VegasFlow and PDFFlow: accelerating Monte Carlo simulation across platforms

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

PDFFlow: hep-ph/2009.06635 VegasFlow: 10.1016/j.cpc.2020.107376

ATLAS Physics Modelling Group subgroup for Generator Infrastructure and Tools
October 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.

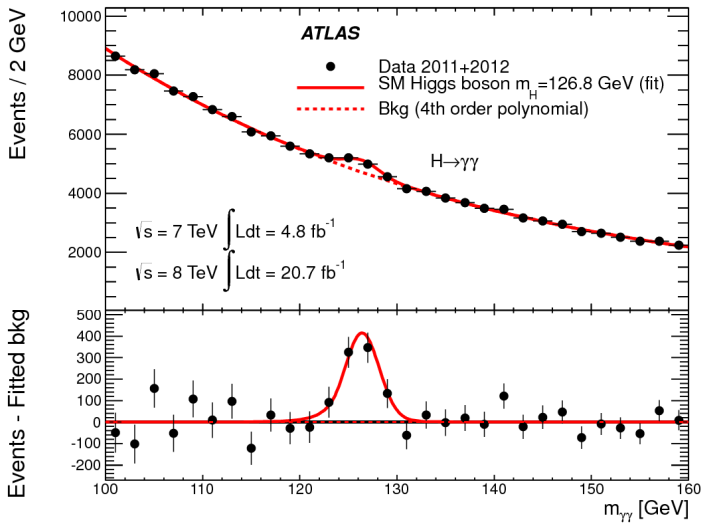
Outline

Parton-level Monte Carlo generators

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

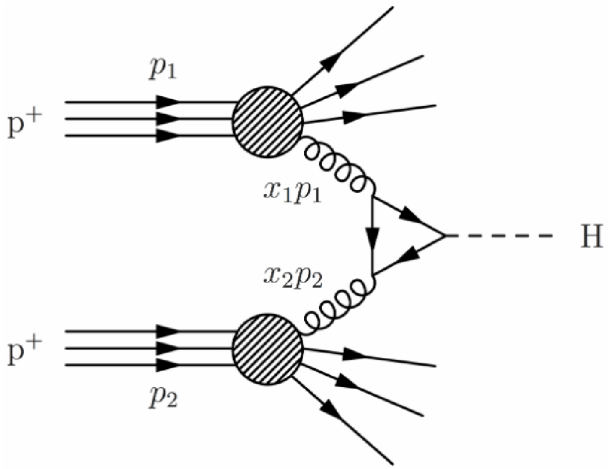
$$\int \mathrm{d}x_1\,\mathrm{d}x_2\, f_1(x_1, q^2) f_2(x_2, q^2) |M(\{p_n\})|^2 \mathcal{J}_m^n(\{p_n\})$$

