

Madgraph5_aMC@NLO on GPUs and vector CPUs

Experience with the first alpha release

Stephan Hageböck^{1,*}, *Taylor Childers*², *Walter Hopkins*², *Olivier Mattelaer*³, *Nathan Nichols*², *Stefan Roiser*¹, *Jørgen Teig*¹, *Andrea Valassi*¹, *Carl Vuosalo*⁴, and *Zenny Wettersten*¹

¹CERN, Geneva, Switzerland

²Argonne National Laboratory, Illinois, USA

³Université Catholique de Louvain, Louvain, Belgium

⁴University of Wisconsin-Madison, Madison, USA

Abstract. Madgraph5_aMC@NLO is one of the most-frequently used Monte-Carlo event generators at the LHC, and an important consumer of compute resources. The software has been reengineered to maintain the overall look-and-feel of the user interface while speeding up event generation on CPUs and GPUs. The most computationally intensive part, the calculation of “matrix elements”, is offloaded to new implementations optimised for GPUs and for CPU vector instructions, using event-level data parallelism. We present the work to support accelerated leading-order QCD processes, and discuss how this work is going to be released to Madgraph5_aMC@NLO’s users.

1 Introduction

As a Monte-Carlo event generator, the Madgraph5_aMC@NLO framework (MG5_aMC) [1] stands at the beginning of the simulation chain for experiments at the Large Hadron Collider (LHC) [2] or other colliders. Event generation, and the subsequent simulation of the interaction of generated particles with the detectors account for almost half of the computing resources spent by the LHC experiments [3, 4]. With longer operation time of the LHC, the collision data recorded by the experiments is increasing, and a corresponding increase in simulated events is desirable to conduct high-precision analyses. At the same time, the computing resources of the Worldwide LHC Computing Grid (WLCG) cannot be increased at the same rate as the amount of recorded data is going to increase. A paradigm shift is needed for event simulation. By 2030, when the High-Luminosity LHC [5] is expected to be in operation, about 20 % of the computing resources are expected to be spent on event generation. This emphasises the importance of projects such as madgraph4gpu [6].

With the advent of SIMD-capable CPUs¹ and GPUs, event generators can be improved by increasing data parallelism. During the madgraph4gpu project, MG5_aMC has been extended to support SIMD computations and GPUs [7–9]. Matrix-element computations as they are executed by MG5_aMC lend themselves particularly well to data parallelism, since the functions to evaluate matrix elements are identical for every event – they are just run for

