Parton Showers on GPUs with GAPS

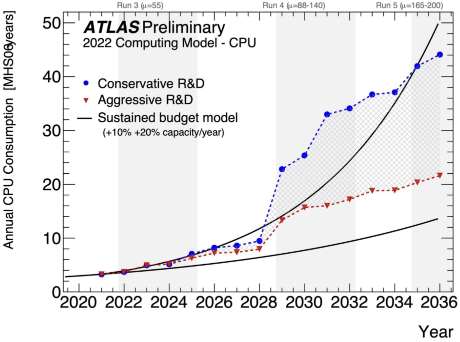
based on [2403.08692] and `siddharthsule/gaps`

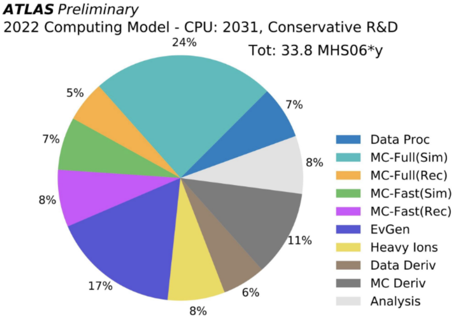
Siddharth Sule (Sid)
with Michael H. Seymour (Mike)

Parton Showers and Resummation, 4 July 2024

Event Generation is Expensive

CPU usage to surpass sustainable budget. Majority is MC, of which EvGen contributes 17% [CERN-LHCC-2022-005].





Event Generation is Expensive

Two ways to approach this problem:

1. Profiling and treating bottlenecks [2209.00843]
2. Parallelising tasks to reduce execution time and cost

Event Generation is Expensive

Two ways to approach this problem:

1. Profiling and treating bottlenecks [2209.00843]
2. Parallelising tasks to reduce execution time and cost

We can accelerate this parallelisation by using **GPUs:**

- Matrix Element + Phase Space: PEPPER [2311.06198], MadGraph4GPU [2303.18244]
- PDF Evaluations: LHAPDF [1412.7420, 2311.06198] PDFFlow [2009.06635]

Where do we go from here?

Even if LO/NLO events are generated on GPU, we still need to plug them into a CPU event generator for MPI, showering and hadronisation…

Where do we go from here?

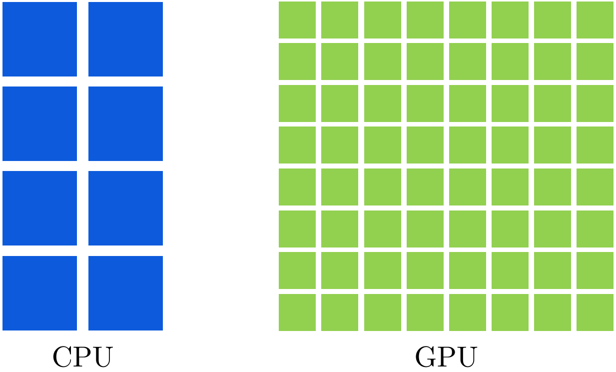
Even if LO/NLO events are generated on GPU, we still need to plug them into a CPU event generator for MPI, showering and hadronisation…

Can we do these on a GPU?

In This Talk

The Parallelised Veto Algorithm

Preliminaries - GPUs and SIMT



Preliminaries - GPUs and SIMT

SIMT = Single Instruction Multiple Threads

CPU: For Loop

```
for i in range(len(a)):
    b[i] = 3*a[i] + 2
```

GPU: Threads in a Kernel

```
if thread_i < len(a):
    b[thread_i] = 3*a[thread_i]
        + 2
```

