MadFlow: automating Monte Carlo simulation on GPU for particle physics

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

[physics.comp-ph/2106.10279] Eur.Phys.J.C 81 (2021) 7, 656

NNPDF
Machine Learning • PDFs • QCD

International Conference in High Energy Physics (ICHEP2022, Bologna)
July 2022

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. Introduction
   - Monte Carlo integrals
   - Parallelizing your code
2. GPU-enabled tools
   - VegasFlow, PDFFlow, MadFlow
   - Need for speed
3. Conclusions
   - Where to obtain the code

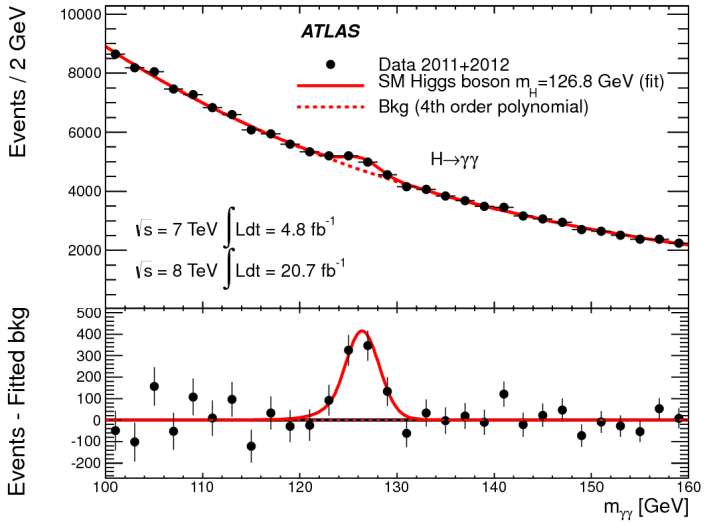
Parton-level Monte Carlo generators

Predictions for observables (for instance for LHC phenomenology) often require the numerical computation of the following integral:

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

where:

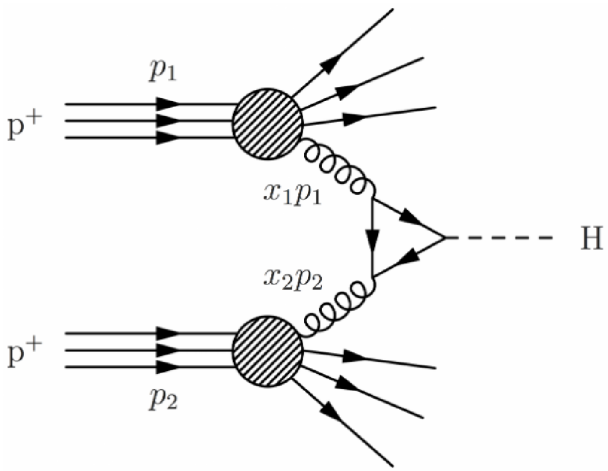
- $f(x, q)$: Parton Distribution Function
- $|M|$: Matrix element of the process
- $\{p_n\}$: Phase space for n particles.
- $\mathcal{J}$: Jet function for n particles to m.



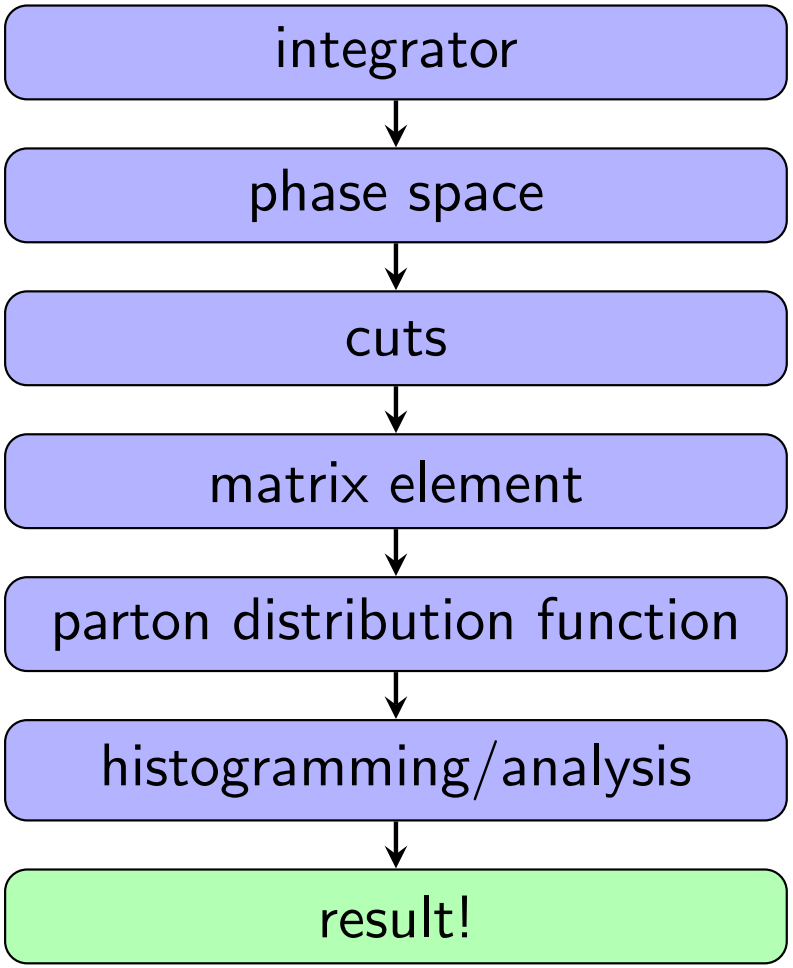
Parton-level Monte Carlo generators ingredients

In practice that requires a number of (mostly independent) ingredients:

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



The numerics being handled numerically by CPU-expensive Monte Carlo (MC) generators.