*e-mail: stephan.hageboeck@cern.ch

¹Single Instruction Multiple Data

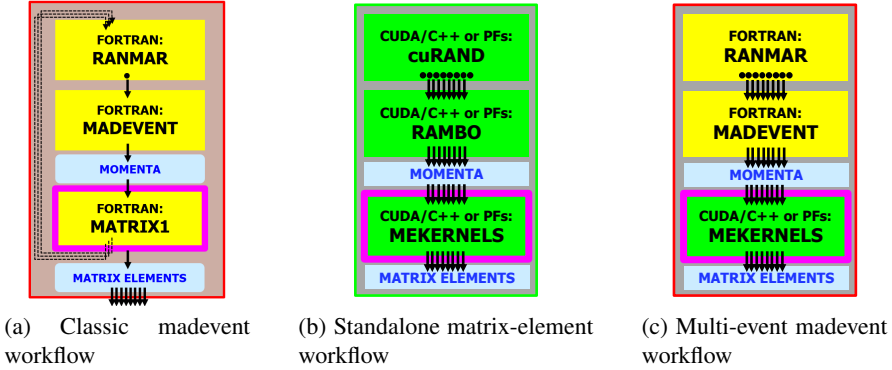


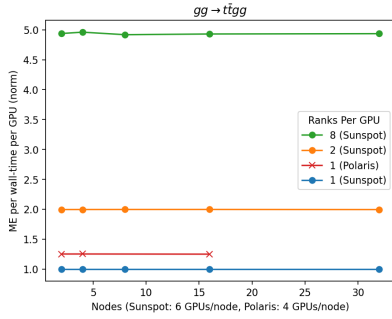
Figure 1: Evolution of the madevent program. (a) The classic madevent workflow ran random numbers, phase-space generation and matrix-element computation in a loop for one event at a time. (b) For madgraph4gpu, a testbed with a simple, multi-event random number and phase-space generator was designed to test the parallel computation of matrix elements in C++, CUDA, or other portability frameworks. (c) Multi-event madevent, where parallel matrix-element computations from (b) can be used to benefit from SIMD vectorisation, multiple cores, or GPUs.

different input data. In addition, the matrix element computations are almost branch free, so vector computations using SIMD or GPU computations with low thread divergence can be employed with high efficiency.

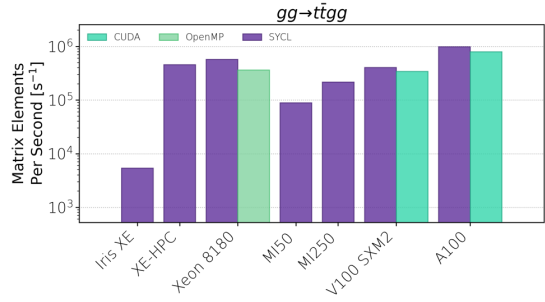
2 A GPU / SIMD backend for MG5_aMC

Madgraph5_aMC@NLO is a code generator to evaluate probability amplitudes, generate events, and compute cross sections. For a given particle collision such as $e^+e^- \rightarrow \mu^+\mu^-$ or $pp \rightarrow t\bar{t}gg$, etc, the Fortran program “madevent” is generated and compiled, which contains a phase-space generator, a phase-space integrator, and a way to compute probability amplitudes (“matrix elements”) for a given collision, see fig. 1a. The established MG5_aMC can generate the matrix-element code in Fortran, C++, and Python. The madgraph4gpu project is a plugin for MG5_aMC, where the C++ matrix-element code generated by the original MG5_aMC was converted to C++ and CUDA to evaluate probability amplitudes for batches of several thousands of events. It relies on SIMD computations on CPUs and the even higher data parallelism on GPUs.

In addition to madevent, the plugin can generate a standalone program that only evaluates matrix elements without the phase-space integration step of madevent (fig. 1b). It relies on the simple phase-space generator “RAMBO”, which is insufficient for LHC simulations, but ideal to quickly generate a batch of test events for throughput measurements. This program was used to design, test and improve the initial madgraph4gpu plugin [7]. During a normal collision simulation, the madevent program calls the same code as the standalone program, so the latter can be used to measure the throughput of different approaches, or optimise the computation of matrix elements for specific hardware. Using the standalone program, speedups of about 250 x against the single-threaded Fortran amplitudes were achieved on the moderately complex process $gg \rightarrow t\bar{t}gg$ using an NVidia Tesla V100 GPU in double precision mode, and 4.6 x to 8.3 x using AVX2 and AVX512 vectorisation in double precision. More



(a) Scaling behaviour of matrix-element throughput for multiple parallel invocations of the standalone program



(b) Throughput of the standalone program for different hardware

Figure 2: Throughput tests using the standalone program with a SYCL backend on the “Sunspot” testbed for the Aurora supercomputer at the Argonne National Laboratory. The program computes matrix elements for the process $gg \rightarrow t\bar{t}gg$.

details were shown on the ACAT 2022 conference [9]. For the complex process $gg \rightarrow t\bar{t}ggg$, the speedups are 3.6 x, 6.7 x and 130 x for AVX2, AVX512, and CUDA on the V100 GPU. This shows that a standalone computation of matrix elements can be sped up significantly using hardware accelerators, but for full event generation, these matrix elements have to be integrated into a larger framework. This will be analysed in section 3. In single-precision mode, the matrix-element throughput can theoretically be doubled, but tracking the precision of the matrix elements showed that single-precision computations are not accurate enough to reliably integrate the phase space.