Preliminaries - GPUs and SIMT

SIMT = Single Instruction Multiple Threads

CPU: For Loop

```
for i in range(len(a)):
    b[i] = 3*a[i] + 2
```

GPU: Threads in a Kernel

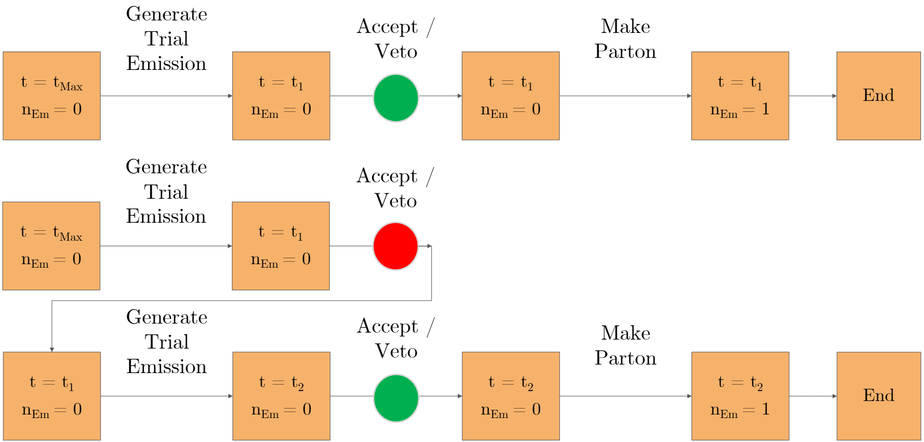
```
if thread_i < len(a):
    b[thread_i] = 3*a[thread_i]
        + 2
```

+ GPUs have ~ 1000 cores, so you can parallelise more tasks than a CPU cluster

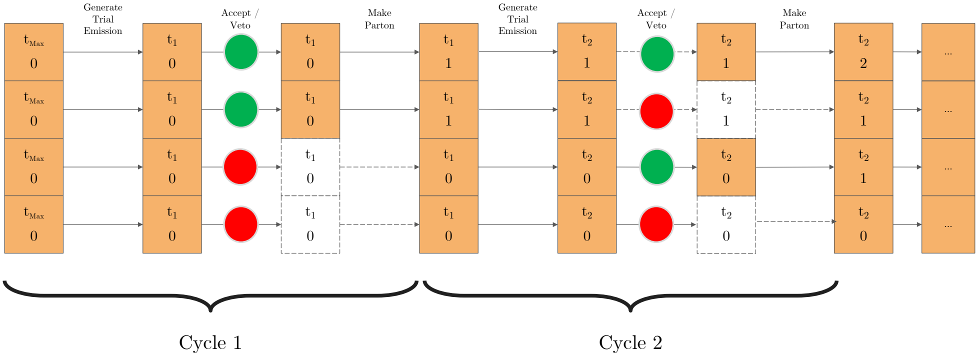
– GPU cores aren't as sophisticated as CPU cores; tasks have to be fairly simple

– SIMT doesn't allow branching tasks by construction (more on this soon…)

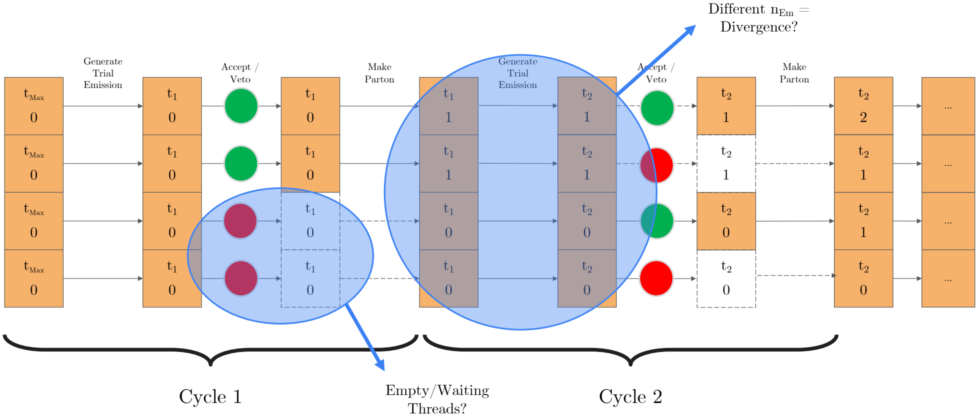
The Veto Algorithm



The Parallelised Veto Algorithm



The Parallelised Veto Algorithm



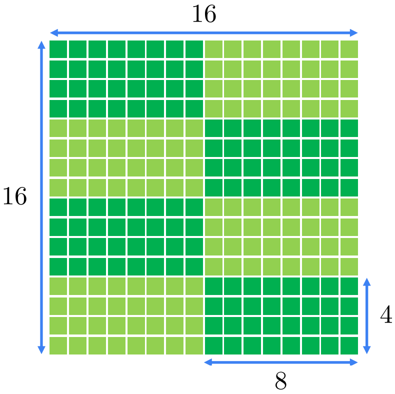
Different n_{Em} = Divergence?

Empty/Waiting Threads?

