EPJ Web of Conferences **337**, 01141 (2025) https://doi.org/10.1051/epjconf/202533701141
CHEP 2024

Experience with the alpaka performance portability library in the CMS software

Jaafar Alawieh[1], *Kareen* Arutjunjan[1], *Miguel* Bacharov Durasov[1], *Ulf* Behrens[2], *Andrea* Bocci[1,*], *James* Branson[3], *Philipp Maximilian* Brummer[1], *Jan Andrzej* Bugajski[1], *Eric* Cano[1], *Sergio* Cittolin[3], *Albert* Corominas I Mariscot[1], *Georgiana-Lavinia* Darlea[4], *Christian* Deldicque[1], *Marc* Dobson[1], *Antonin* Dvorak[1], *Christos* Emmanouil[1], *Antra* Gaile[1], *Dominique* Gigi[1], *Frank* Glege[1], *Guillelmo* Gomez-Ceballos[4], *Patrycja* Gorniak[1], *Jeroen* Hegeman[1], *Guillermo* Izquierdo Moreno[1], *Thomas Owen* James[1], *Tejeswini* Jayakumar[1], *Wassef* Karimeh[1], *Matti J.* Kortelainen[6,**], *Rafał* Krawczyk[2], *Wei* Li[2], *Kenneth* Long[4], *Frans* Meijers[1], *Emilio* Meschi[1], *Srećko* Morović[3], *Babatunde John* Odetayo[1,5], *Luciano* Orsini[1], *Christoph* Paus[4], *Andrea* Petrucci[3], *Marco* Pieri[3], *Dinyar Sebastian* Rabady[1], *Attila* Racz[1], *Wahid* Redjeb[1,7,***], *Theodoros* Rizopoulos[1], *Hannes* Sakulin[1], *Christoph* Schwick[1], *Dainius* Šimelevičius[1,8], *Polyneikis* Tzanis[1], *Cristina* Vazquez Velez[1], and *Petr* Žejdl[1]

[1]CERN, Geneva, Switzerland
[2]Rice University, Houston, Texas, USA
[3]UCSD, San Diego, California, USA
[4]MIT, Cambridge, Massachusetts, USA
[5]University of Benin, Benin City, Nigeria
[6]Fermilab, Batavia, Illinois, USA
[7]RWTH Aachen University, Aachen, Germany
[8]Vilnius University, Vilnius, Lithuania

Abstract. To achieve better computational efficiency and exploit a wider range of computing resources, the CMS software framework (CMSSW) has been extended to offload part of the physics reconstruction to NVIDIA GPUs. To support additional back-ends, as well to avoid the need to write, validate and maintain a separate implementation of the reconstruction algorithms for each back-end, CMS has adopted the alpaka performance portability library.

Alpaka (Abstraction Library for Parallel Kernel Acceleration) is a header-only C++ library that provides performance portability across different back-ends, abstracting the underlying levels of parallelism. It supports serial and parallel execution on CPUs, and extremely parallel execution on NVIDIA, AMD and Intel GPUs.

This contribution will show how alpaka is used in the CMS software to develop and maintain a single code base; to use different toolchains to build the code for each supported back-end, and link them into a single application; to seamlessly select the best back-end at runtime, and implement portable reconstruction algorithms that run efficiently on CPUs and GPUs from different vendors. It will describe the validation and deployment of the alpaka-based implementation in the CMS High Level Trigger, and highlight how it achieves near-native performance.

