Title
Portability: A Necessary Approach for Future Scientific Software

Authors
Bhattacharya, Meghna
Calafiura, Paolo
Childers, Taylor
et al.

Publication Date
2022-03-15

March 21, 2022

Portability: A Necessary Approach for Future Scientific Software

Meghna Bhattacharya[3], Paolo Calafiura[2], Taylor Childers[4], Mark Dewing[4], Zhihua Dong[1], Oliver Gutsche[3], Salman Habib[4], Xiangyang Ju[2], Michael Kirby[3], Kyle Knoepfel[3], Matti Kortelainen[3], Martin Kwok[3], Charles Leggett[2], Meifeng Lin[1], Vincent R. Pascuzzi[1], Alexei Strelchenko[3], Brett Viren[1], Beomki Yeo[2], Haiwang Yu[1]

[1]*Brookhaven National Laboratory, Upton, NY 11973, USA*

[2]*Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA*

[3]*Fermi National Accelerator Laboratory, Batavia, IL 60510, USA*

[4]*Argonne National Laboratory, Lemont, IL 60439, USA*

ABSTRACT

Today's world of scientific software for High Energy Physics (HEP) is powered by x86 code, while the future will be much more reliant on accelerators like GPUs and FPGAs. The portable parallelization strategies (PPS) project of the High Energy Physics Center for Computational Excellence (HEP/CCE) is investigating solutions for portability techniques that will allow the coding of an algorithm once, and the ability to execute it on a variety of hardware products from many vendors, especially including accelerators. We think without these solutions, the scientific success of our experiments and endeavors is in danger, as software development could be expert driven and costly to be able to run on available hardware infrastructure. We think the best solution for the community would be an extension to the C++ standard with a very low entry bar for users, supporting all hardware forms and vendors. We are very far from that ideal though. We argue that in the future, as a community, we need to request and work on portability solutions and strive to reach this ideal.

Submitted to the Proceedings of the US Community Study on the Future of Particle Physics (Snowmass 2021)

Today's world of scientific software for High Energy Physics (HEP) is powered by x86 code. In today's research environment, code that is written for the x86 platform is pretty much guaranteed to run everywhere in the world, from computing centers using batch systems to our own laptops. Through the proliferation of x86 hardware the challenge to write scientific code is reduced to how well and efficiently the code can be engineered for the platform.

But we already see signs that the world is changing. On the High Performance Computing level for sure, where large installations of hardware using GPUs and other accelerators provide more processing power for the same energy consumption as with x86-based supercomputers. This is continued even to the hardware in our computers and laptops, that are starting to move to System-on-a-chip (SoC) architectures as lately Apple with the M1 processor. And the increase in heterogeneity does not stop there, multiple different GPU vendors and CPU vendors are available to optimize the hardware to our research problems.

This makes the challenge of writing efficient scientific code and getting the science out a lot more difficult. It can lead to designing algorithms and implementing them for specific hardware platforms and combinations, and making it very difficult to use not only one but several of these platforms. This adds a new constraint to getting the science out. It is not anymore enough to have access to sufficient computational power, it also needs to be the correct architecture(s) for which the code was developed.

We argue that it would be easier if researchers could develop scientific software and then could execute it on many different hardware combinations without having to rewrite the code over and over again. This white paper is arguing for solutions that enable the writing of portable code by describing one of the current projects to investigate such technologies.

1 The Portable Parallelization Strategies project

The High Energy Physics Center for Computational Excellence (HEP/CCE) is a pilot project whose mandate is to provide strategies for HEP experiments to adapt to using increasingly heterogeneous High Performance Computers. It is split into 4 parts, targeting portable parallelization strategies (PPS), fine-grained I/O and related storage issues (IOS), event generators (EG), and complex workflows (CW) [1]. The PPS group is investigating various portability solutions that will permit single source code to be compiled for and executed on multiple different heterogeneous architectures. This is becoming an essential requirement, as each of these architectures use different languages and APIs, such as CUDA [2] for NVIDIA GPUs, SYCL [3] for Intel GPUs, HIP [4] for AMD GPUs, and HLS [5] for FPGAs (see Fig. 1). HEP experiments, which now have code bases in the million lines of source code, do not have the person power to port their CPU code to each back end. Furthermore, validating and maintaining multiple versions of algorithms written in different languages would be exceedingly onerous.