2.1 Intel and AMD GPUs

Based on the CUDA backend, other backends based on portability frameworks such as SYCL [10], Kokkos [11], and Alpaka [12] were developed to test the feasibility of porting the matrix element code to different hardware. CUDA code can be compiled for AMD GPUs with a small amount of changes, but Intel GPUs would be out of reach. After initial progress with all backends, only the SYCL backend was continued due to limited resources. Figure 2 shows a scaling test and a throughput comparison on different hardware for the intermediate-complexity process $gg \rightarrow t\bar{t}gg$. When starting multiple instances of the standalone program on multiple nodes of a high-performance computer, the throughput should scale with the number of processes. This is confirmed in fig. 2a. Figure 2b shows that matrix-element throughputs of $1 \times 10^5 \text{ s}^{-1}$ to $1 \times 10^6 \text{ s}^{-1}$ can be achieved with various GPUs from AMD, Intel and NVidia. The classic Fortran matrix elements can be computed at a rate of $3 \times 10^3 \text{ s}^{-1}$, so a speedup by two orders of magnitude is achievable depending on the hardware used.

3 Analysis of the full Madevent Workflow

In parallel with the development of the GPU plugin for MG5_aMC, madevent was converted to a multi-event interface. Instead of generating random numbers, input and output particles, and the corresponding matrix element sequentially (event-by-event), random numbers and

Process	Matrix elm	Total	Momenta+ unweight	Matrix elm
$e^+e^- \rightarrow \mu^+\mu^-$	Fortran	9.93± 0.05s	9.75± 0.05s	0.185± 0.001s
	C++ AVX2	9.93± 0.02s 1.00± 0.01×	9.89± 0.02s 0.99± 0.01×	0.045± 0.001s 4.12 ± 0.02 ×
	Cuda Tesla A100	10.33± 0.02s 0.96± 0.01×	10.32± 0.02s 0.94± 0.01×	0.008± 0.001s 24.3 ± 0.4 ×
$gg \rightarrow t\bar{t}gg$	Fortran	106.6 ± 0.2 s	4.55± 0.01s	102.0 ± 0.2 s
	C++ AVX2	29.01± 0.05s 3.67± 0.01×	4.56± 0.01s 1.00± 0.01×	24.45 ± 0.04 s 4.17 ± 0.01 ×
	Cuda Tesla A100	5.78± 0.01s 18.44± 0.04×	4.87± 0.01s 0.93± 0.01×	0.91 ± 0.02 s 112.3 ± 2.1 ×
$gg \rightarrow t\bar{t}ggg$	Fortran	2233.6 ± 1.9 s	8.81± 0.07s	2224.8 ± 1.9 s
	C++ AVX2	697.2 ± 1.2 s 3.20± 0.01×	8.71± 0.01s 1.01± 0.01×	688.5 ± 1.2 s 3.23 ± 0.01 ×
	Cuda Tesla A100	27.78± 0.05s 80.40± 0.16×	9.12± 0.05s 0.97± 0.01×	18.66 ± 0.02 s 119.23 ± 0.14 ×

Table 1: Run times and speedup for different parts of the madevent executable for different processes. Madevent is invoked to produce 2^{18} weighted events, which are subsequently unweighted and written to disk. The Fortran parts of madevent such as phase-space sampling and unweighting run on the CPU, whereas matrix elements are either run on the CPU in Fortran, in C++ with AVX2 extensions, or on the GPU in CUDA. The speedup is measured in comparison to the full-Fortran madevent program. CPU: AMD EPYC 7313, GPU: NVidia Tesla A100

particles are now generated in batches of up to several thousand events. This enables off-loading matrix-element computations to the C++ backend with SIMD acceleration, or to the CUDA backend for computation on a GPU as shown in fig. 1a and fig. 1c in section 2. The phase-space integration logic, as well as unweighting (that is selecting events that are written to the output file) remain in madevent. Inevitably, this leads to serial sections in madevent, so the total achievable speedup will according to Amdahl’s law [13] be lower than the speedup achieved in the standalone program.