*Corresponding author e-mail: andrea.bocci@cern.ch
**M. J. Kortelainen is supported by FermiForward Discovery Group, LLC under Contract No. 89243024CSC000002 with the U.S. Department of Energy, Office of Science, Office of High Energy Physics.
***This work has been sponsored by the Wolfgang Gentner Programme of the German Federal Ministry of Education and Research (grant no. 13E18CHA)

1 Introduction

Since the beginning of the LHC Run 3 data taking period (2022-2026), the Compact Muon Solenoid (CMS) experiment at CERN has deployed a heterogeneous computing farm for the High Level Trigger (HLT) online physics reconstruction and event filter[1]. Before the start of the 2022 data taking, the HLT farm from Run 2 was replaced with 200 nodes, each equipped with two AMD EPYC "Milan" 7763 CPUs and two NVIDIA T4 GPUs. Before the start of the 2024 data taking, the processing power of the HLT farm was increased by about 20%, with the addition of 18 nodes, each equipped with two AMD EPYC "Bergamo" 9754 CPUs and three NVIDIA L4 GPUs - bringing the total to over 30'000 CPU cores and 450 GPUs.

This choice was initially motivated by the expectation that building a CPU-only filter farm for the High Luminosity LHC (HL-LHC) would not be feasible[2], and the Trigger Studies Group (TSG) and Data Acquisition (DAQ) groups deemed wise to start getting experience with the use of GPUs in a real time environment during Run 3, when their use would be beneficial but not critical to guarantee the successful operation of the experiment.

The CMS simulation, reconstruction and online selection software (CMSSW) consists of a modular framework and thousands of algorithms, written in C++ and implemented as objects (so-called "modules") that can be configured and loaded at run-time via a plug-in mechanism. The framework then schedules these algorithms based on their dependencies, and executes them on multiple CPU threads, processing one or more events concurrently. During the preparation for Run 3 the CMSSW framework was extended to support the asynchronous execution of algorithms on NVIDIA GPUs. Various algorithms were rewritten using the CUDA toolkit - the pixel local reconstruction, the pixel-only track and vertex reconstruction, and the electromagnetic and hadronic calorimeters local reconstruction - achieving an offload of about 40%, significantly better than the threshold of 25% that was required to break even between the cost of a CPU-only farm and a GPU-equipped one.

The deployment of the CUDA-based GPU accelerated software and its validation, while successful, introduced additional complexities in the development and maintenance of the CMSSW code base and of the HLT configuration: most of the algorithms used two different data structures and had two separate, independent implementations - a legacy one running only on CPU and a novel one running only on NVIDIA GPUs; in some cases the two implementations also had different interfaces, with the overall reconstruction split across modules in different ways. In order to support running with or without GPUs, the HLT configuration had to include both set of modules, along with "switch" points to direct the execution to the legacy CPU or GPU version of the modules[2, 3].

In order to address these issues, in parallel to the deployment of the CUDA-based software, the TSG and Offline and Computing (O&C) groups decided to evaluate different options for *performance portability*, with the goal of having a single code base that could be compiled and executed on different back-ends: CPUs, GPUs from different vendors (NVIDIA, AMD, eventually Intel), and possibly other kind of accelerators. After a thorough evaluation of OpenMP, alpaka[4–6], Kokkos[7], SYCL, and solutions based on standard C++, the alpaka library appeared as the more mature and better performing solution[8–10].

To simplify the development of efficient data structures for code running on GPUs, the DAQ and TSG groups developed a generic Structure of Array (SoA) approach, optimised to minimise data transfers and take advantage of coalesced access to GPU memory[11]. At the same time, the O&C Core Software group developed a more efficient approach to execute asynchronous operations on a GPU[12].

With the goals to reduce code complexity, avoid multiple implementation of the same algorithms, ease the execution and validation of the algorithms on different back-ends, and simplify the configuration of the HLT application, from 2022 to 2024 the work shifted to the

```
DataFormats/PortableTestObjects/
├── BuildFile.xml
├── README.md
├── interface/
│   ├── alpaka/
│   │   └── TestDeviceCollection.h
│   ├── TestHostCollection.h
│   └── TestSoA.h
└── src/
    ├── alpaka/
    │   ├── classes_cuda.h
    │   ├── classes_cuda_def.xml
    │   ├── classes_rocm.h
    │   └── classes_rocm_def.xml
    ├── classes.h
    └── classes_def.xml
```

```
HeterogeneousCore/AlpakaTest/
├── interface/
│   └── TestCommon.h
├── plugins/
│   ├── BuildFile.xml
│   ├── TestAlpakaAnalyzer.cc
│   ├── alpaka/
│   │   ├── TestAlgo.dev.cc
│   │   ├── TestAlgo.h
│   │   └── TestAlpakaProducer.cc
└── test/
    ├── BuildFile.xml
    ├── reader.py
    ├── testWriterReader.sh
    └── writer.py
```