A Closer Look at GPU Architecture

Threads divided into batches of 32 called *warps*

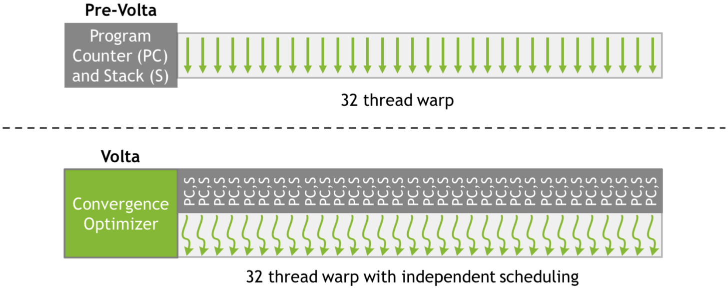
- Warps fully independent, like CPU cores
- For Example: 256 Threads = 8 Warps



So we deal with at-most 32 diverging threads, not thousands [NVIDIA Guide].

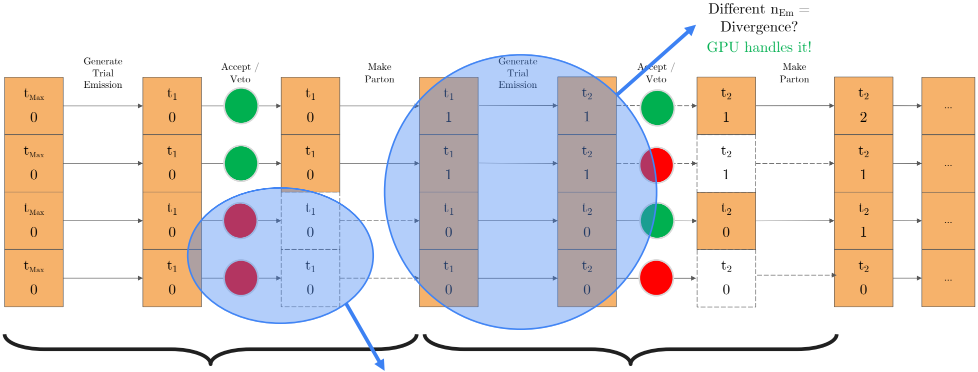
Enter Independent Thread Scheduling

GPUs now have the ability to have threads that diverge [NVIDIA Report].



Threads can execute their own commands (interleaved with other threads' commands).
If a few threads have same command, convergence optimizer runs them together.

The Parallelised Veto Algorithm

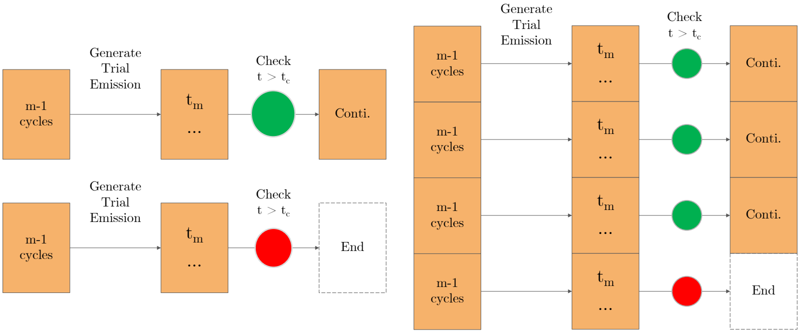


Different n_{Em} = Divergence?
GPU handles it!

Empty/Waiting Threads?
Not efficient, but OK!

The Shower Cutoff

Showering in Parallel means completed events must “wait” for incomplete events.



Implementation and Validation

Implementation Overview

Starting Point: S. Höche's Parton Shower Tutorial [1411.4085]

- Accurate Implementation of Massless Final State Catani–Seymour Shower
- Includes ME, α_s^{NLO}, Durham Alg. and Yoda File Writer

Implementation Overview

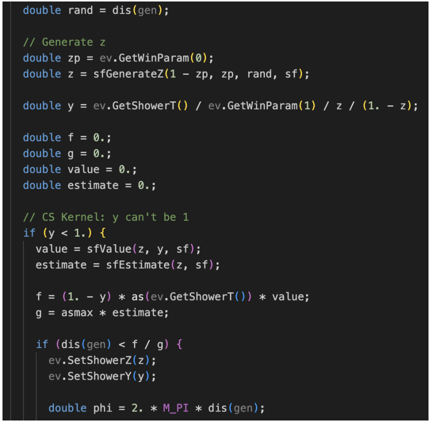
Starting Point: S. Höche's Parton Shower Tutorial [1411.4085]

- Accurate Implementation of Massless Final State Catani–Seymour Shower
- Includes ME, α_s^{NLO}, Durham Alg. and Yoda File Writer