The HEP/CCE-PPS group is currently evaluating Kokkos [6, 7], SYCL [3], Alpaka [8, 9, 10], OpenMP/OpenACC [11] and `std::execution::parallel` [12] by porting a small

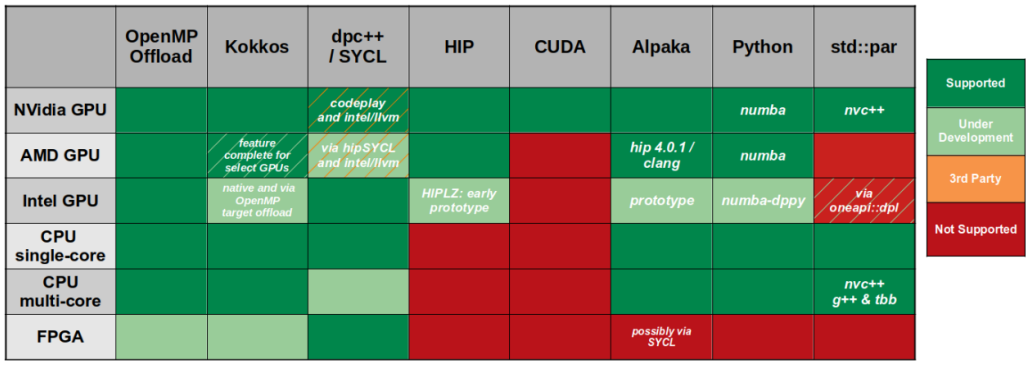

Figure 1: Matrix of portability technologies supporting a variety of hardware architectures.

number of HEP test beds taken from several different experiments to each portability layer (see Sec. 3). These are

- Patatrack and P2R from CMS [13] which perform pixel detector pattern recognition and tracking
- The WireCell toolkit from DUNE [14] which performs space point formation in the liquid argon time projection chamber (TPC)
- FastCaloSim from ATLAS [15] which does a fast parameterized simulation of the liquid argon calorimeter
- A pixel detector tracking workflow from the "A Common Tracking Software" project (ACTS) [16]

The HEP/CCE-PPS group is tightly integrated with a number of HEP experiments, with core developers from each experiment being represented in the group. Each port will be evaluated according to a set of metrics (see Sec. 2), and at the end of the process, the group will make recommendations back to the experiments and the HEP community in general as to the suitability of each technology that was investigated. It should be noted that no one best solution is likely to exist, as the needs and characteristics of each experiment are different.

2 Metrics

With the goal to assist the process to make recommendations to HEP experiments and the HEP community, we designed a set of metrics that are of interest to the HEP communities and how scientific software is being developed, and evaluate all portability technologies in question using this set of properties. The metrics set aims at evaluating the whole programming experience for the developer/user using the portability solution, not just the

specification or the capability of the solution. Hence we collected HEP use-case programs (see Sec. 3) for portability solutions from different sub-fields and implement them in different portability technologies. We get hands-on experience for applying a certain technology to HEP software problems by implementing our use cases in the portability technologies under consideration. This includes building, debugging, and adapting existing code to a given technology, which will be reflected in the evaluation of the metrics.

The metrics set will serve as a point-of-reference for the information about these portability solutions, which more often still lack the needed level of documentation, and help the HEP community make the informed decision when choosing the portability solution to work with.

