

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](https://arxiv.org/abs/hep-ph/2009.06635)

VegasFlow: [10.1016/j.cpc.2020.107376](https://arxiv.org/abs/10.1016/j.cpc.2020.107376)



Machine Learning • PDFs • QCD

HSF WLCG Virtual Workshop
November 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

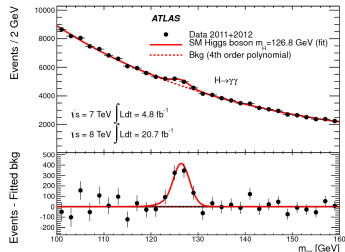
- 1 Motivation
 - Introduction, hep-ph
 - How can we do better
- 2 VegasFlow and PDFFlow
 - What are they?
 - Where to find the code
- 3 Results and extensions
 - PDF interpolation
 - LO and NLO calculations
 - Integration with other software
- 4 Conclusions

Parton-level Monte Carlo generators

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

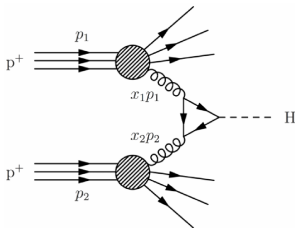
$$\int dx_1 dx_2 f_1(x_1, q^2) f_2(x_2, q^2) |M(\{p_n\})|^2 \mathcal{J}_m^n(\{p_n\})$$

- $f(x, q)$: 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 .

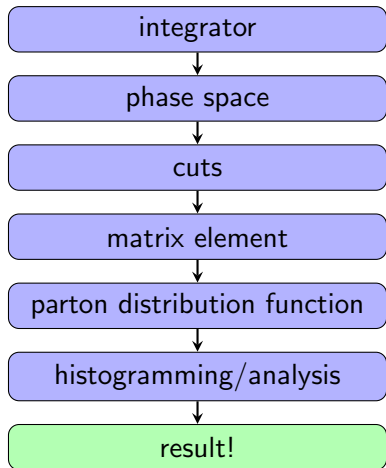


Parton-level Monte Carlo generators ingredients:

$$\int dx_1 dx_2 f_1(x_1, q^2) f_2(x_2, q^2) |M(\{p_n\})|^2 \mathcal{J}_m^n(\{p_n\})$$



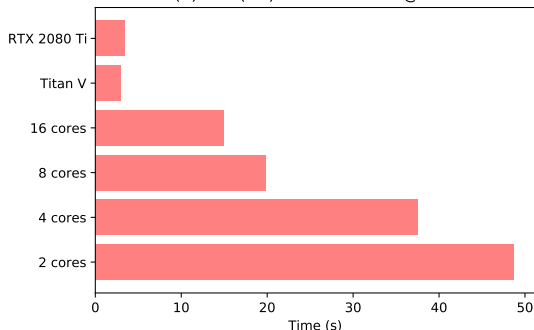
The integrals are usually computed numerically using CPU-expensive Monte Carlo generators.



GPU computing

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

Float-64 performance comparison for a MC integral
Intel(R) Core(TM) i9-9980XE CPU @ 3.00GHz



Monte Carlo integration of a n -dimensional gaussian function

$$I = \int dx_1 \dots dx_n e^{x_1^2 + \dots + x_n^2}$$

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.

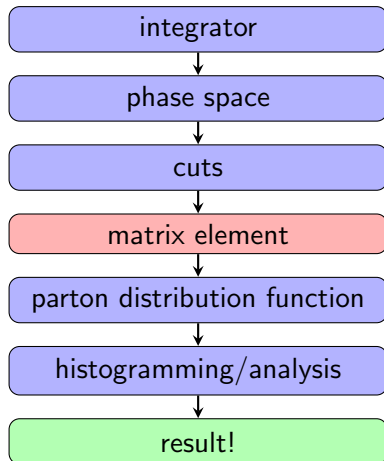
✗ Lack of tools

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

Lack of Tools

Running on a CPU:

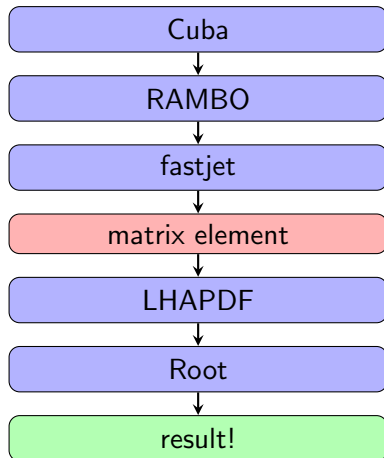
For CPU computation you can focus in the result you are interested in (for instance, the physical process), as there is a complete toolset for producing results.



