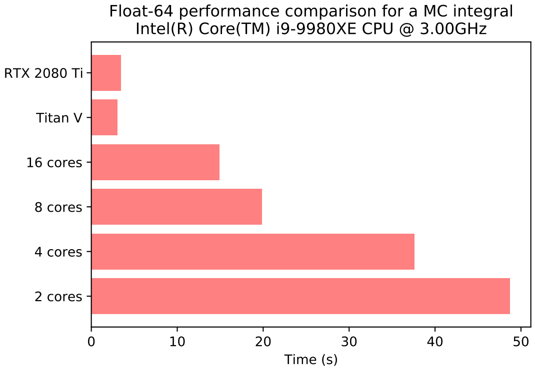


ATLAS projected CPU usage



GPU computing

Monte Carlo simulations are highly parallelizable, which make them a great target for GPU computation.



Monte Carlo integration of a n-dimensional gaussian function

$$I = \int \mathrm{d}x_1 \ldots \mathrm{d}x_n \, e^{x_1^2 + \cdots + x_n^2}$$

GPU computation can increase the performance of the integrator by more than an order of magnitude.

Why is then GPU computing not more widespread?

Most of the more advance phenomenological calculations still rely exclusively on CPU. With only a few libraries providing GPU interfaces such as pySecDec.

✗ Diminishing returns

- Huge CPU-optimized Fortran 77/90 or C++ codebases.
- Publication-ready results are easily obtained expanding existing code.
- It's catch-22: porting the code becomes more and more complicated.

✗ Lack of expertise

- CPU expertise is not necessarily applicable to GPU programming.
- New programming languages: Cuda? OpenCL?
- Low-reward situation when trying to achieve previous performance.

✗ Lack of tools

- Many ready-made tools for CPU.
- GPUs are still decades behind in the hep-ph world.

Why is then GPU computing not more widespread?

Most of the more advance phenomenological calculations still rely exclusively on CPU. With only a few libraries providing GPU interfaces such as pySecDec.

✗ Diminishing returns

- Huge CPU-optimized Fortran 77/90 or C++ codebases.
- Publication-ready results are easily obtained expanding existing code.
- It's catch-22: porting the code becomes more and more complicated.

✗ Lack of expertise

- CPU expertise is not necessarily applicable to GPU programming.
- New programming languages: Cuda? OpenCL?
- Low-reward situation when trying to achieve previous performance.

✗ Lack of tools

- Many ready-made tools for CPU.
- GPUs are still decades behind in the hep-ph world.

Why is then GPU computing not more widespread?

Most of the more advance phenomenological calculations still rely exclusively on CPU. With only a few libraries providing GPU interfaces such as pySecDec.

✗ Diminishing returns
- Huge CPU-optimized Fortran 77/90 or C++ codebases.
- Publication-ready results are easily obtained expanding existing code.
- It's catch-22: porting the code becomes more and more complicated.

✗ Lack of expertise
- CPU expertise is not necessarily applicable to GPU programming.
- New programming languages: Cuda? OpenCL?
- Low-reward situation when trying to achieve previous performance.

✗ Lack of tools
- Many ready-made tools for CPU.
- GPUs are still decades behind in the hep-ph world.

Why is then GPU computing not more widespread?

Most of the more advance phenomenological calculations still rely exclusively on CPU. With only a few libraries providing GPU interfaces such as pySecDec.

✗ Diminishing returns

- Huge CPU-optimized Fortran 77/90 or C++ codebases.
- Publication-ready results are easily obtained expanding existing code.
- It's catch-22: porting the code becomes more and more complicated.

✗ Lack of expertise

- CPU expertise is not necessarily applicable to GPU programming.
- New programming languages: Cuda? OpenCL?
- Low-reward situation when trying to achieve previous performance.

✗ Lack of tools

- Many ready-made tools for CPU.
- GPUs are still decades behind in the hep-ph world.

Why is then GPU computing not more widespread?

Most of the more advance phenomenological calculations still rely exclusively on CPU. With only a few libraries providing GPU interfaces such as pySecDec.

✗ Diminishing returns

- Huge CPU-optimized Fortran 77/90 or C++ codebases.
- Publication-ready results are easily obtained expanding existing code.
- It's catch-22: porting the code becomes more and more complicated.

✗ Lack of expertise

- CPU expertise is not necessarily applicable to GPU programming.
- New programming languages: Cuda? OpenCL?
- Low-reward situation when trying to achieve previous performance.

✗ Lack of tools

- Many ready-made tools for CPU.
- GPUs are still decades behind in the hep-ph world.

Why is then GPU computing not more widespread?

Most of the more advance phenomenological calculations still rely exclusively on CPU. With only a few libraries providing GPU interfaces such as pySecDec.