The metrics are grouped according to the following categories and are documented here [17]:

- Ease of learning for experts and novices
- Ease of code conversion
  - From CPU code to Accelerator (GPU, etc.) code
  - From low level (CUDA, etc.) to higher level portability code
  - From one portability framework to another
- Impact on other existing code
  - Extent of modifications to existing code: does it take over main(), does it affect the threading or execution model, etc.
  - Extent of modifications to Event Data Model (EDM): data transfer and access across different memory space, etc.
- Impact on existing tool chain and build infrastructure
  - Extent of modifications to build rules / system
  - Do we need to recompile the entire software stack?
  - CMake or make changes/integration
- Hardware mapping
  - Is the technology working on all current hardware architectures?
  - Support for new hardware features and new architectures
- Feature availability
  - Reductions, kernel chaining, callbacks, etc
  - Concurrent kernel execution
  - Support for interfacing to optimized math-heavy libraries (FFTs, etc.)
- Ease of Debugging

  - How easy is it to debug implementations of code in the technologies?
- Address needs of all types of workflows
  - Scaling with # of kernels / application
  - Scaling with # of developers
  - Support for users by portability technology developers
- Long-term sustainability and code stability
  - Support model of technologies, stability of implementation if underlying libraries (CUDA) change
  - CUDA is going to be around for a long time, what about the portability solutions?
  - Long term support for technologies by vendors
- Compilation time
  - Separate builds for different architectures?
  - Compatibility with experiment's software distribution strategies
- Performance: CPU and GPU
  - Does the portable code version (CPU and GPU uses same code) degrade the CPU performance or use more memory?
- Aesthetics
  - compatibility with C++ standards
- Interoperability
  - Can you mix portability technologies in the same application? How are external packages treated if they are imported into experiment software stacks and use different portability technologies? (CMSSW [18, 19, 20, 21, 22] is using Kokkos, but Geant [23, 24, 25] is using Alpaka)
  - Interaction with existing thread pool on CPU/GPU back ends?

3 Use Cases

In the following, we briefly describe the use cases that are being used in this study.

FastCaloSim is a parametrized simulation of the ATLAS Liquid Argon Calorimeter [26]. The codebase was originally written in C++, then ported to CUDA. The CUDA implementation consists of 3 relatively small kernels, which perform a memory re-initialization, the main energy deposition simulation, and finally a stream compaction. It has been ported to Kokkos, SYCL, and std::par, targeting NVIDIA, Intel and AMD hardware.

ACTS is a track reconstruction toolkit for general HEP detectors [16], which is based on C++. The R&D lines for ACTS parallelization on heterogeneous architectures consist of a number of core algorithms for tracking on GPUs (traccc), a geometry offloading package designed explicitly for GPUs (detray), and a memory management layer (vecmem) that is architecture neutral. All tracking algorithms, which include hit clusterization, seeding and Kalman filtering (both simple and combinatorial) will be offloaded onto the GPU to minimize the data transfers between host and device.

Wire Cell The Wire-Cell Toolkit [27, 28] is a C++ software library for the simulation, signal processing, reconstruction and visualization of Liquid Argon Time Projection Chamber (LArTPC) detectors for neutrino experiments, such as the planned Deep Underground Neutrino Experiments (DUNE [14]). The use case we study is the LArTPC signal simulation module in Wire-Cell, which simulates the LArTPC detector response. So far the signal simulation module has been re-implemented in Kokkos [29], and investigation with OpenMP is in progress.

Patatrack The Patatrack use case consists of CMS Heterogeneous Pixel Reconstruction [30, 31] code, that processes the raw pixel detector data up to pixel tracks and vertices, extracted into a standalone program. It includes a multi-threaded mock framework providing similar behavior as CMS software framework CMSSW [18, 19, 20, 21, 22], and input data from CMS Open Data [32]. The original code was developed to run on NVIDIA GPUs with CUDA, accompanied with a simple translation header to allow compilation to CPU. We have ported the code to Kokkos and HIP, targeting NVIDIA and AMD GPUs.

P2R is a light-weight mini-app which performs the track propagation in radial direction and Kalman update kernels in track reconstruction [33]. With a simplified geometry and standalone setup, P2R can be used to test the performance of core tracking computation in various portability technologies in a shorter timescale. The original version is adapted from the mkFit project [34], which implements a parallel Kalman Filter Algorithm [35], and has been re-implemented in CUDA, HIP, Kokkos, Alpaka, OpenACC and std::par.