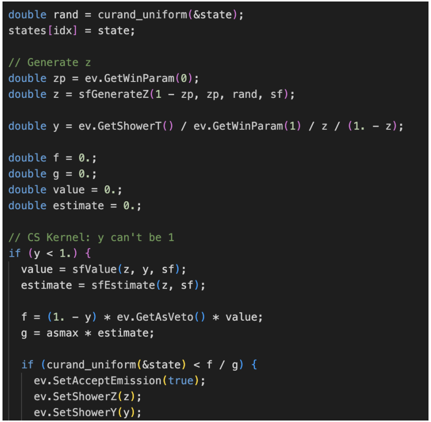
Rewritten for CPU in C++ and for GPU in CUDA C++

1. Ensure identical commands but different structure
2. Validate both codes with results from the Tutorial
3. Compare Execution Times

Implementation - Example Code Snippet

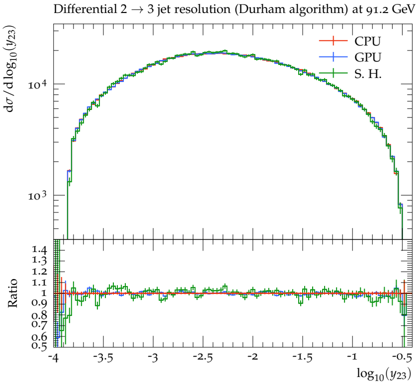


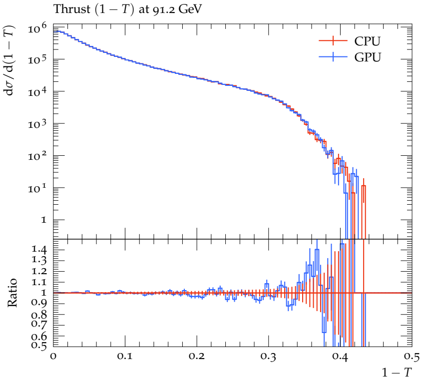
```
double rand = dis(gen);

// Generate z
double zp = ev.GetWinParam(0);
double z = sfGenerateZ(1 - zp, zp, rand, sf);

double y = ev.GetShowerT() / ev.GetWinParam(1) / z / (1. - z);

double f = 0.;
double g = 0.;
double value = 0.;
double estimate = 0.;

// CS Kernel: y can't be 1
if (y < 1.) {
  value = sfValue(z, y, sf);
  estimate = sfEstimate(z, sf);

  f = (1. - y) * as(ev.GetShowerT()) * value;
  g = asmax * estimate;

  if (dis(gen) < f / g) {
    ev.SetShowerZ(z);
    ev.SetShowerY(y);

    double phi = 2. * M_PI * dis(gen);
```



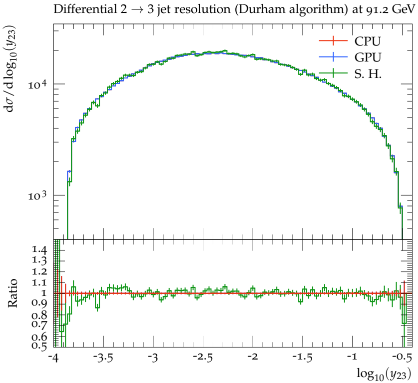
```
double rand = curand_uniform(&state);
states[idx] = state;

// Generate z
double zp = ev.GetWinParam(0);
double z = sfGenerateZ(1 - zp, zp, rand, sf);

double y = ev.GetShowerT() / ev.GetWinParam(1) / z / (1. - z);

double f = 0.;
double g = 0.;
double value = 0.;
double estimate = 0.;

// CS Kernel: y can't be 1
if (y < 1.) {
  value = sfValue(z, y, sf);
  estimate = sfEstimate(z, sf);

  f = (1. - y) * ev.GetAsVeto() * value;
  g = asmax * estimate;

  if (curand_uniform(&state) < f / g) {
    ev.SetAcceptEmission(true);
    ev.SetShowerZ(z);
    ev.SetShowerY(y);
```