Figure 1. Directory and file structure of an alpaka-based data structures library (left) and a plug-in package (right). Files under the new `alpaka/` subdirectories are compiled multiple times, once for each back-end. `.cc` files are compiled by the host compiler, while `.dev.cc` files are compiled by the back-end's device compiler.

"alpaka migration": rewriting CUDA code to take advantage of the central SoA approach, the new asynchronous execution framework, and the alpaka *performance portability* library.

2 Integration of alpaka in the CMS software

Alpaka has been integrated in the CMS software stack in three main areas: the build system, data structures, and the plug-in system.

The build system has been extended to support multiple alpaka back-ends: serial execution on the host CPUs, offloading to NVIDIA GPUs, and AMD GPUs. Support for parallel execution on the CPUs and for additional back-ends, like Intel GPUs, is under development. A dedicated directory structure and a new file type were introduced to separate traditional code running on the CPU from alpaka-based code, as shown in figure 1: files under the new `alpaka/` subdirectories are compiled multiple times, once for each back-end. The resulting binaries are linked together into a back-end specific shared library or plug-in. C++ files with the regular `.cc` extension are compiled multiple times by the host compiler, while C++ files with the newly introduced `.dev.cc` extension are compiled each time by the back-end specific compiler: `gcc`, `nvcc` or `hipcc`. To prevent naming conflicts, this code is either templated on back-end specific types or defined in a back-end specific namespace.

The generic SoA data structures used in CMSSW are agnostic of the memory space where they reside: pageable or pinned host memory, GPU global or shared memory, *etc*. They have been coupled with alpaka memory buffers and memory operations, resulting in "portable collections" with a uniform interface, that can be trivially copied between the host memory and a GPU memory, used by code running on the CPUs or GPUs, and written to and read back from ROOT files.

Integration within the CMSSW framework has taken place in multiple places: the asynchronous execution support has been rewritten using alpaka calls and polling for alpaka-based events; the plug-in mechanism has been extended so that a module declared in the configuration as `module@alpaka` automatically uses the best back-end available for `module`; asynchronous copies of the alpaka-based data structures to and from GPU memory can automatically be scheduled by the framework. Together these features noticeably simplify the configuration of a heterogeneous application: the user can request a mix of CPU-only and

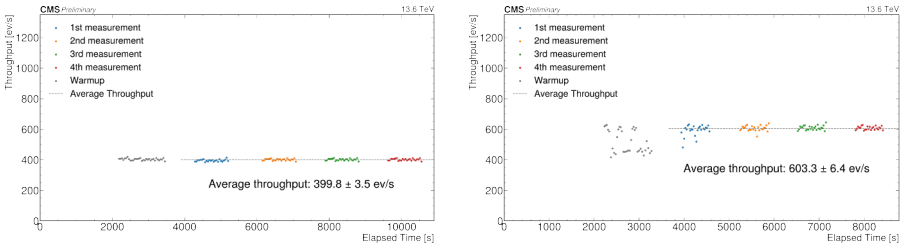


Figure 2. Event processing throughput measured on a 2022 HLT node as described in section 3. The values that are actually used are marked by coloured dots. The left plot (a) shows the throughput of the HLT application running only on CPUs. The right plot (b) shows the throughput of the HLT application offloading part of the reconstruction to GPUs.

alpaka-based modules, and the framework will automatically schedule the alpaka-based ones on a GPU when one is available, and take care of copying the data between the host and the GPU as needed.

The adoption of alpaka in CMS has been beneficial also for the alpaka library itself: many of the features for simplifying writing heterogeneous kernels in CMSSW have later been contributed to alpaka itself, and work is in progress to further contribute various optimisations, like memory and event caching.