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.

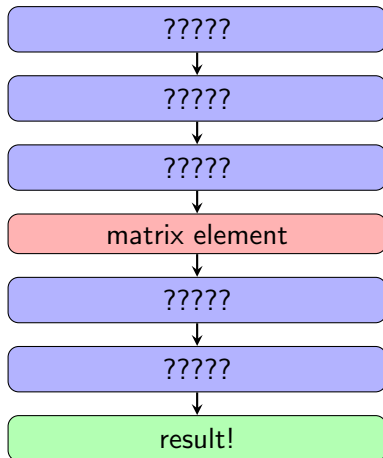


Lack of Tools

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

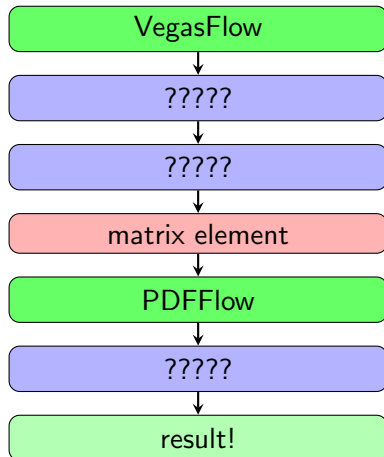
The pdf and vegas-flow libraries focus on speed and efficiency for both the computer and the developer

- 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



Parallelize everything parallelizable: batch what can be batched

As a result, with minimal changes to the algorithms involved, one can achieve a tremendous speed-up.

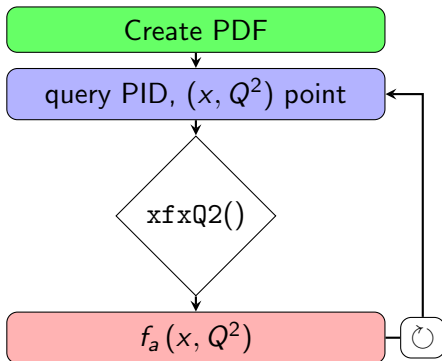


Figure: LHAPDF6

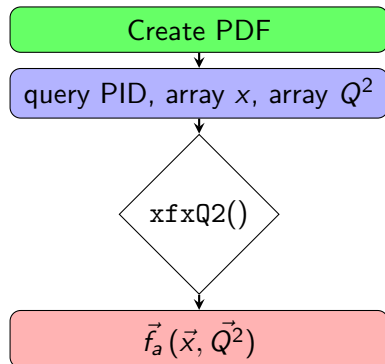


Figure: PDFFlow

Usability status

The goal

The developer writes the code once (for instance, the matrix element for the process they are interested in) and it can automatically be used for both GPU and CPU.

While perfectly possible, it will take some times for the tools to be widely used. In other words, a pheno PhD student starting their $N^3\text{LO}$ calculation today might start producing results in a few years.

The workarounds

We have thus focused on compatibilities with existing code and tools. PDFFlow python and C interfaces follow a structure very similar to LHAPDF while VegasFlow is compatible with integrands written in Cuda, C++ or regular python.

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

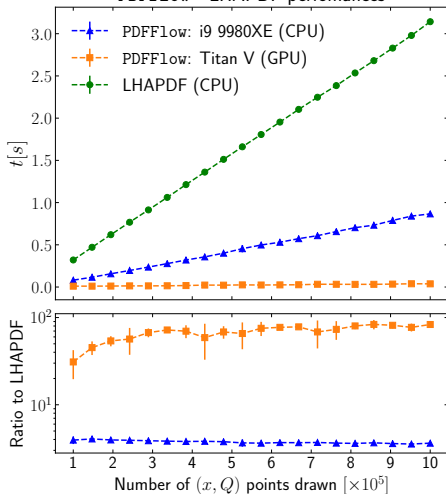
The documentation for these tools is accessible at:

VegasFlow: vegasflow.rtdf.io

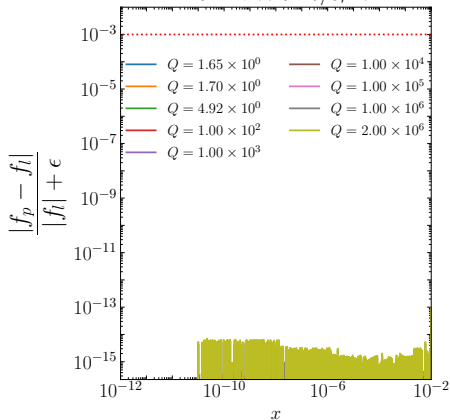
PDFFlow: pdfflow.rtdf.io

LHAPDF vs PDFflow

PDFflow - LHAPDF performances



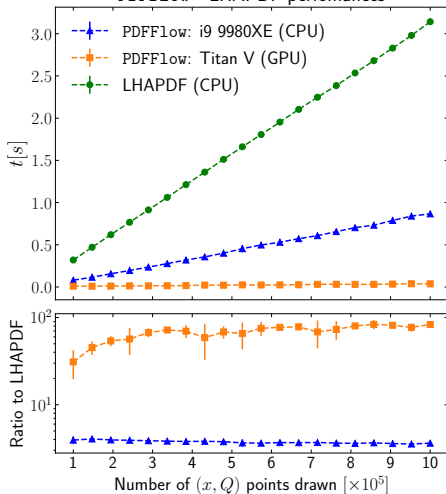
NNPDF31_nlo_as_0118/0, flav = 1



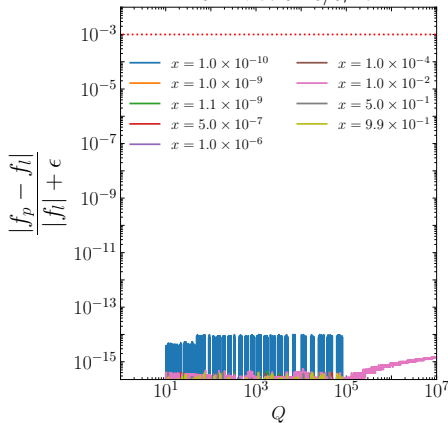
Interpolation in x for fixed q .

LHAPDF vs PDFFlow

PDFflow - LHAPDF performances



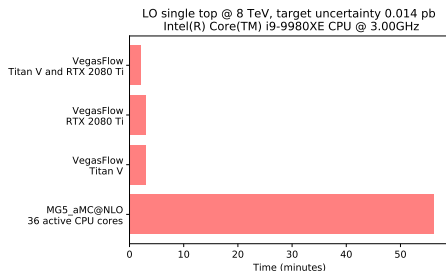
NNPDF31_nlo_as_0118/0, flav = 1



Interpolation in q for fixed x .

VegasFlow Vs Madgraph LO

For Leading Order calculations the advantages are immediately visible



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

Figure: Plain Madgraph Vs C++-like implementation

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 on the phase space, cuts or subtraction terms
- Proof-of-concept, not a full parton-level MC simulator.
- ✓ Great potential for accelerating fixed order calculations.

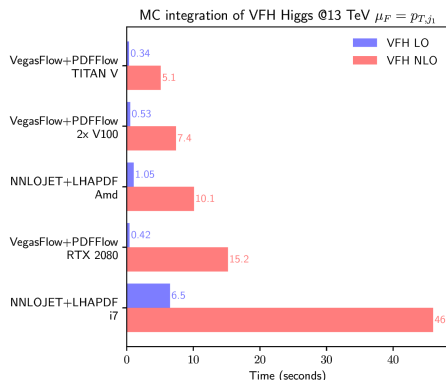
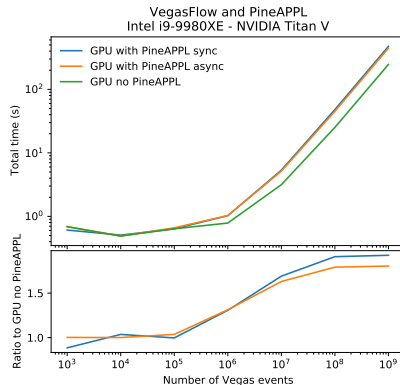
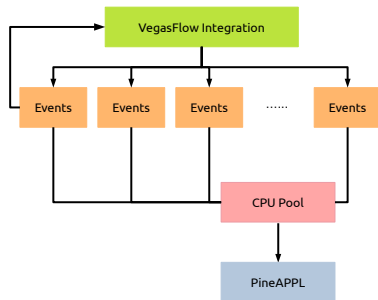


Figure: NNLOJET+LHAPDF vs VegasFlow+PDFFlow

PineAPPL

The grid filling tool PineAPPL (Carrazza, Nocera, Schwan, Zaro, [hep-ph/2008.12789](https://arxiv.org/abs/hep-ph/2008.12789)) addresses the problem of generating grids to produce predictions for generic set of PDFs.