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

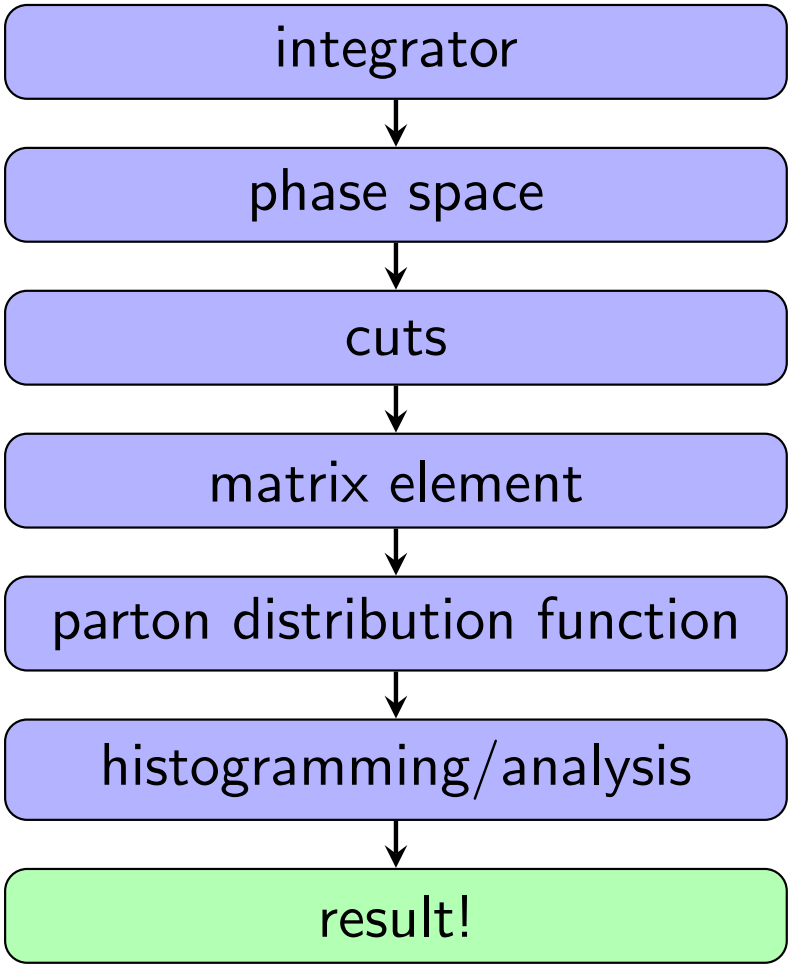


Parton-level Monte Carlo generators ingredients:

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

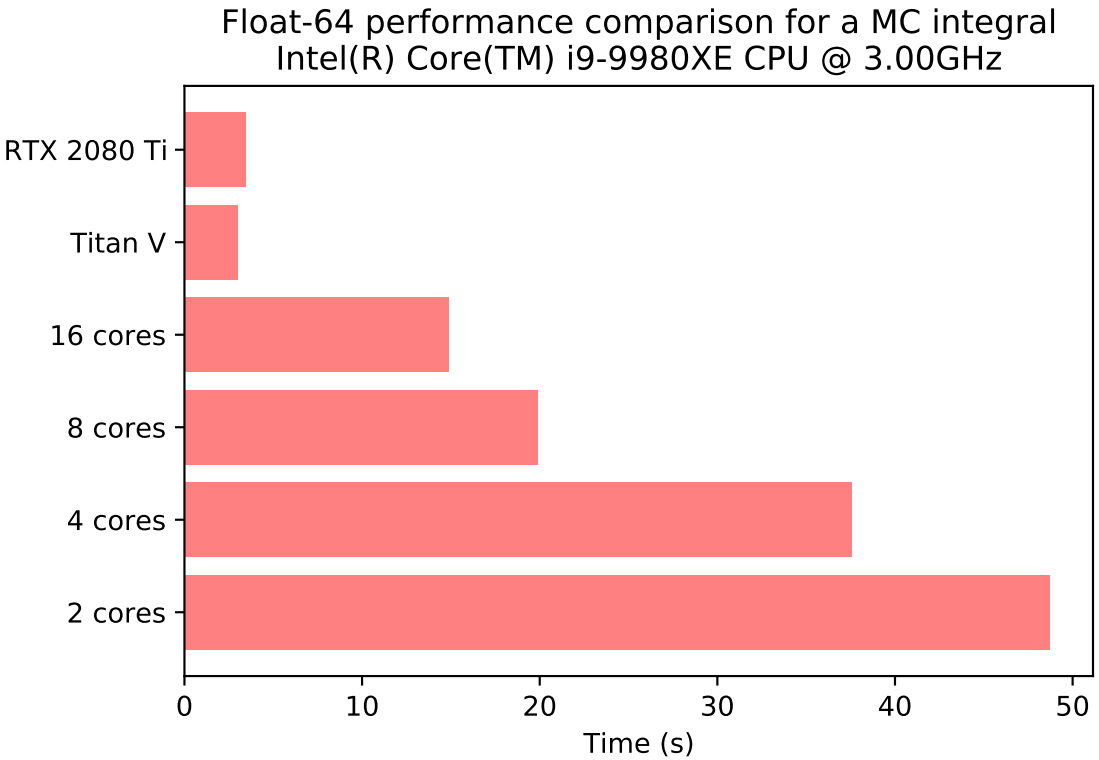


The integrals are usually computed numerically using CPU-expensive Monte Carlo generators.



