MadFlow: automating Monte Carlo simulation on GPU for particle physics processes

Marco Rossi[1,2]

in collaboration with S. Carrazza[1], J.C. Martinez[1] and M. Zaro[1]

[1]University of Milan [2]CERN-openlab

arXiv:2106.10279 [physics.comp-ph]

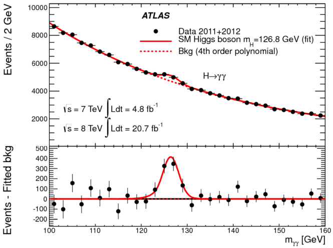
Offshell 2021 - The virtual HEP conference on Run4@LHC
6 – 9 July 2021

Parton-level Monte Carlo generators

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

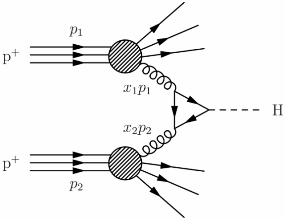
$$\int \mathrm{d}x_1\, \mathrm{d}x_2\, f_1(x_1, q^2) f_2(x_2, q^2) |M(\{p_n\})|^2 \mathcal{J}_m^n(\{p_n\})$$

- $\rightarrow$ $f(x, q)$: 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.

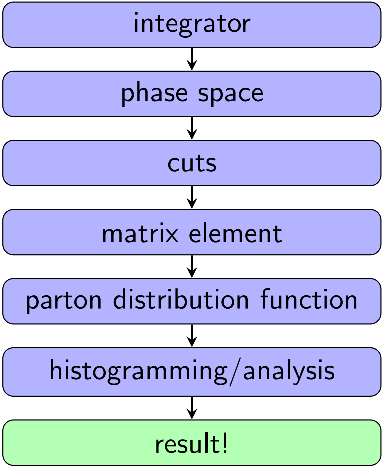


Parton-level Monte Carlo generators ingredients:

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



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



integrator

phase space

cuts

matrix element

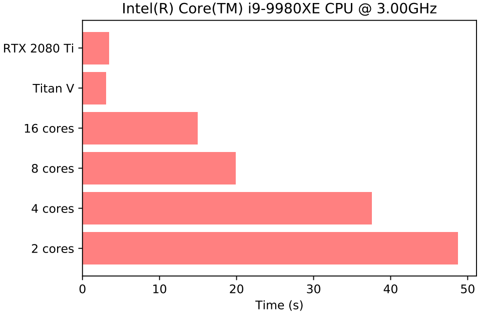
parton distribution function

histogramming/analysis

result!

GPU computing

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



Quick Example:
n-dimensional gaussian function

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

Every event is independent of all other events!

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

Why is then GPU computing not more widespread in HEP?

Most of the more advance theoretical calculations still rely exclusively on CPU.

- ✗ 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 in HEP?

Most of the more advance theoretical calculations still rely exclusively on CPU.

✗ 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 in HEP?

Most of the more advance theoretical calculations still rely exclusively on CPU.

✗ 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.

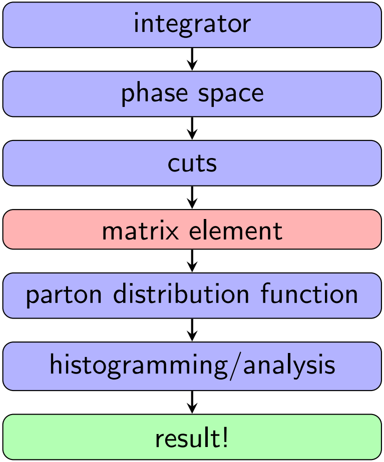
✗ Lack of tools

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

Lack of Tools

Running on a CPU:

Worry only about what you are interested in. For instance, if we want an NNLO computation for $H \to j$ and we have a $Z \to j$ computation we only need to change the matrix elements.



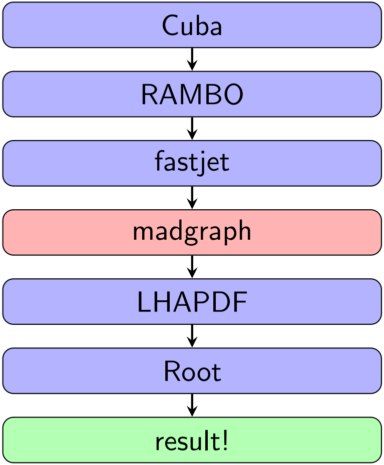
Lack of Tools

Running on a CPU:

Even if we don't already have some obscure and private fortran-based framework already built, there exists a complete tool set for producing results.