The generation of such grids is a common use of Monte Carlo generators.

Dask

Another very common situation for users of Monte Carlo tools is the possibility of running in a distributed system. In VegasFlow this task is facilitated by the implementation of a dask interface.

```
>>> 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)
```

Cluster systems not included in the dask library should be easy to implement following the same internal logic.

Summary

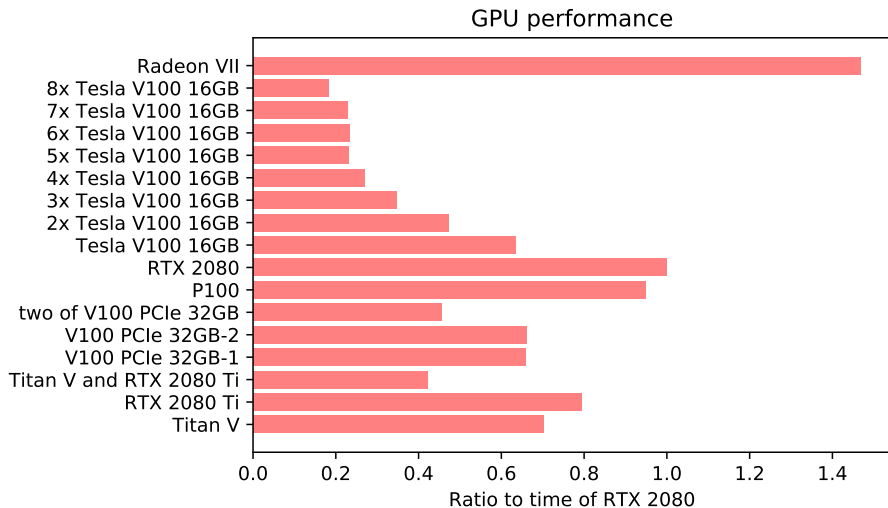
- GPU computation is increasingly gaining traction in many areas of science but it is still not heavily used in particle physics phenomenology.
- Being competitive with CPU for MC simulations.
- Efforts in that direction
- ✓ VegasFlow and PDFFlow provide a framework to run in any device.
- ✓ Good *synergy* with other libraries.
- ✓ Easy implementation of new-generation or NN-based integration algorithms.

Where to obtain the code

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

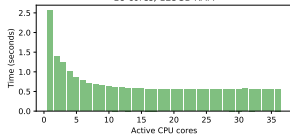
Thanks!

Benchmark on different GPUs

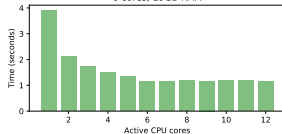


Benchmark on different CPUs

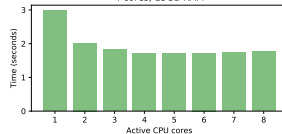
Intel(R) Core(TM) i9-9980XE CPU @ 3.00GHz
18 cores, 125GB RAM



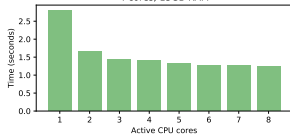
Intel(R) Xeon(R) Gold 6126 CPU @ 2.60GHz
6 cores, 19GB RAM



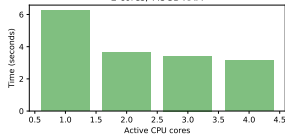
Intel(R) Core(TM) i7-4770 CPU @ 3.40GHz
4 cores, 15GB RAM



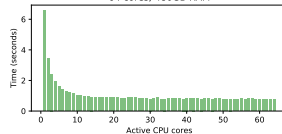
Intel(R) Core(TM) i7-6700K CPU @ 4.00GHz
4 cores, 15GB RAM



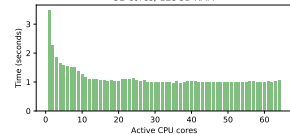
Intel(R) Core(TM) i3-2100 CPU @ 3.10GHz
2 cores, 7.5GB RAM



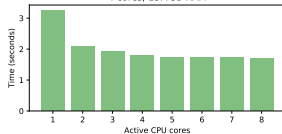
Intel(R) Xeon(R) CPU E5-2686 v4 @ 2.30GHz
64 cores, 480GB RAM



AMD Ryzen Threadripper 2990WX 32-Core
32 cores, 125GB RAM



Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz
4 cores, 15.4GB RAM



Intel(R) Core(TM) i9-9980XE CPU @ 3.00GHz
18 cores, 125GB RAM