The age of precision

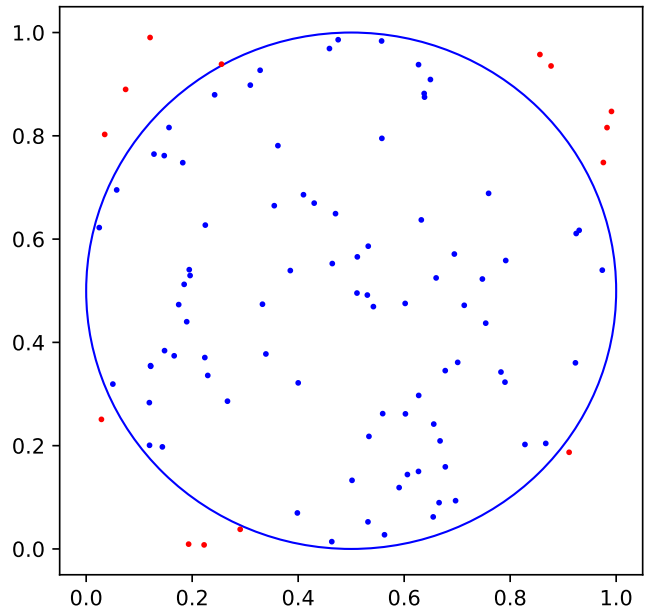
MC algorithms are very convenient as they provide an estimate of the integral and of the error:

$$\int \mathrm{d}^d x f(\vec{x}) \simeq \frac{1}{N} \sum_{i=1}^{N} f(\vec{x}_i) = I \qquad \text{var} = \frac{1}{N} \left(\frac{1}{N} \sum_{i=1}^{N} f(\vec{x}_i)^2 - I^2 \right)$$

However, the error decreases with the number of shots only as $\frac{1}{\sqrt{N}}$.

Despite techniques to reduce the number of shots necessary, the $\mathcal{O} \simeq \frac{1}{\sqrt{N}}$ holds.

$\pi \simeq 3.2$

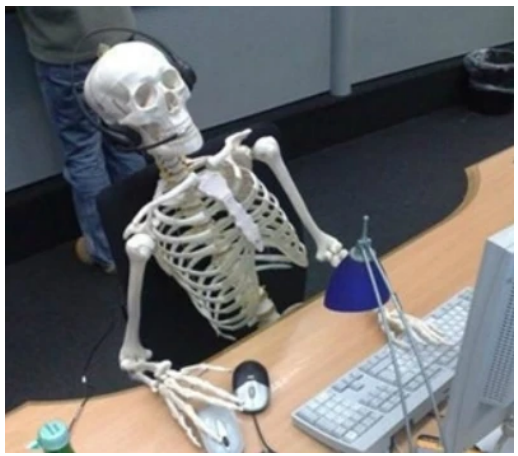


The age of precision

MC algorithms are very convenient as they provide an estimate of the integral and of the error:

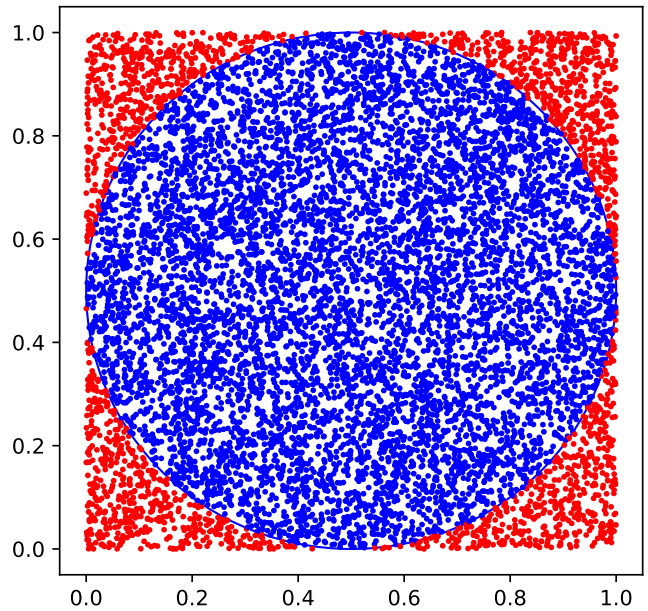
$$\int \mathrm{d}^d x f(\vec{x}) \simeq \frac{1}{N} \sum_{i=1}^{N} f(\vec{x}_i) = I \qquad \text{var} = \frac{1}{N} \left(\frac{1}{N} \sum_{i=1}^{N} f(\vec{x}_i)^2 - I^2 \right)$$

However, the error decreases with the number of shots only as $\frac{1}{\sqrt{N}}$.