In the following, the integration of C++ or CUDA matrix elements into madevent will be studied in more detail. The madevent executable is specific to each process being simulated, but it is mostly the matrix-element part that changes for different processes. Before matrix elements on GPUs were implemented, madevent spent most CPU cycles in the matrix-element computation², so this part was naturally optimised by the MG5_aMC developers. The introduction of C++ and CUDA matrix elements speeds up the matrix-element step by about a factor 4 for C++ with AVX2 extensions and $> 100\times$ with CUDA, depending on the complexity of the process, so now the other parts of madevent also become relevant for the total run time, as shown in table 1. The table shows that for very simple processes with short matrix-element computations such as $e^+e^- \rightarrow \mu^+\mu^-$, the achievable speedup is limited. Due to an additional latency for transferring momenta and weights to a different computation backend, a speedup of the matrix elements doesn’t speed up the full program, because the matrix

²except for very simple processes

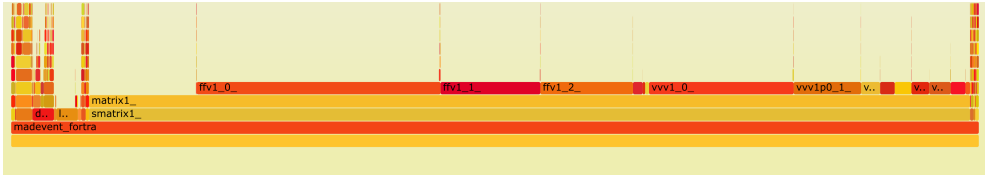
elements run for an extremely short time. For processes with intermediate complexity such as $gg \rightarrow t\bar{t}gg$, on the other hand, the speedup for matrix elements reaches $4\times$ for C++, and $112\times$ for CUDA. The matrix elements are a significant part of the total run time here, so the overall speedup reaches almost $4\times$ with C++ and $18\times$ with CUDA matrix elements. For processes with high complexity, the matrix elements strongly dominate the run time, so $3\times$ and $80\times$ total speedup are achieved for C++ and CUDA simulating the process $gg \rightarrow t\bar{t}ggg$. Due to the Fortran parts of madevent taking an almost constant time independent of the matrix-element backend used, the total-program speedup increases with the complexity of the matrix elements being simulated. This scaling is good for MG5_aMC's users, since especially the complex processes with slow matrix elements will be of interest.

The sequential parts of madevent that limit the speedup will be analysed in the following. Figure 3 shows where madevent for $gg \rightarrow t\bar{t}gg$ spends CPU cycles, depending on what type of matrix elements are used. The Fortran version spends most of the CPU cycles in the function `matrix1_`, fig. 3a, which computes the matrix elements. In fig. 3b on the other hand, the matrix elements are barely visible due to the CUDA acceleration. The matrix elements are computed in the deep call stack on the right of the figure, which is so narrow that function names could not be shown. With CUDA acceleration, the bulk of the run time is spent in unweighting (`unwgt_`), event I/O (`sample_put_point_`), phase-space sampling (`sample_full_`), and the evaluation of the Parton Distribution Functions (PDFs, `lh_polint_`).