- ✓ PDF providers
- ✓ Phase space generators
- ✓ Integrator libraries...

some of which can still provide that sweet 70s' Fortran taste

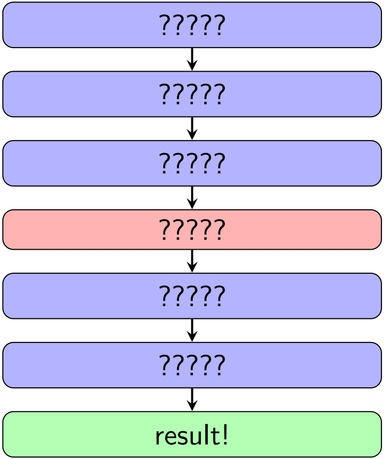


Lack of Tools

Running on a GPU:

There is no such tool set yet

so it needs to be written from scratch

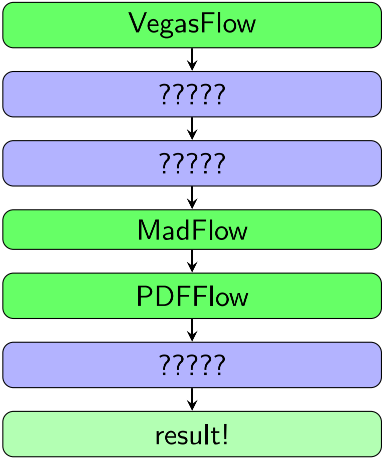


Filling up the box: moving with the flow

The flow suite 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/MadFlow
github.com/N3PDF/PDFFlow

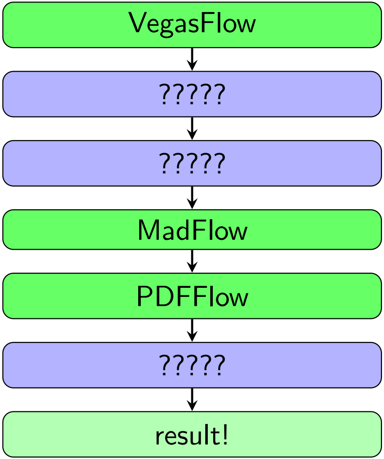


Interface with Madgraph's matrix generation

As a first step towards a full parton-level fixed-order Monte Carlo generator we interface the "flow" suite with Madgraph's matrix element generator.

We take advantage of ALOHA to produce tensorized versions of the matrix elements that can be efficiently run in GPU.

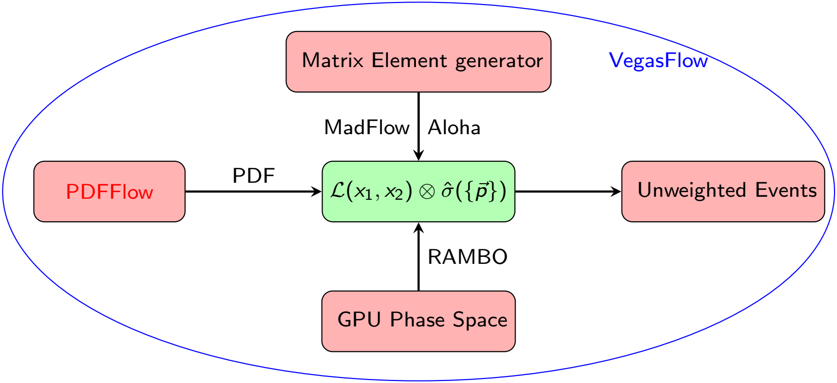
We aim to be modular enough that different ME providers can be used or even combined.