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

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

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 theoretical 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 theoretical 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 theoretical 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 theoretical 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 theoretical 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 theoretical 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 theoretical 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 theoretical 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 theoretical 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 theoretical 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?

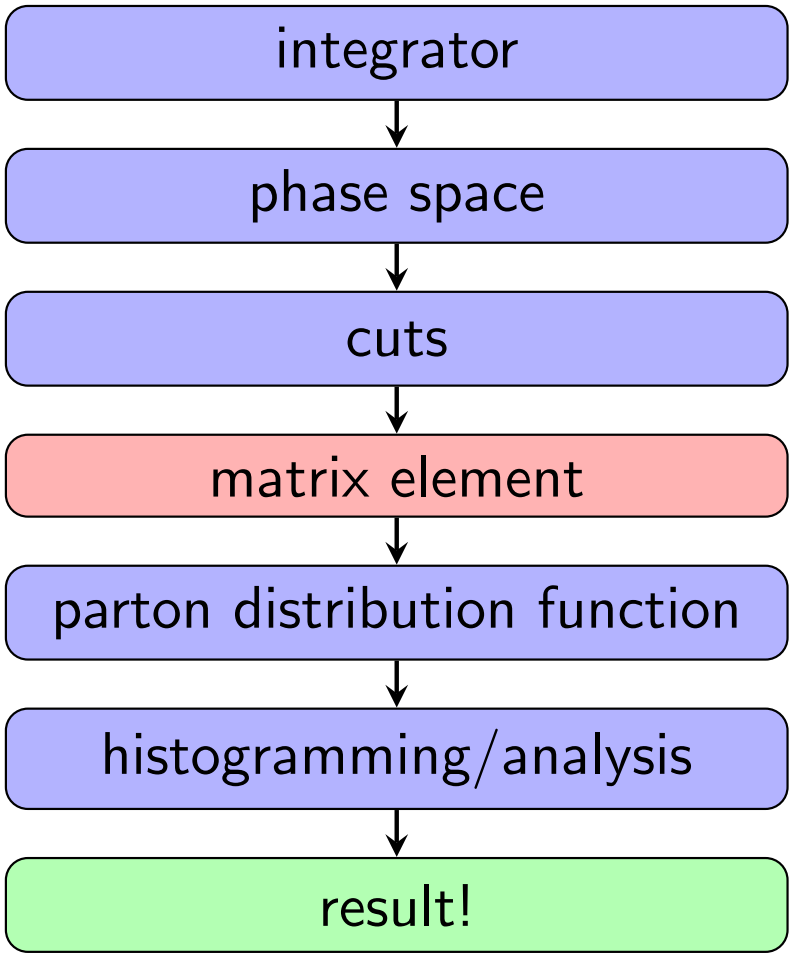
Most of the more advance theoretical calculations still rely exclusively on CPU. With only a few libraries providing GPU interfaces such as pySecDec.

- ✗ 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:

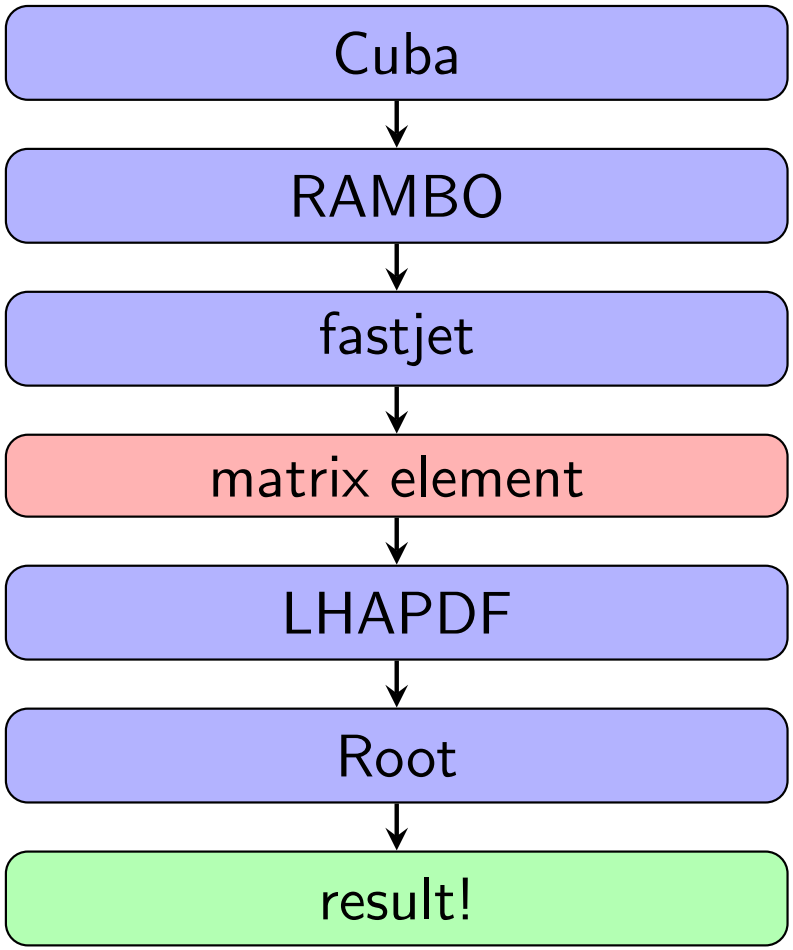
For CPU computation you can focus in the result you are interested in (for instance, the physical process), as there is a complete toolset for producing results.



Lack of Tools

Running on a CPU:

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

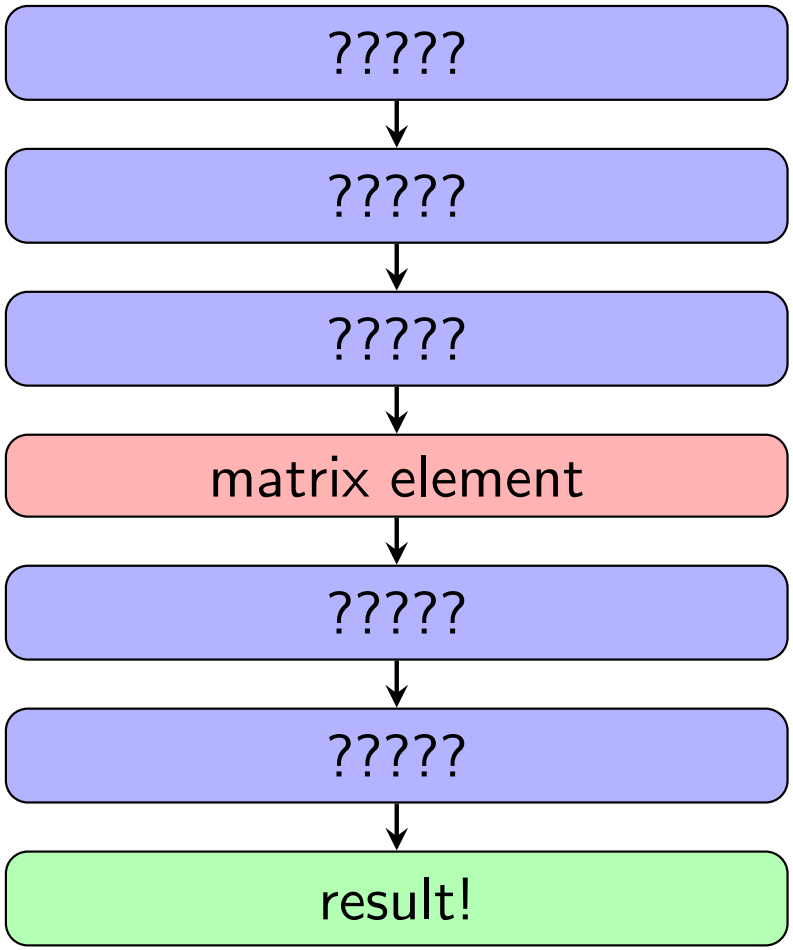


Lack of Tools

Running on a GPU:

For CPU computation you can focus in the result you are interested in (for instance, the physical process), 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

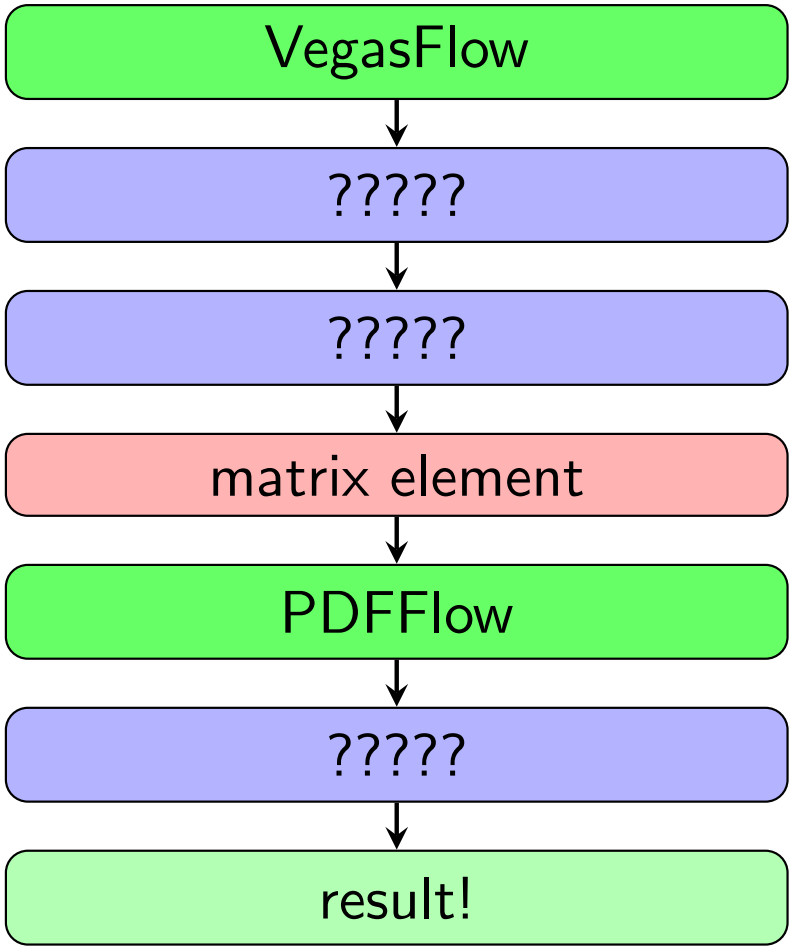


A new toolset: VegasFlow and PDFflow

The pdf and vegas-flow libraries focus on speed and efficiency for both the computer and the developer

- Python and TF based engine
- Compatible with other languages: cuda, c++
- Seamless CPU and GPU computation out of the box
- Easily interfaceable with NN-based integrators

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



Open source for HEP

Where to obtain the code

Both VegasFlow and PDFFlow are open source and can be found at the N3PDF organization repository `github.com:N3PDF`

How to install

Can be installed from the repository or directly with `pip`:

```
~$ pip install vegasflow pdfflow
```

Documentation

The documentation for these tools is accessible at:
VegasFlow: `vegasflow.rtfd.io`
PDFFlow: `pdfflow.rtfd.io`

Summary

- GPU computation is increasingly gaining traction in many areas of science but it's not heavily used in particle physics phenomenology.

→ Despite being competitive with CPU for MC simulations.

✓ VegasFlow and PDFFlow provide a framework to run in any device.