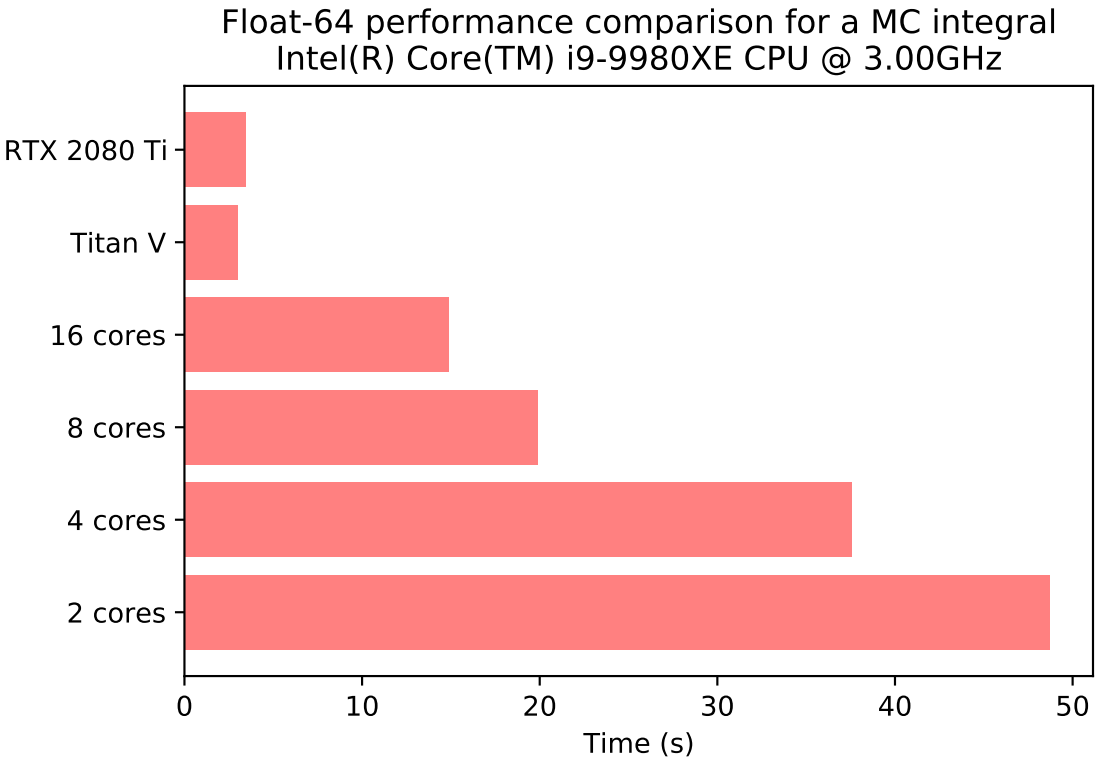
It can take forever

$\pi \simeq 3.16$



Parallel computing to the rescue

MC integrations are made out of independent events, making them a perfect target for massive parallelization. Exactly where GPU shines!



Quick Example:
n-dimensional gaussian function

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

Every event is independent of all other events!

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

If it is so good, why are we not using GPUs everywhere?

At least in the field of theoretical calculations there are a few points holding progress back

✗ 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.

If it is so good, why are we not using GPUs everywhere?

At least in the field of theoretical calculations there are a few points holding progress back

✗ 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.

If it is so good, why are we not using GPUs everywhere?

At least in the field of theoretical calculations there are a few points holding progress back

✗ 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.

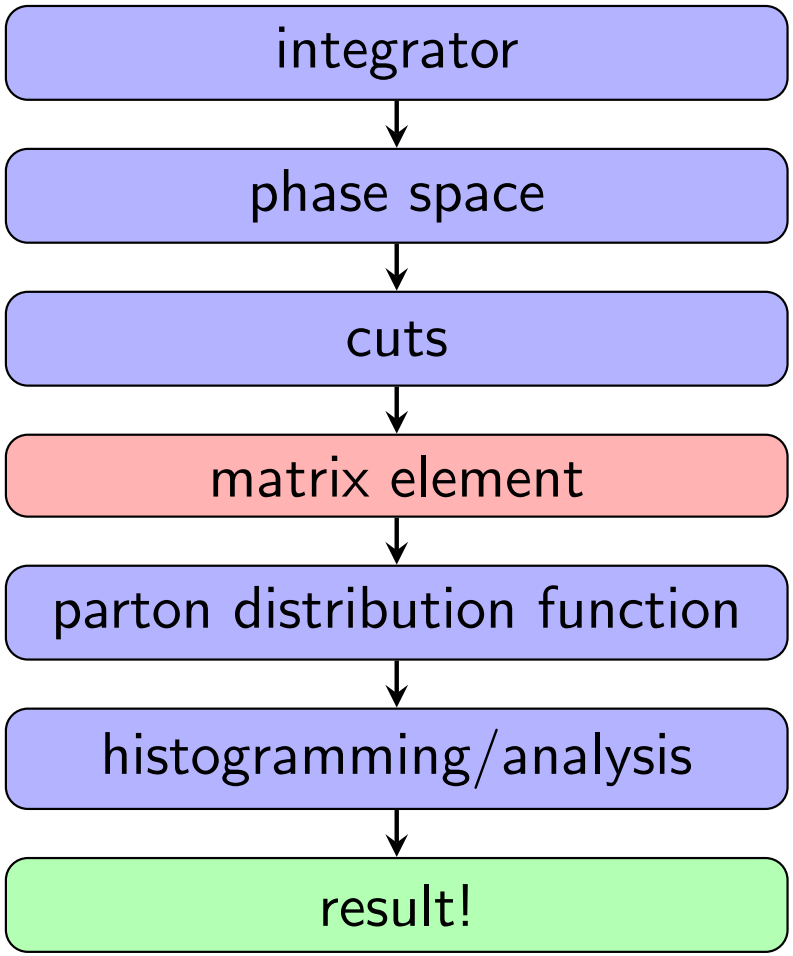
✗ Lack of tools
- Many ready-made tools for CPU.
- GPUs are still decades behind in the hep-ph world.

Lack of Tools

Running on a CPU:

Indeed, you can usually only worry about the part of the calculation that you are interested in (say, a new NNLO matrix element).
While you can find tools that solve everything else (if you didn't already had that tools yourself!)