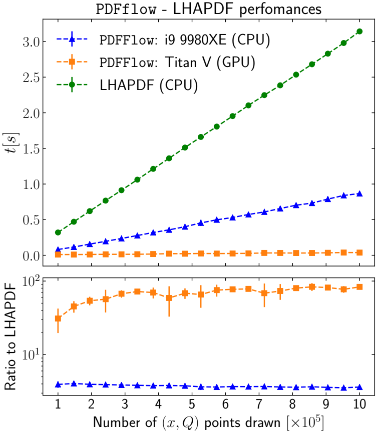
The MadFlow framework

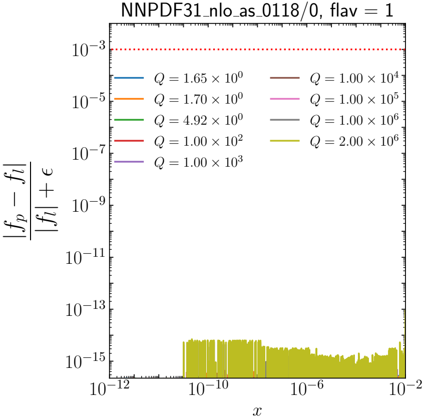
PDF interpolation

“PDFFlow” interpolating algorithm to match LHAPDF in accuracy and outperform in performances.



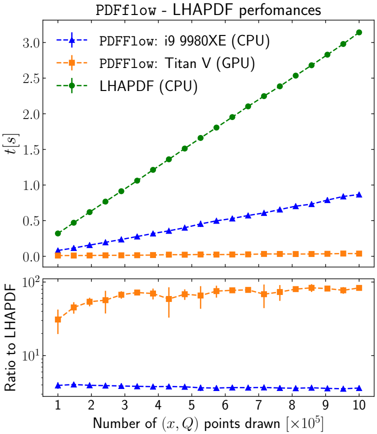
LHAPDF vs PDFFlow

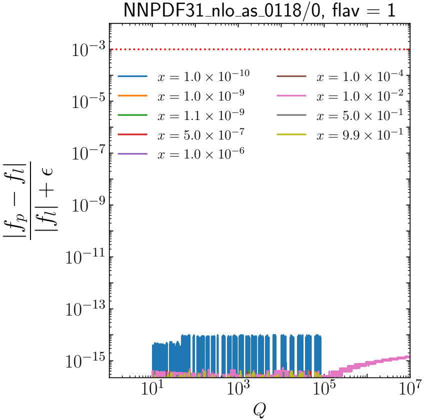




Interpolation in x for fixed Q.

LHAPDF vs PDFFlow



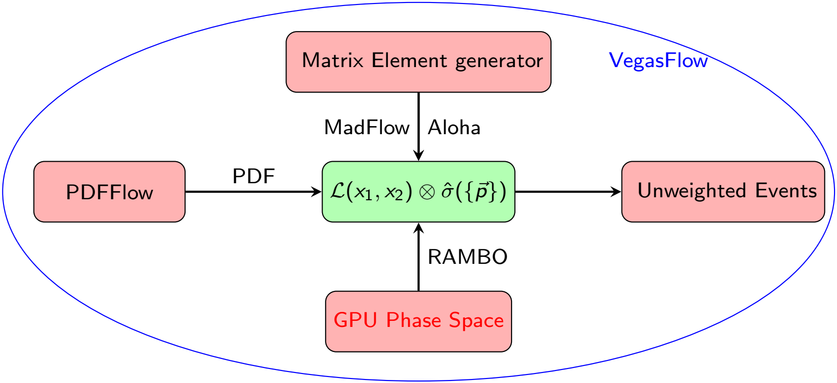


Interpolation in Q for fixed x.

The MadFlow framework

RAMBO

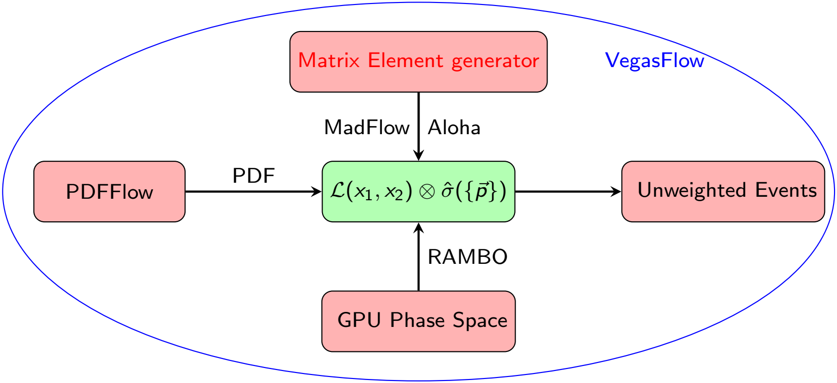
Phase space generator which takes an array of (`n_events`) random numbers and returns (`n_events`) an array of phase space points



The MadFlow framework

Matrix Element (at LO)