✗ Diminishing returns

- Huge CPU-optimized Fortran 77/90 or C++ codebases.
- Publication-ready results are easily obtained expanding existing code.
- It's catch-22: porting the code becomes more and more complicated.

✗ Lack of expertise

- CPU expertise is not necessarily applicable to GPU programming.
- New programming languages: Cuda? OpenCL?
- Low-reward situation when trying to achieve previous performance.

✗ Lack of tools

- Many ready-made tools for CPU.
- GPUs are still decades behind in the hep-ph world.

Why is then GPU computing not more widespread?

Most of the more advance phenomenological calculations still rely exclusively on CPU. With only a few libraries providing GPU interfaces such as pySecDec.

✗ Diminishing returns

- Huge CPU-optimized Fortran 77/90 or C++ codebases.
- Publication-ready results are easily obtained expanding existing code.
- It's catch-22: porting the code becomes more and more complicated.

✗ Lack of expertise

- CPU expertise is not necessarily applicable to GPU programming.
- New programming languages: Cuda? OpenCL?
- Low-reward situation when trying to achieve previous performance.

✗ Lack of tools

- Many ready-made tools for CPU.
- GPUs are still decades behind in the hep-ph world.

Why is then GPU computing not more widespread?

Most of the more advance phenomenological calculations still rely exclusively on CPU. With only a few libraries providing GPU interfaces such as pySecDec.

✗ Diminishing returns

- Huge CPU-optimized Fortran 77/90 or C++ codebases.
- Publication-ready results are easily obtained expanding existing code.
- It's catch-22: porting the code becomes more and more complicated.

✗ Lack of expertise

- CPU expertise is not necessarily applicable to GPU programming.
- New programming languages: Cuda? OpenCL?
- Low-reward situation when trying to achieve previous performance.

✗ Lack of tools

- Many ready-made tools for CPU.
- GPUs are still decades behind in the hep-ph world.

Why is then GPU computing not more widespread?

Most of the more advance phenomenological calculations still rely exclusively on CPU. With only a few libraries providing GPU interfaces such as pySecDec.

✗ Diminishing returns
- Huge CPU-optimized Fortran 77/90 or C++ codebases.
- Publication-ready results are easily obtained expanding existing code.
- It's catch-22: porting the code becomes more and more complicated.

✗ Lack of expertise
- CPU expertise is not necessarily applicable to GPU programming.
- New programming languages: Cuda? OpenCL?
- Low-reward situation when trying to achieve previous performance.

✗ Lack of tools
- Many ready-made tools for CPU.
- GPUs are still decades behind in the hep-ph world.

Why is then GPU computing not more widespread?

Most of the more advance phenomenological calculations still rely exclusively on CPU. With only a few libraries providing GPU interfaces such as pySecDec.

✗ Diminishing returns

- Huge CPU-optimized Fortran 77/90 or C++ codebases.
- Publication-ready results are easily obtained expanding existing code.
- It's catch-22: porting the code becomes more and more complicated.

✗ Lack of expertise

- CPU expertise is not necessarily applicable to GPU programming.
- New programming languages: Cuda? OpenCL?
- Low-reward situation when trying to achieve previous performance.

✗ Lack of tools

- Many ready-made tools for CPU.
- GPUs are still decades behind in the hep-ph world.

Why is then GPU computing not more widespread?

Most of the more advance phenomenological calculations still rely exclusively on CPU. With only a few libraries providing GPU interfaces such as pySecDec.

✗ Diminishing returns
  - Huge CPU-optimized Fortran 77/90 or C++ codebases.
  - Publication-ready results are easily obtained expanding existing code.
  - It's catch-22: porting the code becomes more and more complicated.

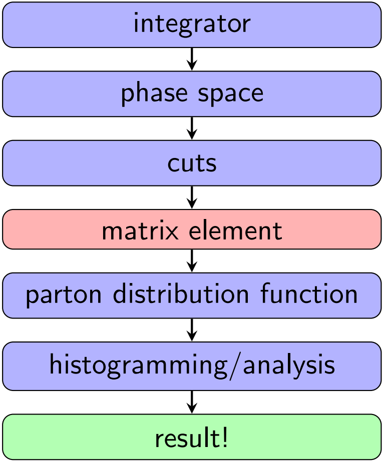
✗ Lack of expertise
  - CPU expertise is not necessarily applicable to GPU programming.
  - New programming languages: Cuda? OpenCL?
  - Low-reward situation when trying to achieve previous performance.

✗ Lack of tools
  - Many ready-made tools for CPU.
  - GPUs are still decades behind in the hep-ph world.