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

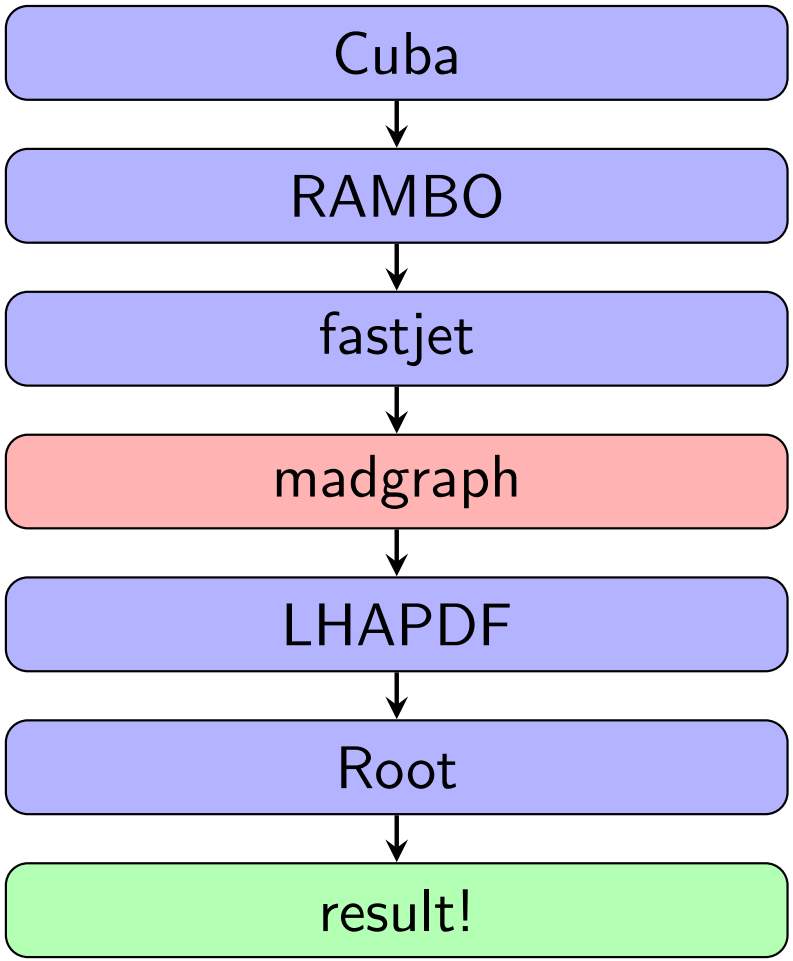


Lack of Tools

Running on a CPU:

Indeed, you can usually only worry about the part of the calculation that you are interested in (say, a new NNLO matrix element).
While you can find tools that solve everything else (if you didn't already had that tools yourself!)

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

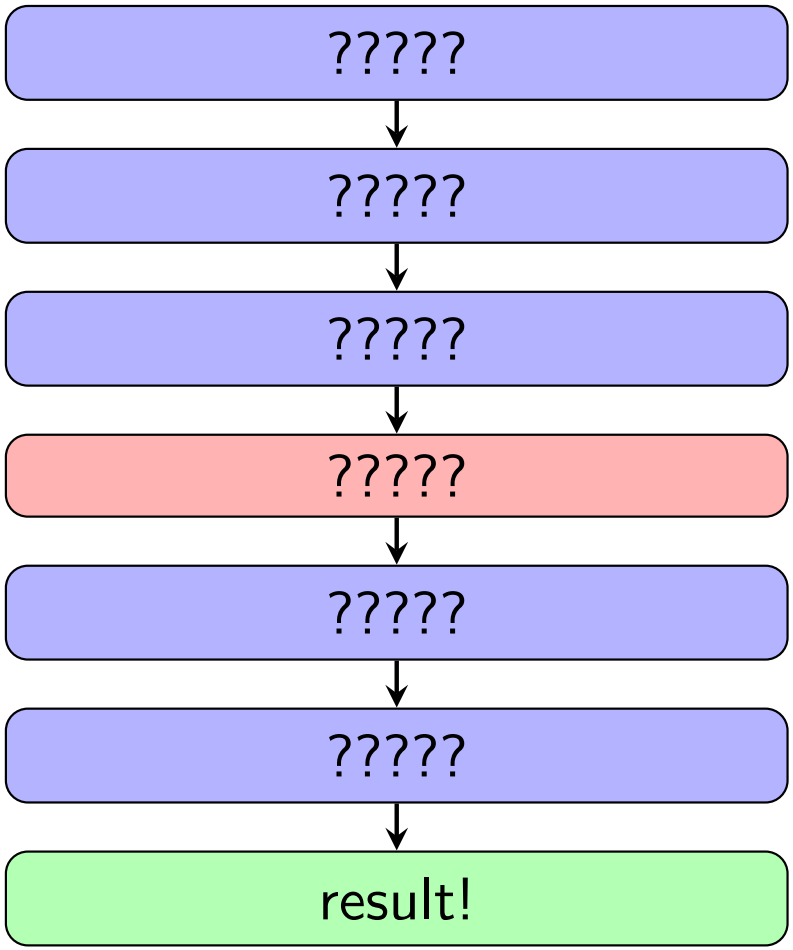


Lack of Tools

Running on a GPU:

There is no such tool set yet

so it needs to be written from scratch



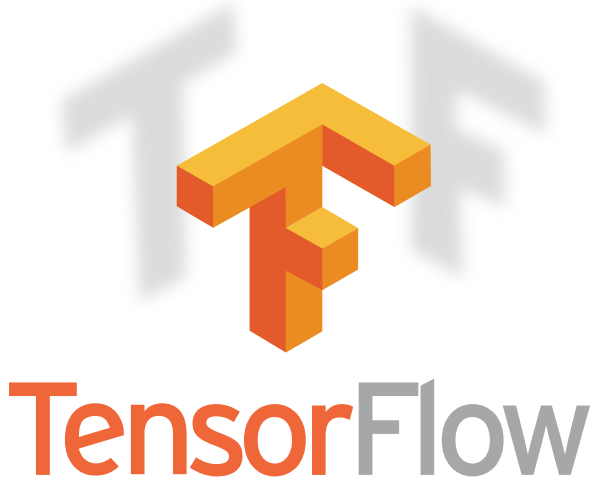
Filling up the box: tools for modern computation

The goal is to provide tools that can facilitate the transition:

VegasFlow: Monte Carlo library with different algorithms that can be used in any device: single-threaded and multi-threaded CPUs or AMD/nvidia GPUs.

PDFFlow: Bulk PDF interpolation, specially well suited for parallel calculation where sequential steps can harm performance.