3 Performance of the High Level Trigger

The performance of the HLT was measured running on a 2022 HLT node equipped with two AMD EPYC "Milan" 7763 CPUs and two NVIDIA T4 GPUs, running the 2024 HLT configuration over 13.6 TeV proton-proton collisions data with an average pileup of 60. Each measurement consists of 8 jobs running in parallel, each with 32 threads, in order to saturate all CPU cores on the machine. Each job runs over 120 000 events; the first 20 000 events are not included in the measurement, to let the system reach stable thermal and running conditions. Each measurement is repeated five times, ignoring the first one to ensure the data and application are cached. The event processing time and throughput are then averaged over the remaining four measurements.

Figure 2 shows the event processing throughput for the HLT application running only on CPU (a), 400 ± 4 ev/s, and offloading part of the reconstruction to GPUs (b), 603 ± 6 ev/s, with an increase of $50.9\% \pm 0.7\%$. Figure 3 shows the distribution of the event processing time grouped by the main detector or reconstruction algorithm when running only on CPU (a), while (b) highlights the part of the reconstruction written using the alpaka library, that can be offloaded to GPUs: the pixel local reconstruction, the pixel-only track and vertex reconstruction, the electromagnetic and hadronic calorimeters local reconstruction, and part of the particle flow reconstruction, spanning about one third of the HLT processing time. Figure 3 (c) shows the timing distribution when running also on GPUs, and (d) summarises the gains in event processing time: from 638 ms per event running only on CPUs to 421 ms per event running also on GPUs, a reduction of about 34%.

4 Power efficiency

The reduction in event processing time and the corresponding increase in throughput, however, is only part of the improvement achieved by a heterogeneous computing farm equipped

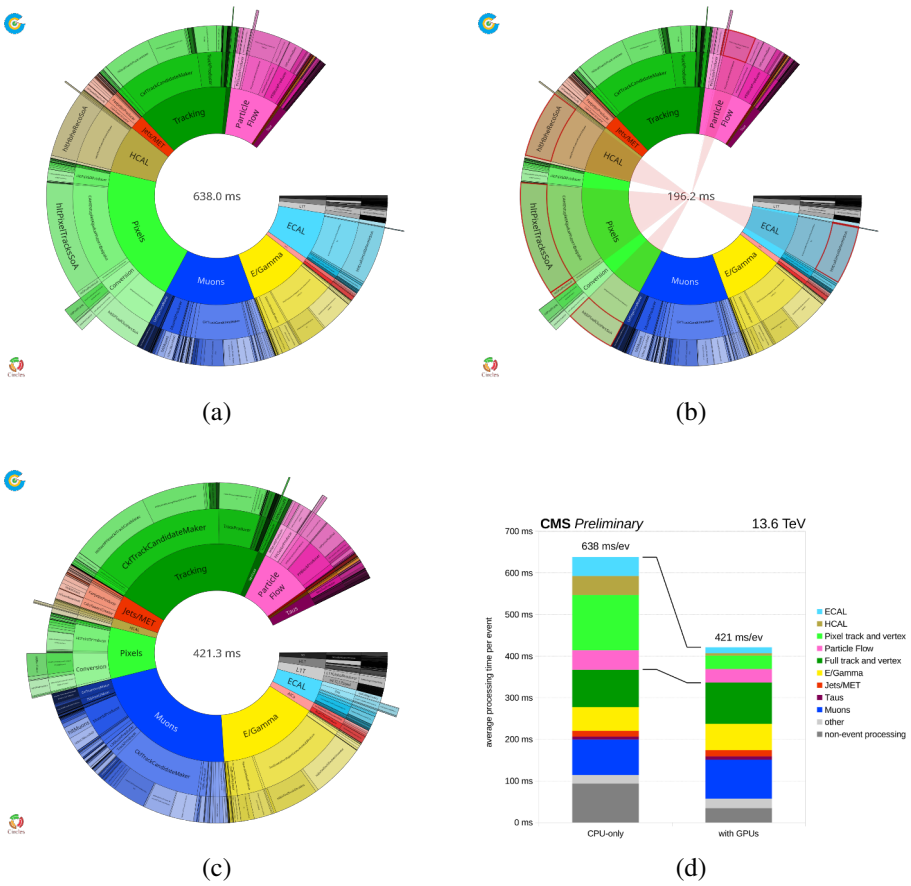
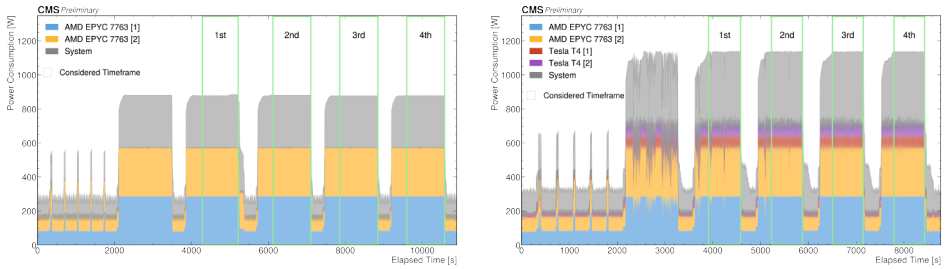