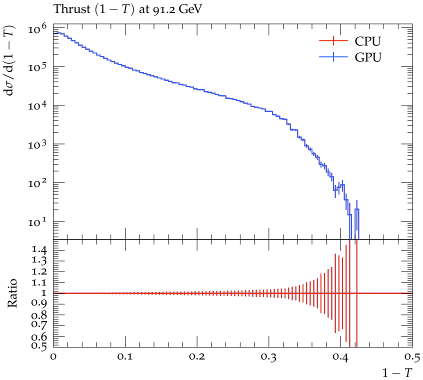
Validation - Jet Rates and Event Shapes





(New!) Using Identical RNG and Seed


Differential 2 → 3 jet resolution (Durham algorithm) at 91.2 GeV


Thrust $(1-T)$ at 91.2 GeV

Comparison of Execution Times

Definitions and Criteria

Execution Time: Time taken for a component in its *entirety*:

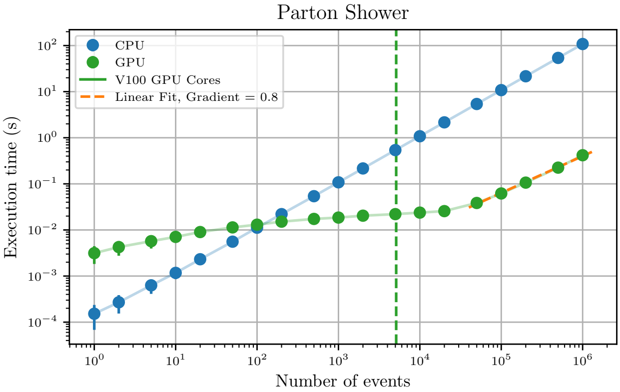
- Include any initialisation and setup tasks in execution time
- Mainly because GPU tasks are surrounded by smaller CPU Tasks

We compare **1 CPU Core** against **1 CPU Core + 1 GPU**

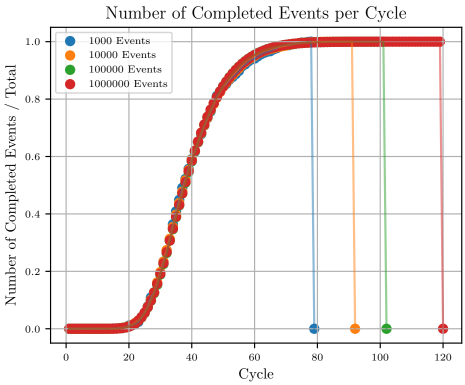
- CPU: Intel Xeon CPU E5-2620 v4 @ 2.10GHz [Specifications]
- GPU: NVIDIA Tesla V100 for PCIe, 16 GB, 5,000 Cores [Specifications]

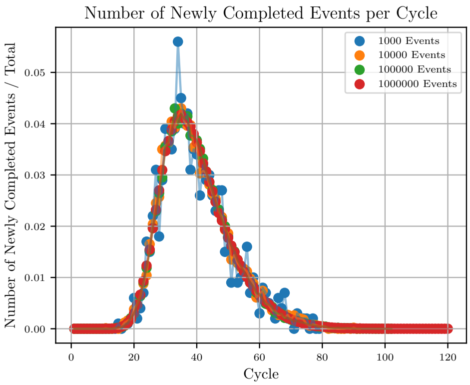
Execution Time - Parton Shower

Maximum speed up: 258x at 1,000,000 events (0.4s vs 103s)



Studying the Shower Cycles

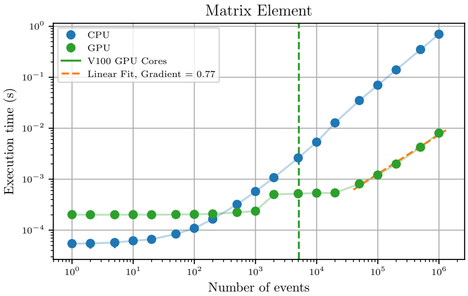


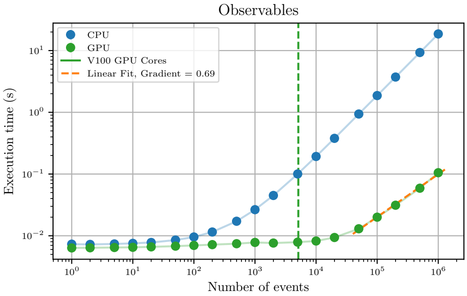


IMPORTANT: Cycle $\neq$ Time! A smaller number of events leads to quicker cycles.