Lack of Tools

Running on a CPU:

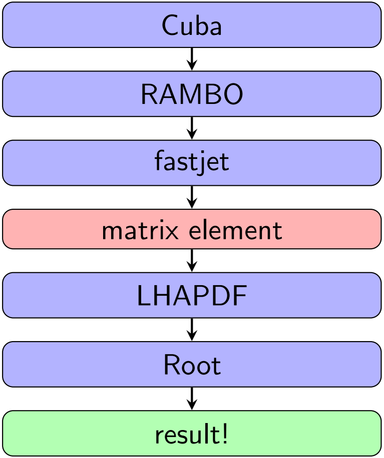
For CPU computation you can focus in the result you are interested in, as there is a complete toolset for producing results.



Lack of Tools

Running on a CPU:

For CPU computation you can focus in the result you are interested in, as there is a complete toolset for producing results.

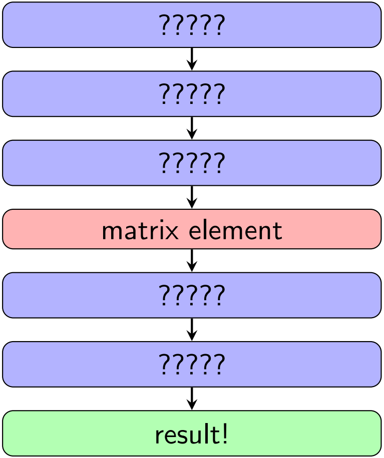


Lack of Tools

Running on a GPU:

For CPU computation you can focus in the result you are interested in, as there is a complete toolset for producing results.

There is still no such complete toolset for GPU computation which means one has to write code from scratch



A new toolset: VegasFlow and PDFflow

We present a Monte Carlo integration library focused on speed, efficiency for the computer and the developer.

- ✓ Python and TensorFlow based engine
- ✓ GPU and CPU compatible out of the box
- ✓ Choose your language: Python, Cuda, C++
- ✓ Seamlessly compatible with NN-based integrators

What about PDFFlow?

Together with VegasFlow we are also working on a implementation of PDF interpolation to run on GPU also based on python+TensorFlow.
To know more, please see the talk tomorrow at 8 AM (CEST) by Marco Rossi.

A new toolset: VegasFlow and PDFflow

We present a Monte Carlo integration library focused on speed, efficiency for the computer and the developer.

- ✓ Python and TensorFlow based engine
- ✓ GPU and CPU compatible out of the box
- ✓ Choose your language: Python, Cuda, C++
- ✓ Seamlessly compatible with NN-based integrators

What about PDFFlow?

Together with VegasFlow we are also working on a implementation of PDF interpolation to run on GPU also based on python+TensorFlow.
To know more, please see the talk tomorrow at 8 AM (CEST) by Marco Rossi.

A new toolset: VegasFlow and PDFflow

We present a Monte Carlo integration library focused on speed, efficiency for the computer and the developer.

- ✓ Python and TensorFlow based engine
- ✓ GPU and CPU compatible out of the box
- ✓ Choose your language: Python, Cuda, C++
- ✓ Seamlessly compatible with NN-based integrators

What about PDFFlow?

Together with VegasFlow we are also working on a implementation of PDF interpolation to run on GPU also based on python+TensorFlow.
To know more, please see the talk tomorrow at 8 AM (CEST) by Marco Rossi.

A new toolset: VegasFlow and PDFflow

We present a Monte Carlo integration library focused on speed, efficiency for the computer and the developer.

- ✓ Python and TensorFlow based engine
- ✓ GPU and CPU compatible out of the box
- ✓ Choose your language: Python, Cuda, C++
- ✓ Seamlessly compatible with NN-based integrators

What about PDFFlow?

Together with VegasFlow we are also working on a implementation of PDF interpolation to run on GPU also based on python+TensorFlow.
To know more, please see the talk tomorrow at 8 AM (CEST) by Marco Rossi.

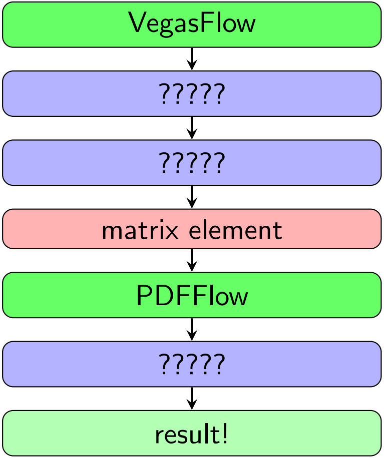
A new tool: VegasFlow

Framework for evaluation of high dimensional integrals based on MC algorithms.

Version 1.0 includes:

- ✓ Plain Monte Carlo: to be used as a template for writing more complicated algorithms.
- ✓ Vegas: importance sampling algorithm by G. Peter Lepage.