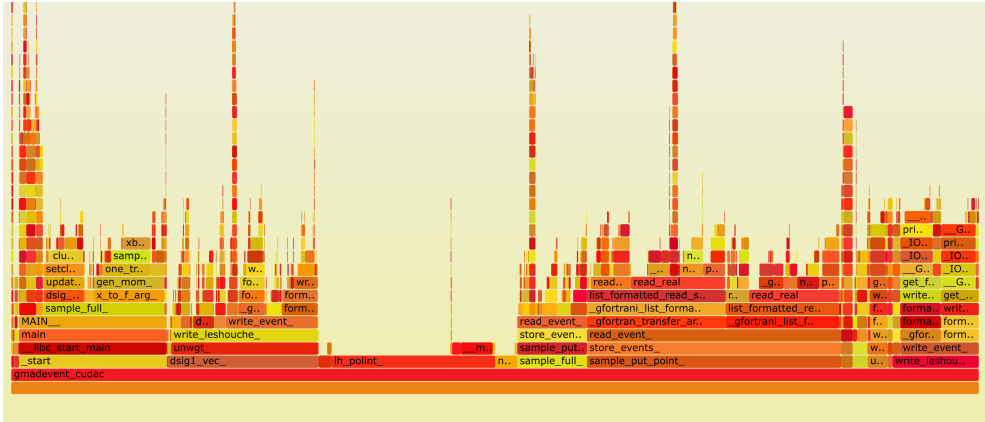
Analysing these parts further, it became evident that the madevent unweighting algorithm limits the total speedup, since it requires writing events to temporary storage if they cannot be discarded immediately. To sample unweighted events from a batch of weighted events, the maximum event weight of each batch needs to be known. However, since madevent was designed to iterate through an event sample one by one, it used to employ a running maximum. It therefore temporarily accepted events, and wrote them to temporary storage, but most of these were discarded later once the global maximum was known. These events were written to a file, and had to be read again once the decision could be made to accept or discard them (cf. `write_event_` and `read_event_` in fig. 3b).

Given that madevent has been converted to a multi-event interface to enable GPU and vectorised computations, c.f. fig. 1c, the maximum weight of a set of events can be computed in one go instead of using a running maximum. To do this, weights from Jacobian terms and PDFs are transferred to the GPU, and multiplied with the matrix elements that are already in GPU memory to compute the full event weight. The maximum of these weights is computed in parallel, and transferred back to the host. The knowledge of this weight improves the efficiency of the unweighting steps, as low-weight events can be discarded at a higher rate, and fewer events have to be written into or retrieved from temporary storage. For $gg \rightarrow t\bar{t}gg$, for example, the default unweighting of madevent temporarily stored 89 923 out of 278 506 events, but the final sample consisted of 870 events. Using the GPU-assisted unweighting, 1475 out of 278 506 were retained, and the final unweighted sample consisted of 1053 unweighted events. The amount of events that were stored temporarily but ultimately discarded was therefore reduced to 5%, and the number of events produced by madevent was increased. In table 2, the speedup by employing GPU matrix elements with batch unweighting is shown. In contrast to table 1, the zero-width t-channel mode is off, so absolute run times cannot be compared. By employing GPU-assisted unweighting, the slowdown that is normally incurred by transferring momenta from madevent to an accelerated matrix-element backend is compensated for, and a speedup of the momenta+unweight step is achieved.

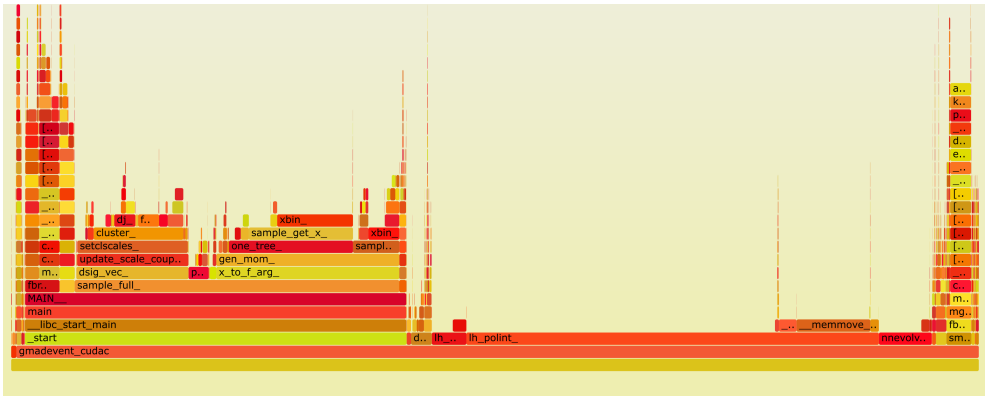
The impact on madevent is shown in fig. 3c. A large reduction of `write_event_` and `read_event_` calls is observed with respect to fig. 3b, and the madevent run time without the matrix-element part is reduced from 6.3 s to 4.7 s as shown in table 2. The number of events in temporary storage, unweighting efficiency, and the achievable speedup with GPU-assisted