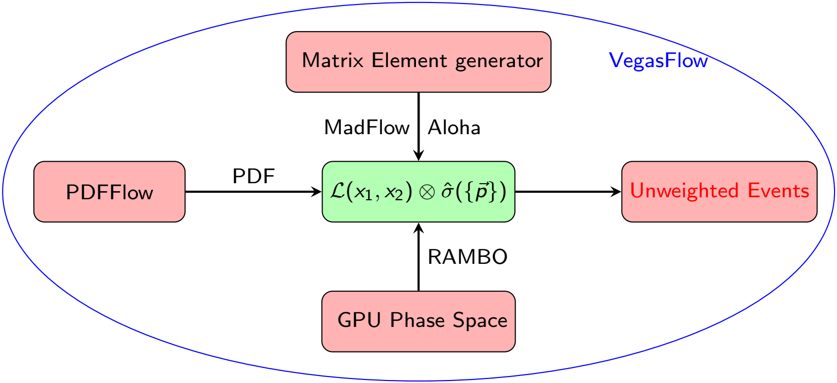
“Madflow” exploits Madgraph and ALOHA routines to generate vectorized metacode with TensorFlow primitives.



The MadFlow framework

Unweighted events

“Madflow” exploits Madgraph accept/reject algorithms to provide unweighted events for queried partonic processes.



Unweighted events and LHE files

- Collect weighted events from PS generator asynchronously.
  - ✓ IO operation do not harm integration performances.
- Dump events in Les Houches Event (LHE 3.0) file format.
- Exploit Madgraph unweighting function to accept/reject weighted events.
  - ✗ Low selection efficiency due to RAMBO (around 5%).

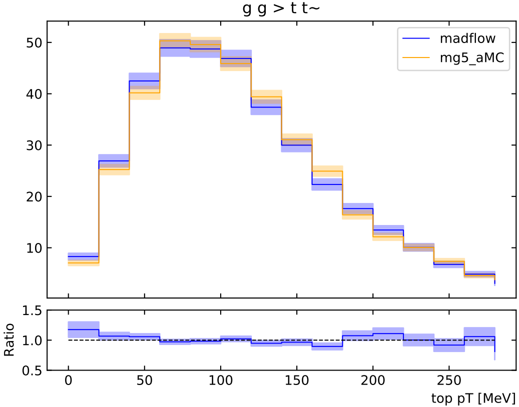
Results: accuracy

Exact same ME and feynman diagrams ✓

Efficient phase space (RAMBO) ✗

Histograms made with unweighted events stored in LHE file ✓

Good compatibility within $2-5\%$ in each bin ✓



Top transverse momentum distribution.

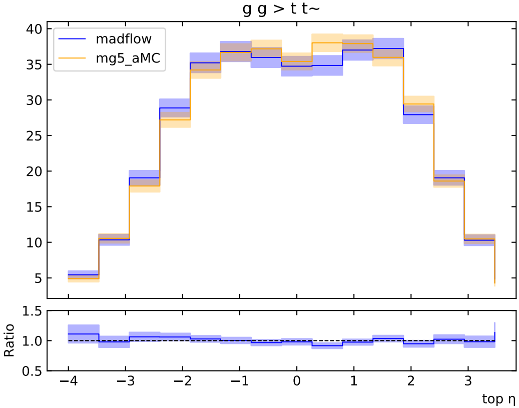
Results: accuracy

Exact same ME and feynman diagrams ✓

Efficient phase space (RAMBO) ✗

Histograms made with unweighted events stored in LHE file ✓

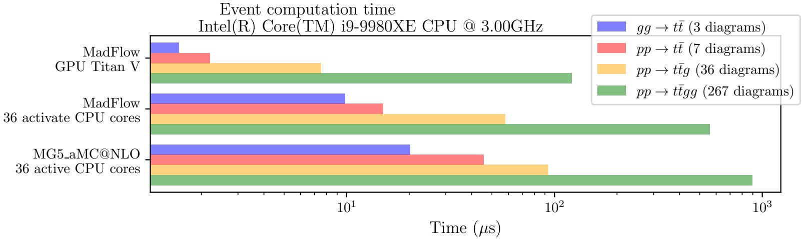
Good compatibility within $2-5\%$ in each bin ✓



Top rapidity distribution.

MadFlow vs plain Madgraph LO

Event computation time for different processes.



✓ MadFlow outperforms Madgraph for all processes.