Random Number Generators We have leveraged the SYCL programming model and its interoperability with third-party libraries to demonstrate cross-platform performance portability across heterogeneous resources. We have implemented NVIDIA and AMD random number generator extensions to the oneMKL open-source interfaces library [36]. The utility of our extensions are exemplified in a real-world setting via a high-energy physics simulation application, showing the performance of implementations that capitalize on SYCL interoperability are at par with native implementations, attesting to the cross-platform performance portability of a SYCL-based approach to scientific codes.

4 Preliminary Results

We gathered some preliminary results from our studies of various portability solutions.

4.1 Kokkos

Kokkos is a programming model and a C++ library for portable performance applications [6]. It provides high-level parallel algorithms, such as for, prefix scan, and reduction, that can be nested with some restrictions, as well as multidimensional array data types. The mapping of work of the algorithms and the default layout of the multidimensional array depend on the chosen back end. Currently (Kokkos 3.5) these back ends include CPU serial, CPU parallel with OpenMP or Posix Threads, and device parallel with CUDA, HIP, HPX [37], or SYCL. The high abstraction level on both algorithms and data is expected to provide reasonably good out-of-the-box performance also on computing architectures beyond CPUs and GPUs.

We were able to express all the custom algorithms in the use cases (FastCaloSim, Wire Cell Toolkit, Patatrack, p2r) in the Kokkos' programming model, but we experienced some challenges. Kokkos requires its run time library to be built for one set of host serial, host parallel, and device parallel back end at a time, and e.g. in case of CUDA the library can support exactly one major GPU architecture version. While this approach works fine for HPC codes that are typically compiled for a specific supercomputer, it poses challenges for HEP experiment frameworks for which a single build is expected to be used in about 200 data centers with different computer hardware. In Kokkos 3.5, the CPU serial back end is thread safe, but it cannot be efficiently used from multiple threads, limiting its current usefulness in multi-threaded applications that process multiple collision events concurrently. Kokkos developers are working to improve the performance for this use case. Much of the HEP data is structured, and multidimensional arrays are useful only in limited use cases, implying a need for a separate solution for data structures. Currently Kokkos does not provide a unified, portable interface to Fast Fourier Transform (FFT) algorithms (e.g. to optimize platform-specific implementations), but such interface is being worked on.

For the WireCell toolkit use case, we have seen moderate performance speedups from multi-core CPUs, AMD GPUs and NVIDIA GPUs, using Kokkos. We have also demonstrated that running multiple concurrent processes to share the GPUs can further improve the performance gains, setting a promising direction for efficient utilization of HPC systems in Wire-Cell. For other use cases we saw performance degradation for specific hardware platforms (AMD GPUs in case of FastCaloSim).

4.2 SYCL

SYCL is an open-standard C++-based programming model that facilitates parallel programming on heterogeneous platforms. It provides a single source programming model, enabling developers to write both host-side and kernel code in the same file. Employing

C++-based template programming, developers can leverage higher-level programming features in writing accelerator-enabled applications with the ability to integrate the native acceleration API, when needed, by using different interoperability interfaces provided by SYCL. The latest specification, SYCL 2020, is based on ISO C++17 standard, and features standard programming with templates and lambda functions to develop optimized code which can be offloaded to special purpose compute accelerators such as GPUs, FPGA, or AI/ML accelerators. The SYCL specification is designed to be a higher level abstraction above low-level native acceleration APIs with interoperability between existing libraries and other parallel programming models and can be built on top of OpenMP, Vulkan [38], OpenCL [39], Kokkos, Raja [40], or some other back end. The SYCL programming model offers performance portability across various vendor hardware and interoperability with both open-source and closed-source (proprietary) software. As SYCL evolves, HPC-critical features will continue to be incorporated into the specification.