(a) Fortran-only execution



(b) Fortran + CUDA execution



(c) Fortran + CUDA execution with GPU-assisted unweighting

Figure 3: Flamegraph analysis [14] of CPU cycles spent in three madevent runs for the process $gg \rightarrow t\bar{t}gg$. The fortran-only version **(a)** spends most of the time in the matrix-element computation (`matrix1_`). In **(b)**, this function is sped up about 170 x using a GPU. Now, phase-space sampling, unweighting, and PDF evaluation dominate the run time. In **(c)**, the unweighting step was improved in addition to using GPU matrix elements by employing more efficient, GPU-assisted unweighting. This results in a speedup of 1.3 x on the host side. The total run time is now dominated by phase-space sampling (`sample_full_`) and PDF evaluation (`lh_polint_`).

Process	Matrix elm	Total	Momenta+unweight	Matrix elm
$gg \rightarrow t\bar{t}gg$	Fortran	108.10± 0.27s	6.27± 0.41s	101.84± 0.14s
	C++ AVX2	31.08± 0.01s	6.88± 0.01s	24.20± 0.02s
		3.48± 0.01×	0.91± 0.06×	4.21± 0.01×
	Cuda Tesla A100	5.32± 0.03s	4.67± 0.02s	0.66± 0.02s
		20.32± 0.13×	1.34± 0.09×	155.4 ± 2.7 ×

Table 2: Run times and speedup for the madevent executable with batch unweighting. In contrast to table 1, the zero-width t-channel mode is deactivated, so absolute run times cannot be compared between the two tables. When GPU matrix elements are used, the unweighting of each batch of 16 384 events completes faster and with higher efficiency.

unweighting depend on the process being simulated, but in all cases, the direct computation of the maximum reduces the number of events to be stored.

Given that madevent can be used in a multi-event workflow also without GPUs, a similar unweighting strategy could be employed if Fortran and C++ matrix elements are used. This algorithmic improvement will be tested in madevent. It should be stressed that the main reason for a speedup is not the usage of the GPU to compute the maximum weight, but it is the knowledge of the maximum event weight for a larger batch of events. This leads to better accept/reject decisions, and fewer events are written or read to/from files.

With GPU matrix elements and GPU-assisted batch unweighting, the run times to compute $gg \rightarrow t\bar{t}gg$ are now dominated by the PDF evaluation (`lh_polint_`) and the phase-space sampling (`sample_full_`) steps as shown in fig. 3c. These could be reduced further, by e.g. employing accelerated PDF libraries or by reworking the phase-space algorithm of madevent. Currently, though, the focus of the madgraph4gpu project is on releasing the plugin for testing by the LHC experiments.

4 Summary and Outlook: Releasing madgraph4gpu to Madgraph5_aMC@NLO’s users

The madgraph4gpu plugin was shown to significantly speed up matrix-element computations, especially for processes with many final-state particles. These are computationally expensive to simulate, so this plugin should be a default choice for MG5_aMC if high statistics are required. Even if MG5_aMC users don’t have access to GPU resources, they can use the plugin to make use of SIMD computations, which are supported by all modern CPUs.

At the time of the conference, however, MG5_aMC users were not able to generate their own madevent executables accelerated by the madgraph4gpu plugin. First “gridpacks” had been produced by the madgraph4gpu developers with CUDA and vectorised C++ backends, but this required a non-standard MG5_aMC. For a gridpack, MG5_aMC pre-samples the phase space of a given process, and optimises the integration grid. This ensures that the sampling frequency of phase-space regions matches their probability density. This pre-sampled grid is written into files, and packed into archives together with the madevent executables for each subprocess. These archives can be used to run larger batch jobs that generate the desired number of collision events. For a few select processes, the ATLAS and CMS experiments were able to confirm that the gridpacks can benefit from GPU acceleration. The gridpacks were only tested – and equality of results against MG5_aMC verified – for a limited number of subprocesses such as $e^+e^- \rightarrow \mu^+\mu^-$ and $gg \rightarrow t\bar{t}0-3g$.