✗ GPU efficiency decreases with amount of diagrams:

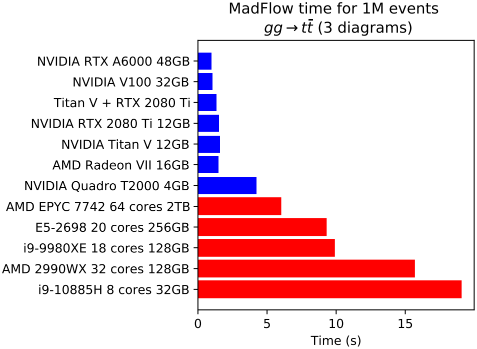
- less GPU memory constraint with new Nvidia A series ($40 - 80$ GB).
- implement compressing techniques.

✓ Able to handle many diagrams (2000+ for $pp \to t\bar{t}ggg$):

- enough to cover NNLO computation complexity (future work).

Results: performance

- Generate 1M events.
- Cut $p_T > 30$GeV on out-going particles.

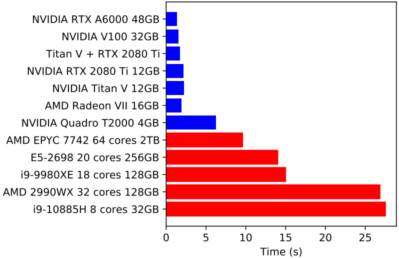


✓ Consumer grade GPUs outperform expensive powerful CPUs.

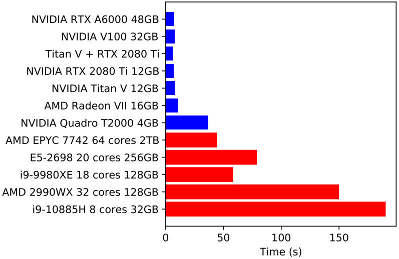
✗ No MC performance optimization techniques implemented yet (ME Leading-Color approximation, MC helicity sampling ...).

Results: performance

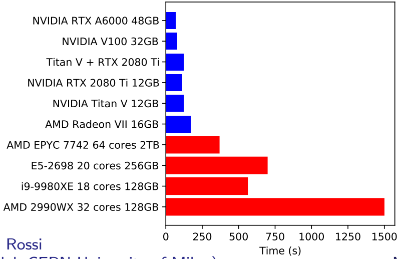
Increasing the number of diagrams works well on GPU.



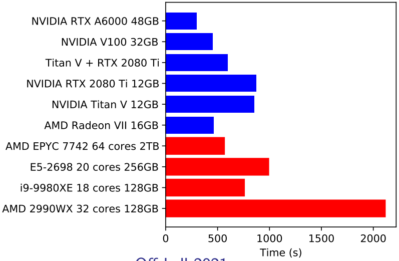
MadFlow time for 1M events
$pp \to t\bar{t}$ (7 diagrams)



MadFlow time for 1M events
$pp \to t\bar{t}g$ (36 diagrams)



MadFlow time for 1M events
$pp \to t\bar{t}gg$ (267 diagrams)



MadFlow time for 100k events
$pp \to t\bar{t}ggg$ (2604 diagrams)

Open source for HEP

Where to obtain the code

The entire Flow suite is open source and can be found at the N3PDF organization repository `github.com/N3PDF`

How to install

MadFlow can be installed from repository only.
VegasFlow and PDFFlow can be installed from the repository or directly with `pip`:

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

Documentation

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

Summary

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

→ Is competitive with CPU for MC simulations.

→ A lot of effort on GPU-based computations.

✓ VegasFlow, PDFFlow and MadFlow provide a framework to run on any device.

✓ Generate all the different pieces (ME, PS, PDFs, integration algorithm) needed for fixed order calculations.

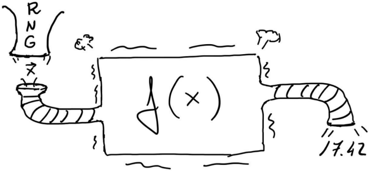
✓ Leading order integration and unweighted events generator in LHE format.

Thanks!