- Python and TF-based engine.
- Compatible with other languages: Cuda, C++, Rust, Fortran
- Seamless CPU and GPU computation out of the box (develop in a laptop, deploy in a cluster)



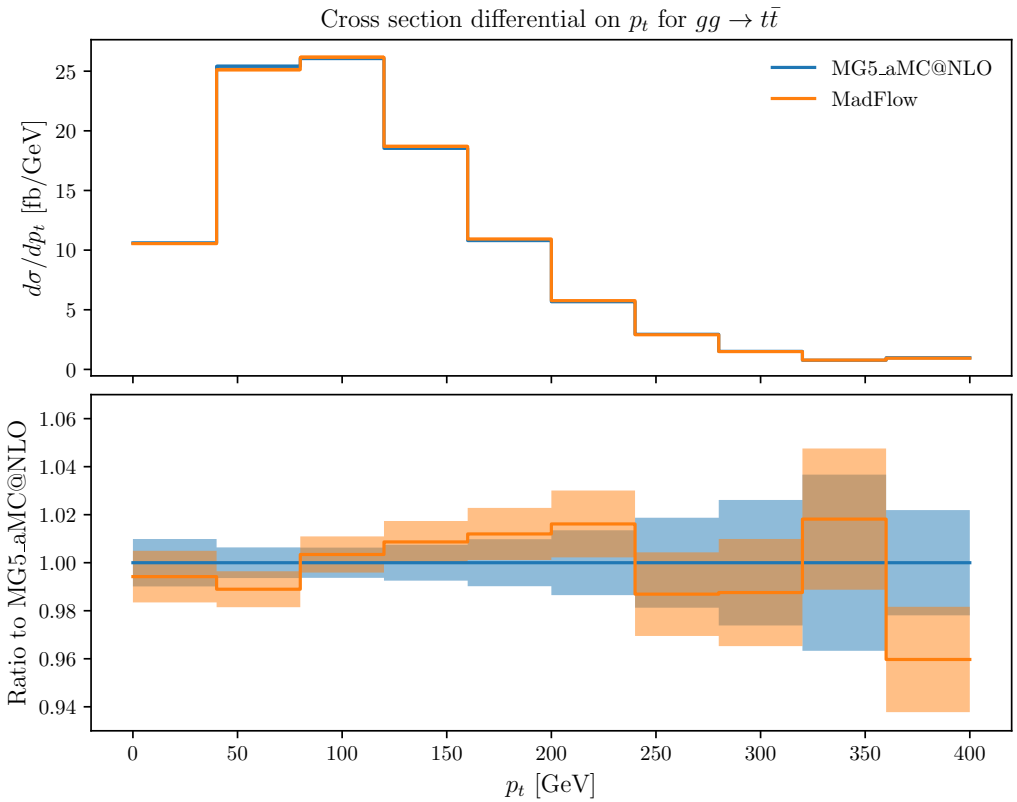
MadFlow: a madgraph interface

An example of what can be obtained is **MadFlow**: Taking advantage of Madgraph's ALOHA we produce `tensorflow`-versions of the matrix elements.

The `TensorFlow` library contains all necessary kernels to run the matrix elements in parallel.

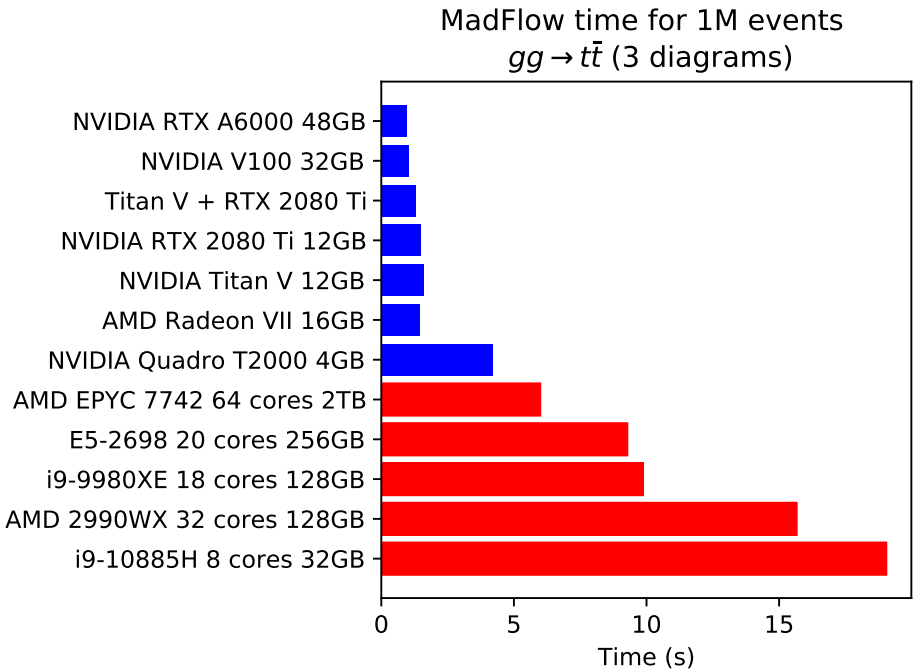
Everything can run in both a CPU or a GPU