✓ Easy implementation of new-generation or NN-based integration algorithms (**already working on that!**)

Where to obtain the code

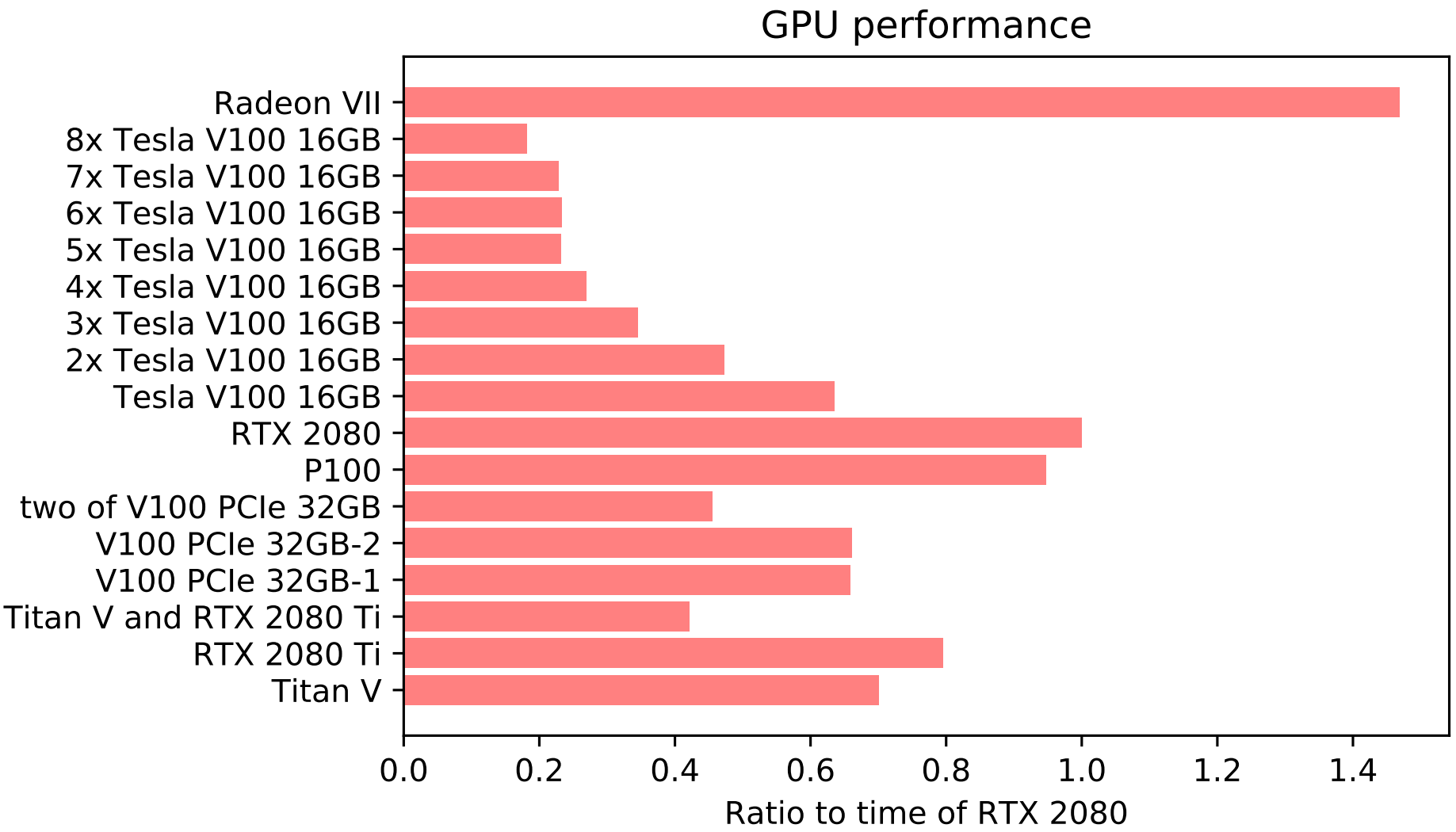
VegasFlow and PDFFlow are opensource and available at `github.com:N3PDF/pdfflow` and `github.com:N3PDF/VegasFlow`

Next:

✓ And now Marco Rossi will tell us about PDFFlow and will show some specific examples.

Thanks!

Benchmark on different GPUs



Benchmark on different CPUs