Source code available at:
`github.com/N3PDF/VegasFlow`



VegasFlow: open source for HEP

Where to obtain the code

VegasFlow is opensource and available at `github.com:N3PDF/VegasFlow`

How to install

You can install it using either `pip` or `conda`:

```
~$ pip install VegasFlow
~$ conda install VegasFlow
```

Documentation

The documentation for VegasFlow is accessible at: `VegasFlow.rtfd.io`

Run a simple integrand

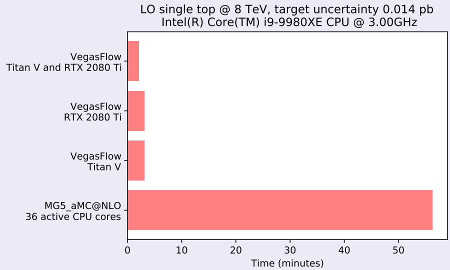
```
>>> @tf.function
>>> def complicated_integrand(xarr, **kwargs):
>>>     return tf.reduce_sum(xarr, axis=1)
>>> from VegasFlow.vflow import VegasFlow
# Instantiate the integrator
# limit the number of events to be computed at once
# (hardware dependent!)
>>> n_dim = 10
>>> n_events = int(1e6)
>>> integrator = VegasFlow(n_dim, n_events, events_limit = int(1e5))
# Register the integrand
>>> integrator.compile(complicated_integrand)
# Run a number of iterations
>>> res = integrator.run_integration(n_iter = 5, log_time = True)


Result for iteration 0:  5.0000 +/- 0.0009(took 0.47029 s)
Result for iteration 1:  5.0006 +/- 0.0003(took 0.32042 s)
                ⋮
Final results:  4.99995 +/- 8.95579e-05
```

VegasFlow Vs Madgraph LO

For Leading Order calculations the advantages are immediately visible

Plain Madgraph Vs C++-like implementation



- We have ported an old fortran code, no GPU-specific optimization.
- Phase Space, spinors, cuts... all done 'the old way"

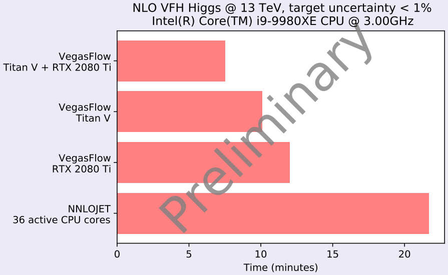
i.e., there's room for improvement by developing GPU-specific code!
What about NLO?

VegasFlow for NLO calculations

Still can't achieve an order of magnitude for NLO. But it is already better!

- Same caveats as before → no GPU-specific optimization
- Proof-of-concept, not a full parton-level MC simulator.

NNLOJET+LHAPDF vs VegasFlow+PDFFlow



Summary

- GPU computation is increasingly gaining traction in many areas of science.
- GPU is a technology not heavily used in particle physics phenomenology.

→ Despite being competitive with CPU for MC simulations.

✓ VegasFlow provides a framework to run in both GPU and CPU.

✓ Can immediately apply existing expertise.

✓ Easily implementation of new-generation NN-based integrators.

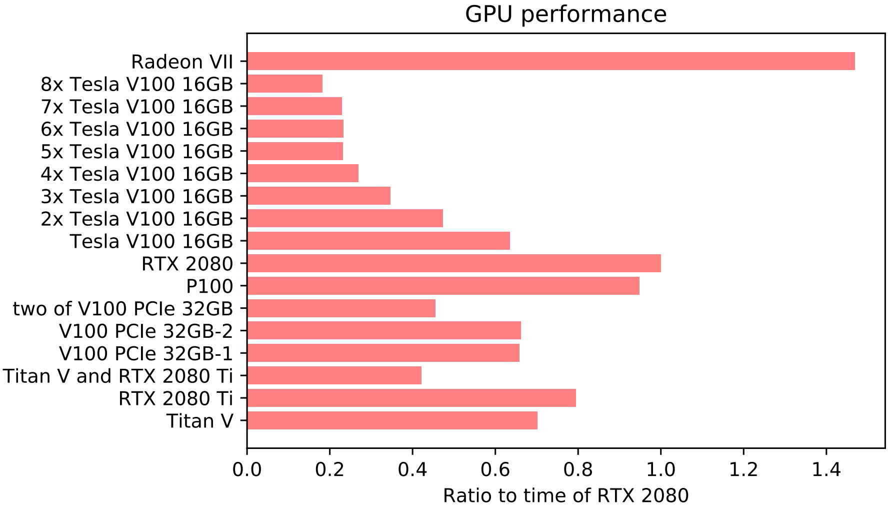
Going forward:

✓ More GPU-ready tools in the works.

✓ Working with other groups to interface VegasFlow with existing tools.

Thanks!

Benchmark on different GPUs



Benchmark on different CPUs