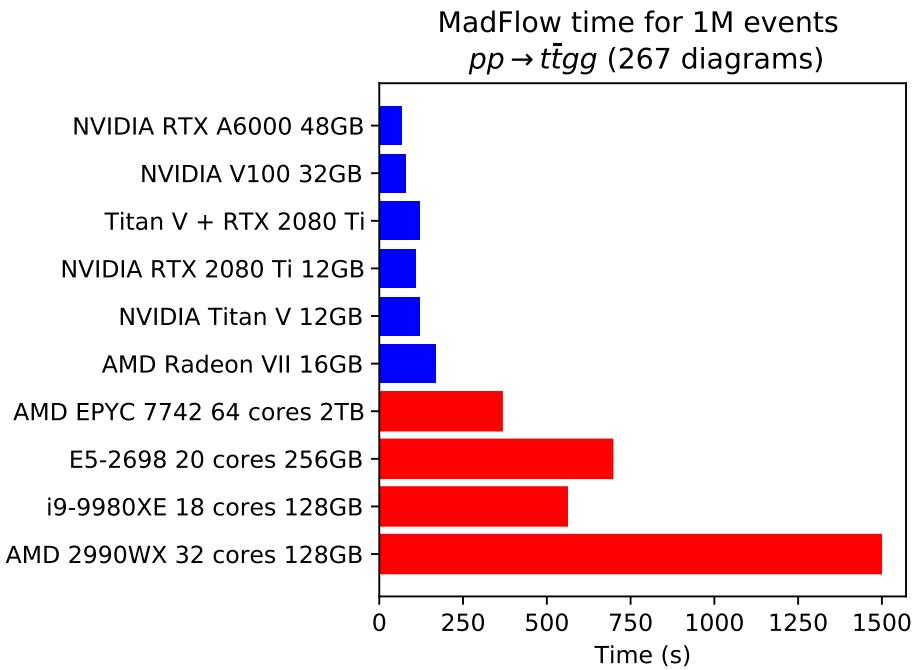
Perfect compatibility ✓



MadFlow in different devices

We see speed-ups for both complex and simple processes



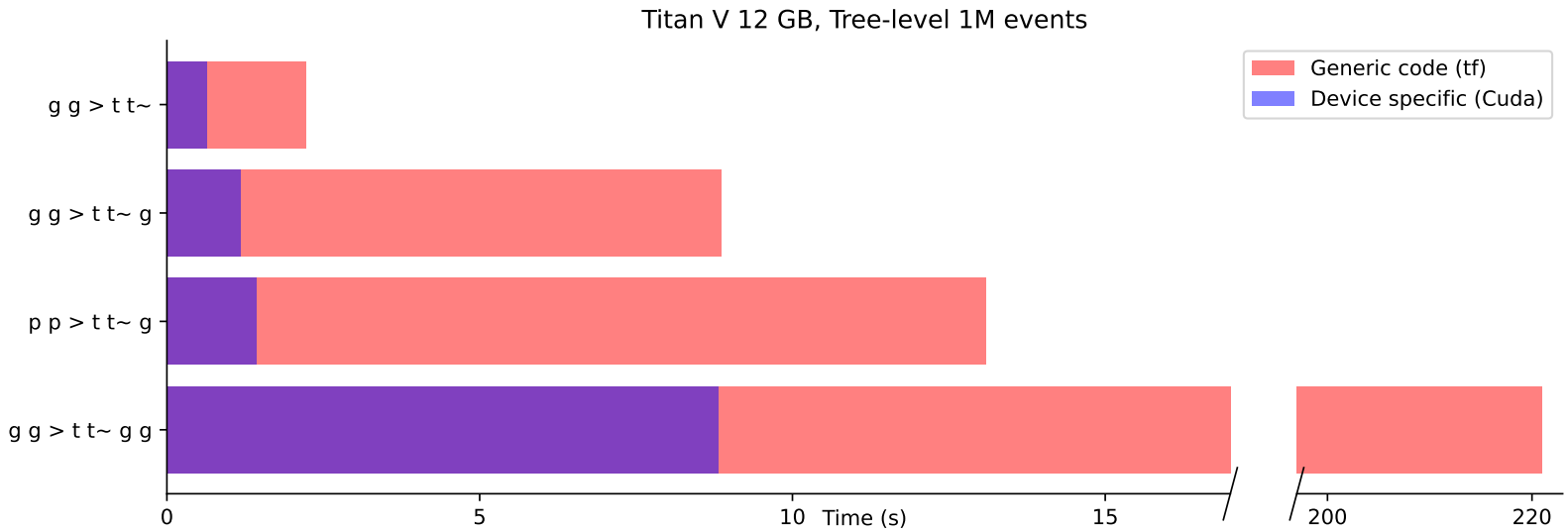


While having the whole thing in `TensorFlow` gives us great flexibility (the exact same code is running in all those systems) we might want to forfeit some flexibility in exchange for device-based optimization.

Interfacing with CUDA

Main goal: flexibility and ease-of-use, that also means one can also make life harder for oneself... for a benefit.

- ✗ Limited to a single architecture
- ✗ Requires a transpiler to the "low"-level language of choice
- ✓ More efficient memory management and performance
- ✓ We can limit this more complicated step only to the bottlenecks



Open source for HEP

Where to obtain the code

Vegasflow, **PDFFlow** and **MadFlow** are open source and can be found at the N3PDF organization repository `github.com/N3PDF` (alongside other projects by the group)

How to install

They can all easily be installed with `pip`:

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

Documentation

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

Summary

- Monte Carlo simulations (fixed-order and otherwise) are great targets for parallelization on hardware accelerators.

→ Despite being more than competitive with CPU not many groups are working on it! (see talk yesterday by A. Valassi)

✗ Maybe we still have a big entry barrier?

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

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

✓ **Remove all entry barrires while still leaving space for further optimization**

Available open source

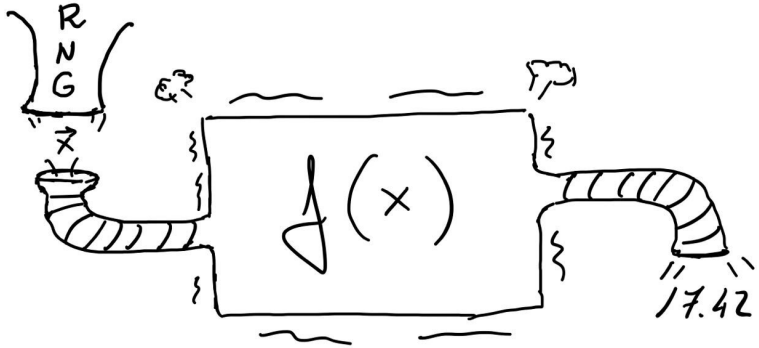
the code for madflow is available at
https://github.com/n3pdf/madflow

Thanks!