Matrix Element and Observables

Maximum speed up: 87x for ME and 177x for Observables at 1,000,000 events


Matrix Element


Observables

Maximum Speedup of ME + PS + Ob: 239x for 1,000,000 events (0.5s vs 120s)

Context - Costs of CPUs vs CPU+GPU

Type	ID	Cost	Power
CPU	Intel Xeon (16 Cores)	≈ $ 500	85 W (5 W per core)
GPU	NVIDIA V100	≈ $ 5000	250 W

Around 15–16 CPUs (240–256 Cores) needed to match 1 CPU Core + 1 GPU:

- 16 × $ 500 = $ 8000, more expensive than CPU+GPU
- 16 CPUs = 1360 W, 1 CPU Core + 1 GPU = 255 W

Using GPUs leads to cheaper setup and 5x less power consumption

Next Steps and Conclusion

Next Steps

Aim: To produce a Complete ISR+FSR shower and measure speedups

- To combine with LHAPDF on GPU (work done by M. Knobbe)
- Followed by simulating LHC and DIS processes

Also: To develop this code as a *parton shower sandbox*

- For use in testing new shower kernels and kinematics
- As a starting point for anyone interested in GPU Hadronisation :)

Summary

Using modern GPU technology, individual steps of the veto algorithm can be run for many events together. This allows us to simulate parton showers in *parallel*.

While there are setbacks, we demonstrate a reduction in execution time whilst producing identical results.

Summary

Using modern GPU technology, individual steps of the veto algorithm can be run for many events together. This allows us to simulate parton showers in *parallel*.

While there are setbacks, we demonstrate a reduction in execution time whilst producing identical results.

This talk is based on “An Algorithm to Parallelise Parton Showers on a GPU” [2403.08692]. The program, GAPS (a GPU Amplified Parton Shower), can be accessed here:

`https://gitlab.com/siddharthsule/gaps`

Try it, test it, break it, critique it, and send us your feedback!

...And That's All!

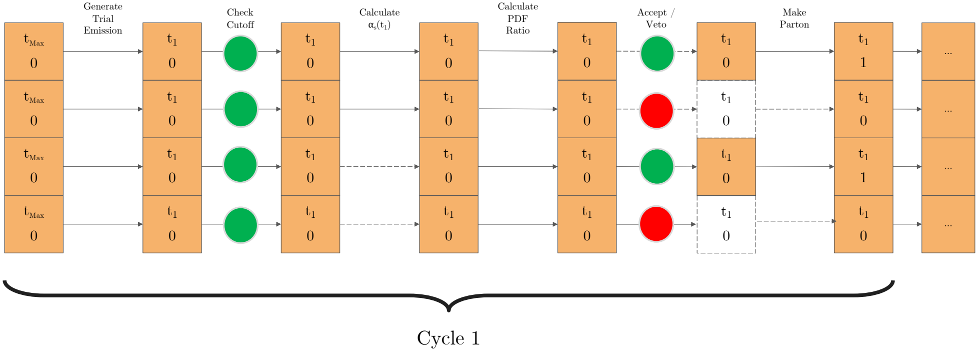
Thanks for Listening!

My details:

- siddharth.sule@manchester.ac.uk
- Office: Room 6.08, Schuster Building, Uni of Manchester, M13 9PL

Appendix Slides

The Paralleised Veto Algorithm - In Full



NVIDIA V100 GPU Diagram

Profiling

Name	**Instances**	**Total Time (ns)**	**Time (%)**
Selecting the Winner Emission	119	291,300,033	46.3
Device Prep.	1	105,578,119	16.8
Vetoing Process	119	45,518,957	7.2
Thrust	1	42,478,087	6.8
Durham Algorithm	1	26,657,439	4.2
Checking Cutoff	119	25,702,499	4.1
Doing Parton Splitting	119	23,646,846	3.8
Calculating $\boldsymbol{\alpha}_s$	119	20,654,227	3.3
Histogramming	1	17,950,803	2.9
Matrix Element	1	7,605,270	1.2
Set Up Random States	1	7,439,289	1.2
Jet Mass/Broadening	1	5,758,537	0.9
Validate Events	1	5,580,426	0.9
Prep Shower	1	2,651,686	0.4
Set Up $\boldsymbol{\alpha}_s$ Calculator	1	8,800	0.0
Pre Writing	1	5,856	0.0
Set Up ME Calculator	1	3,968	0.0

Unanswered Questions / To Do List

Can you combine some of your Kernels in your Diagram? How does it affect the Speed Up?

What about allowing the user to vary $\alpha_s(m_Z)$ and t_C?

S. Höche's Tutorial also contains Matching. Are you planning on implementing that?

Is it possible to connect this to something like PEPPER or MadGraph4GPU?

Any attempts to optimize the code?