Figure 3. Distribution of the event processing time for the 2024 HLT menu running on a 2022 HLT node as described in section 3. The innermost ring indicates the detector or high-level object being processed, optionally accompanied by an extra ring that indicates GPU processing or the conversion of data structures to legacy format; the next ring indicates the C++ module; the outermost ring indicates the instance of the module, that is the concrete algorithms being run. The top left plot (a) shows the timing of the HLT application running only on CPUs. The top right plot (b) highlights the part of the HLT application that can be offloaded to GPUs. The bottom left plot (c) shows the timing of the HLT application offloading part of the reconstruction to GPUs. The bottom right plot (d) shows the comparison between the timing of the CPU-only and GPU-enabled HLT applications.

with GPUs. The other improvement is the reduction in power consumption for a given processing throughput, *i.e.* the energy cost of the processing a fixed amount of data, measured in Joules per event.

To assess the impact of the use of GPUs on the overall power consumption the measurements described in the previous section were instrumented to measure the power drawn by the CPUs, by the GPUs, and by the whole system: the CPU power usage is measured via the `/sys/class/powercap/intel-rapl` interface of the Linux kernel, the GPU power usage is measured via the `nvidia-smi` command, and the total system power usage is measured directly by a smart power distribution unit. When measuring the performance and power



(a) (b)

Figure 4. Power consumption measured on a 2022 HLT node as described in sections 3 and 4. The values that are actually used are marked by a green rectangle. The left plot (a) shows the power consumption measured on a CPU-only machine, where the full HLT application ran only on CPU. The left plot (b) shows the power consumption on a GPU-equipped machine, where about 34% of the HLT application was offloaded to GPUs.

Table 1. Power consumption and event processing throughput measured on a 2022 HLT node equipped with two AMD EPYC "Milan" 7763 CPUs and zero or two NVIDIA T4 GPUs as described in section 3.

GPUs	Power consumption	Event throughput
0	879.8 ± 1.4 W	339.8 ± 3.5 ev/s
2	1135.5 ± 1.9 W	603.3 ± 6.4 ev/s

consumption of the CPU-only configurations, the GPUs were physically removed from the server.

The measurements from figures 2 and 4 are summarised in table 1: the use of GPUs raises the power consumption from 880 ± 1 W to 1136 ± 2 W, but increases the throughput from 340 ± 4 ev/s to 603 ± 6 ev/s, improving the energy efficiency from 2.59 ± 0.03 J/ev to 1.88 ± 0.11 J/ev, a 37% improvement.