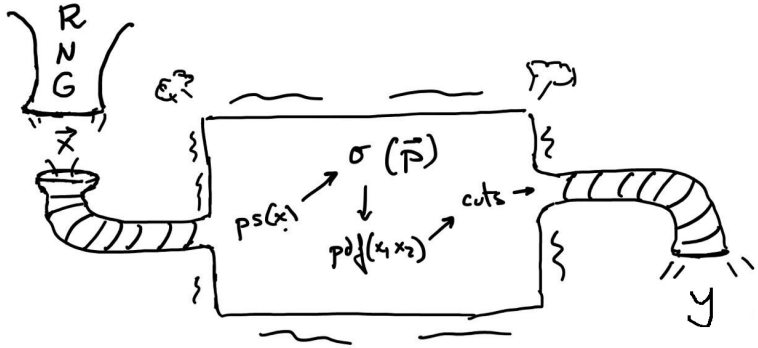
Act in parallel: CPU

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

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



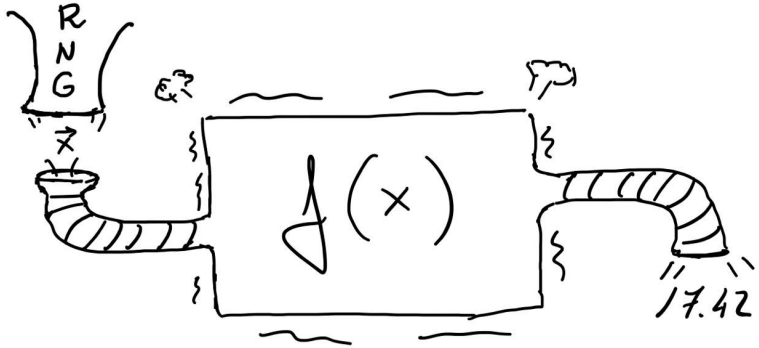
(the function $f(\vec{x})$ might be arbitrarily complicated)



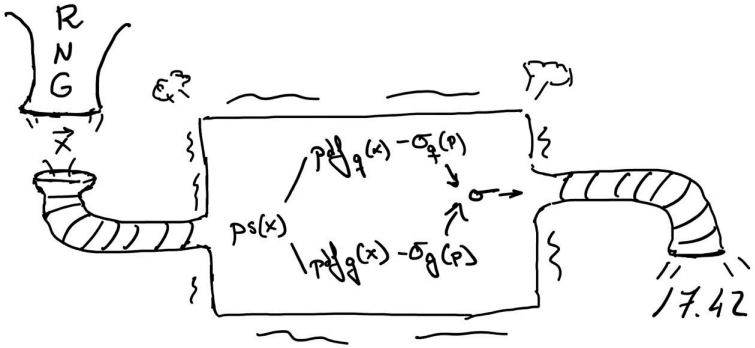
Act in parallel: CPU

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

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

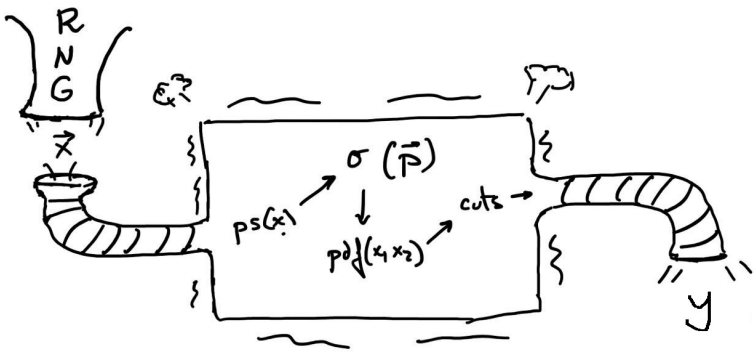


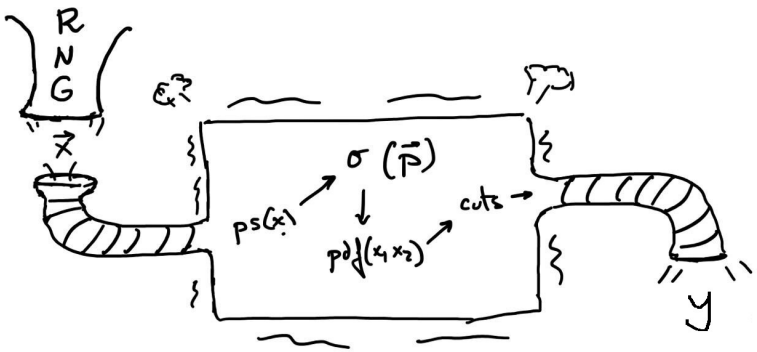
(the function $f(\vec{x})$ might be arbitrarily complicated)