The madgraph4gpu plugin currently only supports leading-order QED and QCD processes, because running couplings in weak interactions and SUSY or BSM processes will require a more elaborate treatment in the GPU backend. These are on the plan of work. Processes with multiple subprocesses, such as proton-proton collisions $pp \rightarrow t\bar{t}0-2j$, that is none to two additional jets, are being tested currently.

Lastly, work is underway to improve the integration into Madgraph5_aMC@NLO. In the future, MG5_aMC users will be able to check out the madgraph4gpu plugin, and run the generation of the matrix-element code using the native MG5_aMC interface. We hope that this work reduces waiting times of MG5_aMC users, and helps to alleviate the pressure on scarce computing resources in light of growing datasets during Run 3 and the High-Luminosity LHC.

References

- [1] J. Alwall, R. Frederix, S. Frixione, V. Hirschi, F. Maltoni, O. Mattelaer, H.S. Shao, T. Stelzer, P. Torrielli, M. Zaro, *Journal of High Energy Physics* **7**, 79 (2014)
- [2] L. Evans, P. Bryant, *Journal of Instrumentation* **3**, S08001 (2008)
- [3] ATLAS Collaboration (ATLAS), Tech. rep., CERN, Geneva (2022), <https://cds.cern.ch/record/2802918>
- [4] CMS Offline Software and Computing, Tech. rep., CERN, Geneva (2022), <https://cds.cern.ch/record/2815292>
- [5] LHC Coordination, *Lhc long-term schedule* (2023), last accessed Aug 2023, <https://lhc-commissioning.web.cern.ch/schedule/LHC-long-term.htm>
- [6] Madgraph for GPU team, *madgraph4gpu* (2023), <https://github.com/madgraph5/madgraph4gpu>
- [7] Valassi, Andrea, Roiser, Stefan, Mattelaer, Olivier, Hageboeck, Stephan, *EPJ Web Conf.* **251**, 03045 (2021)
- [8] A. Valassi, T. Childers, L. Field, S. Hageboeck, W. Hopkins, O. Mattelaer, N. Nichols, S. Roiser, D. Smith, *Developments in Performance and Portability for MadGraph5_aMC@NLO*, in *Proceedings of 41st International Conference on High Energy physics — PoS(ICHEP2022)* (2022), Vol. 414, p. 212
- [9] A. Valassi, T. Childers, L. Field, S. Hageböck, W. Hopkins, O. Mattelaer, N. Nichols, S. Roiser, D. Smith, J. Teig et al., *J. Phys. Conf. Ser. ACAT2022* (2023), 2303.18244
- [10] The Khronos SYCL Working Group, *Sycl 2020 specification* (2023), last accessed Sept 2023, <https://registry.khronos.org/SYCL/specs/sycl-2020/html/sycl-2020.html>
- [11] H. Carter Edwards, C.R. Trott, D. Sunderland, *Journal of Parallel and Distributed Computing* **74.12**, 3202–3216 (2014)
- [12] E. Zenker, B. Worpitz, R. Widera, A. Huebl, G. Juckeland, A. Knüpfer, W.E. Nagel, M. Bussmann (2016), 1602.08477
- [13] G.M. Amdahl, *Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities*, in *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference* (Association for Computing Machinery, New York, NY, USA, 1967), AFIPS '67 (Spring), p. 483–485, ISBN 9781450378956, <https://doi.org/10.1145/1465482.1465560>
- [14] B. Gregg, *Blazing Performance with Flame Graphs* (USENIX Association, Washington, D.C., 2013), <https://www.usenix.org/conference/lisa13/technical-sessions/plenary/gregg>