To study the possible gains from a heavier use of GPUs, the HLT application can be customised to run only the alpaka-based part and used as a benchmark to compare the energy efficiency of GPUs and CPUs for a realistic workflow: this approach runs the same reconstruction algorithms either entirely on CPUs, or almost entirely on GPUs. The small amount of CPU processing power used by the GPU-based application can be taken into account running the GPU-based application in parallel to the CPU-only one, and measuring the increase in the total power consumption and event processing throughput. This measurement was performed on one of the 2024 HLT nodes, equipped with two AMD EPYC "Bergamo" 9754 CPUs and up to four NVIDIA L4 GPUs.

The CPU-only baseline performance was measured as described in the previous section, running enough copies of the modified HLT application in parallel to fully saturate the CPUs. The measurement is then repeated after adding one GPU to the system, running in parallel to the CPU-only applications enough GPU-based applications to saturate the GPU, and measuring the increase in power consumption and total event processing throughput. Then, the measurements are repeated adding a second, third and fourth GPU, until the machine is fully populated.

The results are summarised in table 2 and shown in figure 5: saturating the two AMD EPYC "Bergamo" 9754 CPUs achieves a throughput of 599.1 ± 1.9 ev/s with a power consumption of 525.0 ± 0.2 W, resulting in a processing cost of 0.876 ± 0.003 J/ev; on average

Table 2. Power consumption and event processing throughput for a 2024 HLT node equipped with two AMD EPYC "Bergamo" 9754 CPUs and up to four NVIDIA L4 GPUs, running enough modified alpaka-only HLT benchmark applications to saturate the CPUs and GPUs.

GPUs	Power consumption	Event throughput
0	1050.0 ± 0.5 W	1198.2 ± 3.8 ev/s
1	1133.7 ± 0.6 W	1690.5 ± 8.4 ev/s
2	1257.5 ± 2.9 W	2229.5 ± 6.1 ev/s
3	1342.7 ± 2.9 W	2764.7 ± 2.7 ev/s
4	1456.1 ± 5.0 W	3222.7 ± 8.4 ev/s

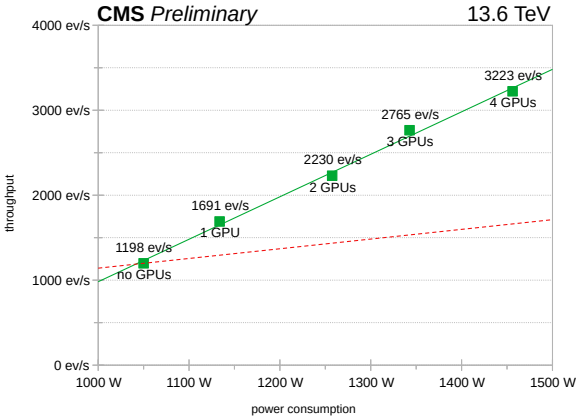


Figure 5. Event processing throughput as a function of the power consumption for a 2024 HLT node equipped with two AMD EPYC "Bergamo" 9754 CPUs and up to four NVIDIA L4 GPUs, running enough modified alpaka-only HLT benchmark applications to saturate the CPUs and GPUs. For reference, the red dashed line shows the event processing throughput of the CPU-only configuration scaled linearly with the power consumption.

each NVIDIA L4 GPU adds a throughput of 512.3 ± 8.0 ev/s with a power consumption of 102.1 ± 3.4 W, resulting in a processing cost of 0.199 ± 0.007 J/ev - a factor four better than the CPU-only case.

These results validate the assumption made in [2] that offloading a large part of the HLT processing to GPUs can substantially reduce the power and cooling requirements of the online filter farm.

5 Conclusions

The deployment of a heterogeneous HLT farm in Run 3 has allowed CMS to increase the online data processing rate by 50%, while improving its energy efficiency by 37%.

The adoption of the alpaka *performance portability* library and its integration in the CMS build system and software framework has contributed the simplification of the development, validation and maintenance of the algorithms running on CPUs and GPUs, the configuration of the HLT application, and the overall adoption of GPUs in CMS.