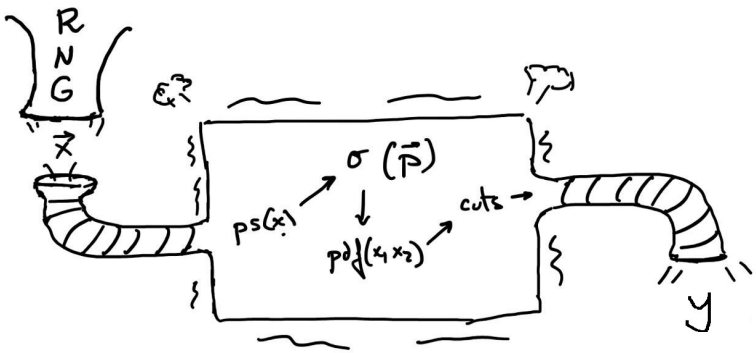


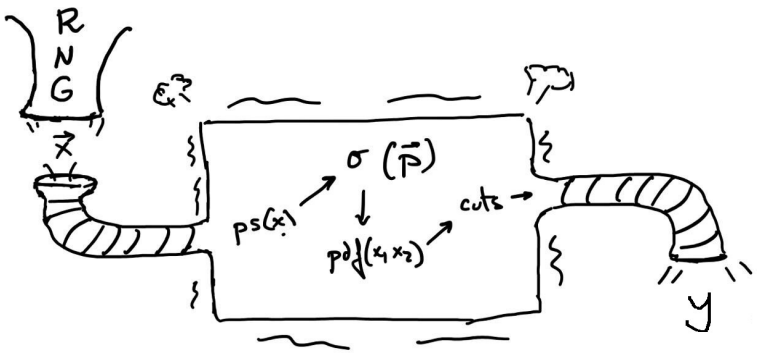
Act in parallel: CPU

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



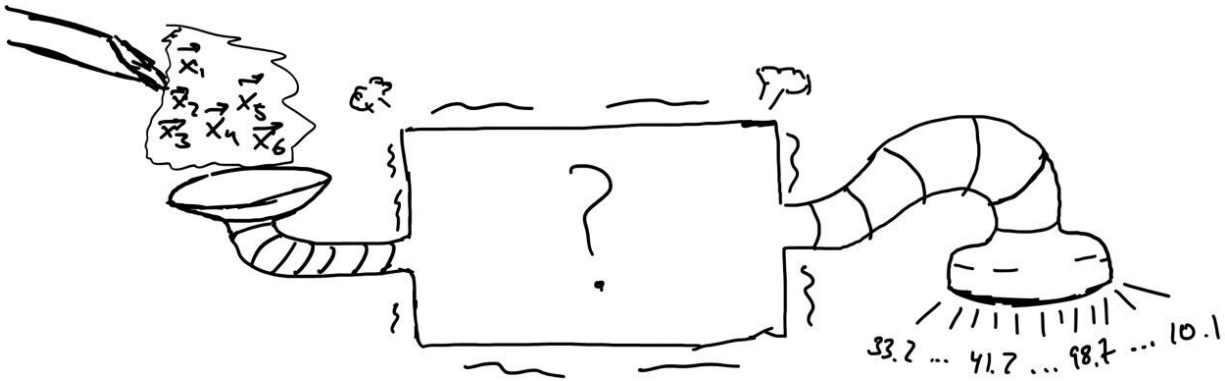






Act in parallel: GPU

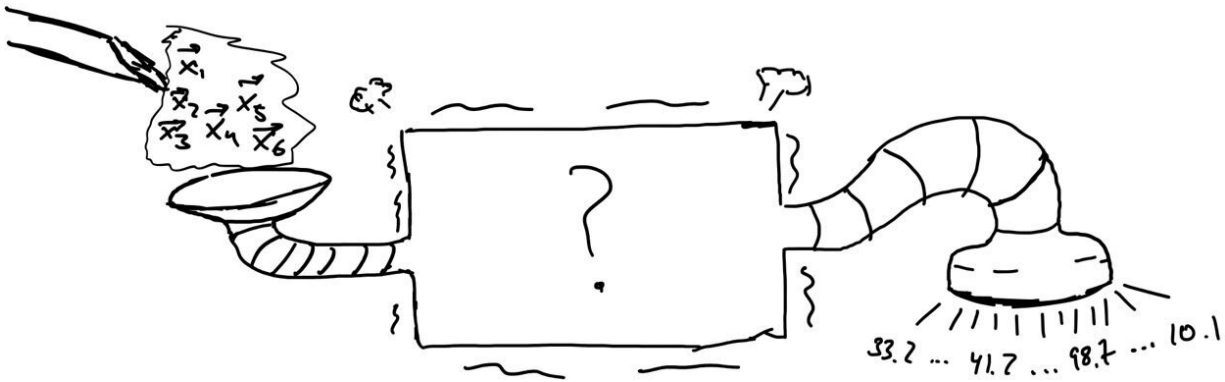
What can we do then in these machines?



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

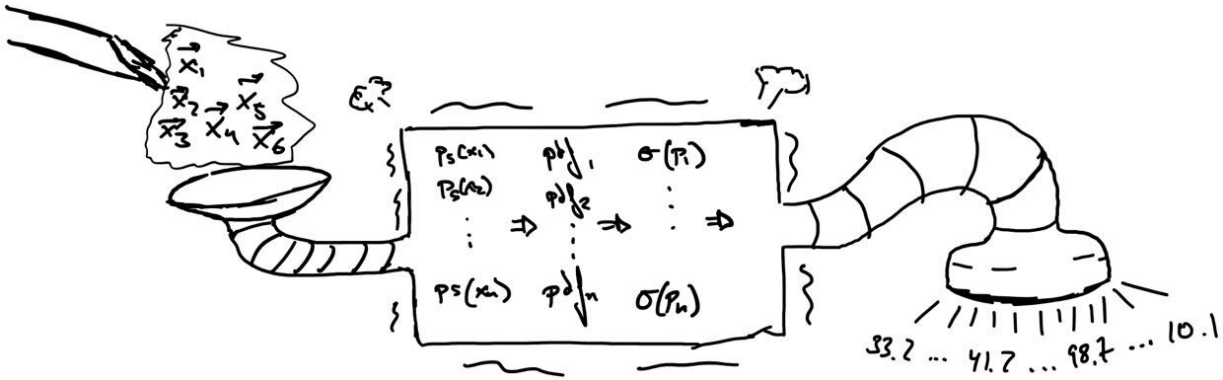
Act in parallel: GPU

What can we do then in these machines?



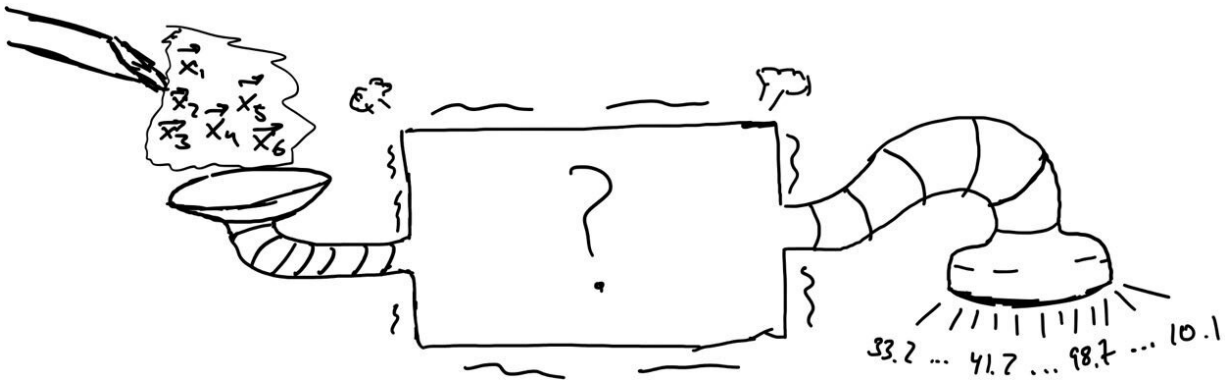
We need a completely different machine, which takes a different input and a different output

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!