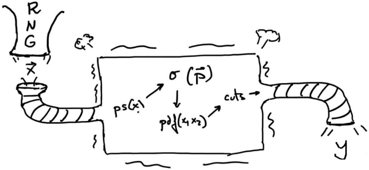
Act in parallel: CPU

The way we do Monte Carlo calculations in CPU already allows for a certain degree of parallelization

$$I = \frac{1}{N} \sum f(\vec{x}_i)$$



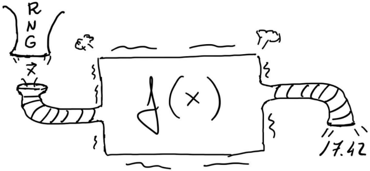
(the function $f(\vec{x})$ might be arbitrarily complicated)



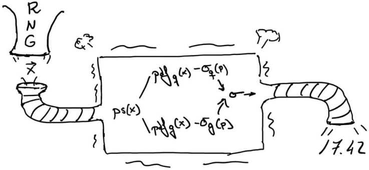
Act in parallel: CPU

The way we do Monte Carlo calculations in CPU already allows for a certain degree of parallelization

$$I = \frac{1}{N} \sum f(\vec{x}_i)$$

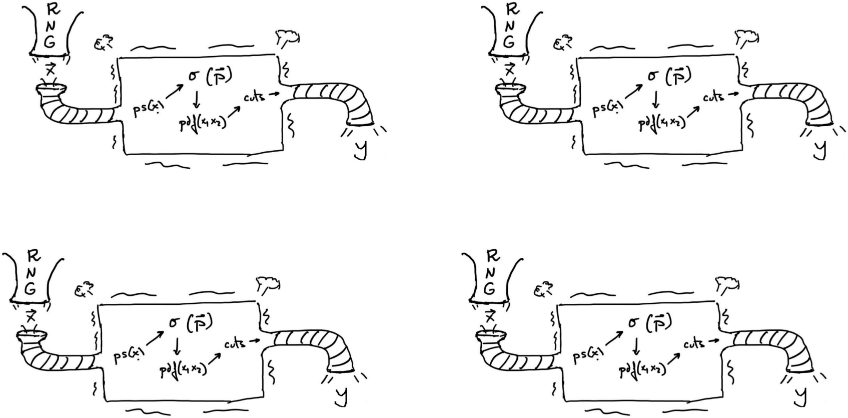


(the function $f(\vec{x})$ might be arbitrarily complicated)



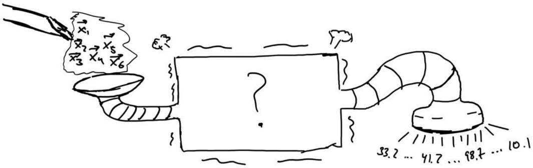
Act in parallel: CPU

The way we do Monte Carlo calculations in CPU already allows for a certain degree of parallelization



Act in parallel: GPU

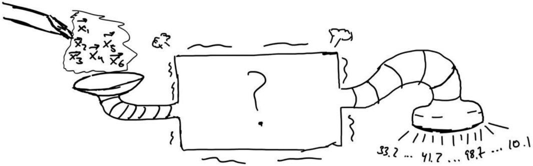
What can we do then in these machines?



We need a completely different machine, which takes a different input and a different output

Act in parallel: GPU

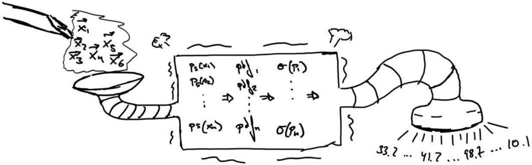
What can we do then in these machines?



$\vec{x}_1$ $\vec{x}_2$ $\vec{x}_3$ $\vec{x}_4$ $\vec{x}_5$ $\vec{x}_6$
?
33.2 ... 41.7 ... 98.7 ... 10.1

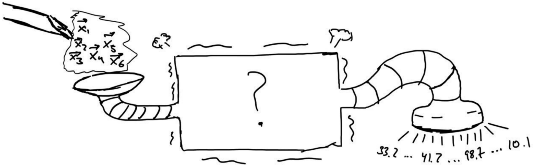
We need a completely different machine, which takes a different input and a different output

All operations must act on all inputs at once!



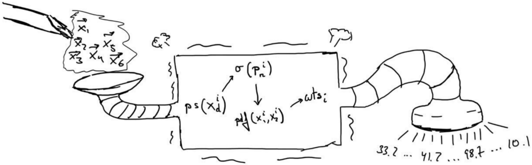
Act in parallel: GPU

What can we do then in these machines?



We need a completely different machine, which takes a different input and a different output

All operations must act on all inputs at once!



So far so good, but how can we do it?