This work is still ongoing: rewriting more of the online and offline reconstruction to run on GPUs, targeting both the closing years of Run 3 and the future HL-LHC data taking

periods; validating the support for AMD GPUs, in order to reduce the vendor lock-in and leverage resources like the LUMI HPC; and adding support for more alpaka back-ends to CMSSW, like SYCL for CPUs and Intel GPUs, and possibly Altera FPGAs.

Additional measurements and results based on newer HLT nodes can be found in [13].

References

[1] CMS Collaboration, Development of the CMS detector for the CERN LHC Run 3, JINST **19**, P05064 (2024), CMS-PRF-21-001, CERN-EP-2023-136, https://cds.cern.ch/record/2870088, arXiv:2309.05466. doi:10.1088/1748-0221/19/05/P05064

[2] CMS Collaboration, The Phase-2 Upgrade of the CMS Data Acquisition and High Level Trigger (2021), CERN-LHCC-2021-007, CMS-TDR-022, https://cds.cern.ch/record/2759072.

[3] A. Bocci, D. Dagenhart, V. Innocente, C. Jones, M. Kortelainen, F. Pantaleo, M. Rovere, on behalf of the CMS Collaboration, Bringing heterogeneity to the cms software framework, EPJ Web Conf. **245**, 05009 (2020). doi:10.1051/epjconf/202024505009

[4] B. Worpitz, Master thesis, Technische Universität Dresden (2015), http://dx.doi.org/10.5281/zenodo.49768

[5] E. Zenker, B. Worpitz, R. Widera, A. Huebl, G. Juckeland, A. Knüpfer, W.E. Nagel, M. Bussmann, Alpaka - An Abstraction Library for Parallel Kernel Acceleration (IEEE Computer Society, 2016), 1602.08477, http://arxiv.org/abs/1602.08477

[6] A. Matthes, R. Widera, E. Zenker, B. Worpitz, A. Huebl, M. Bussmann, Tuning and optimization for a variety of many-core architectures without changing a single line of implementation code using the Alpaka library (2017), 1706.10086, https://arxiv.org/abs/1706.10086

[7] C.R. Trott, D. Lebrun-Grandié, D. Arndt, J. Ciesko, V. Dang, N. Ellingwood, R. Gayatri, E. Harvey, D.S. Hollman, D. Ibanez et al., Kokkos 3: Programming model extensions for the exascale era, IEEE Transactions on Parallel and Distributed Systems **33**, 805 (2022). doi:10.1109/TPDS.2021.3097283

[8] A. Bocci, A. Czirkos, A.D. Pilato, F. Pantaleo, G. Hugo, M. Kortelainen, W. Redjeb, on behalf of the CMS Collaboration, Performance portability for the CMS reconstruction with alpaka, Journal of Physics: Conference Series **2438**, 012058 (2023). doi:10.1088/1742-6596/2438/1/012058

[9] M.J. Kortelainen, M. Kwok, T. Childers, A. Strelchenko, Y. Wang, on behalf of the CMS Collaboration, Porting CMS heterogeneous pixel reconstruction to Kokkos, EPJ Web Conf. **251**, 03034 (2021). doi:10.1051/epjconf/202125103034

[10] N. Andriotis, A. Bocci, E. Cano, L. Cappelli, T. Di Pilato, L. Ferragina, G. Hugo, M.J. Kortelainen, M. Kwok, J.J. Olivera Loyola et al., Evaluating performance portability with the CMS heterogeneous pixel reconstruction code, EPJ Web of Conf. **295**, 11008 (2024). doi:10.1051/epjconf/202429511008

[11] E. Cano, on behalf of the CMS Collaboration, Tech. rep., CERN, Geneva (2023), https://cds.cern.ch/record/2872252

[12] A. Bocci, C. Jones, M.J. Kortelainen, Performance of heterogeneous algorithm scheduling in cmssw, EPJ Web of Conf. **295**, 11017 (2024). doi:10.1051/epjconf/202429511017

[13] CMS Collaboration, 2024 HLT timing and throughput results (2024), CMS-DP-2024-082, all plots and results can be found at https://twiki.cern.ch/twiki/bin/view/CMSPublic/DP2024082.