The applicability of our SYCL-based Random Number Generators (RNGs) [41, 42] has been evaluated in a GPU port of FastCaloSim [43]. The interfaces we developed enabled the seamless integration of SYCL RNGs into FastCaloSim with no code modification across the evaluated platforms. The SYCL 2020 interoperability functionality enabled custom kernels and vendor-dependent library integration to be abstracted out from the application, leading to improvement of maintainability of the application and reducing the source lines of code. Using our RNG interfaces, we achieve comparable performance with native solutions on different architectures. Whereas the original C++ version of FastCaloSim had two separate code bases, x86 and CUDA, our RNG work has enabled event processing on a variety of major vendor hardware from a single SYCL entry point. Hence, the SYCL RNG based integration facilitates the code maintainability by reducing the FastCaloSim code size without introducing any significant performance overhead.

4.3 OpenMP

OpenMP is a directive-based programming model that has evolved from a shared-memory programming model for multicore CPU architectures to one with rich features to support GPU accelerator offloading. The current version 5.1 of the OpenMP specification includes support for loop-level target offloading, memory management, and asynchronous CPU/GPU execution, all of which may be crucial for experimental HEP workflows. Major HPC hardware vendors such as AMD, HPE, Intel and NVIDIA are all onboard with the OpenMP model, and have been actively developing the compiler infrastructure to support the new and improved OpenMP features. However, as a directive-based programming model, it may not offer the same level of flexibility as language-based programming models such as Kokkos or SYCL. Nevertheless, the premise of only needing to add a few pragmas to the code and letting the compilers handle the low-level optimizations holds great promise for the future, when even more diverse and complex HPC architectures are to be expected. With the anticipated industrial support, OpenMP may become the portable programming model of choice in the future, and R&D into how HEP software can take full advantage of this potential should be supported timely so that we don't fall behind the rest of the scientific computing community.

Our initial investigations into the OpenMP target offloading features have shown that in addition to the simple loop parallelism, many performance-enhancing features are also well supported and relatively easy to use. These examples include asynchronous kernel executions, interoperability with optimized vendor libraries and the ability to specify memory spaces. Compiler support for OpenMP offloading has also been improving rapidly, with several functional compilers available on the market, such as the open-source LLVM Clang compiler, GNU C Compiler, HPE's CCE compiler, along with NVIDIA's, AMD's and Intel's compilers to support their own GPU architectures. While the performances with these compilers can vary quite a bit depending on the specific use case, they have all been steadily improving.

4.4 std::par

std::parallel::execution (std::par) has been part of the C++ standard since C++17, and offers a high level interface to execute the contents of loops concurrently. Until recently, the concurrent back ends have been limited to the host side, using libraries such as TBB [44] to execute on different CPU threads and cores. Recently, both Intel and NVIDIA have released compilers that can target GPU devices (oneapi::dpl and nvc++ respectively). Given their high level nature, most low level functions and optimizations of domain specific languages such as CUDA and SYCL are not available, resulting in a loss of overall performance. However, the entry bar to users is extremely low, and requires little knowledge of GPU programming. Both of these compilers are still rather immature, exhibiting a number of compiler bugs and lack of build system integration, so performance numbers should be taken lightly. The continued development of these compilers is however a good indication that vendors are seeking standards based solutions, with both low and high level APIs.

5 Conclusions

The portable parallelization strategies (PPS) project of the High Energy Physics Center for Computational Excellence (HEP/CCE) is investigating solutions to the changing hardware architecture landscape of today and in the future. With accelerators like GPUs becoming more mainstream, especially in HPC systems, the development of scientific code is at a crossroad leaving the convenient era of x86-only code. To continue writing scientific code efficiently with a large and not always professionally trained user community to run on all hardware architectures, we think we need community solutions for portability techniques that will allow the coding of an algorithm once, and the ability to execute it on a variety of hardware products from many vendors. To that effect, the PPS project is investigating the feasibility of currently available portability solutions and will compare them based on a defined set of metrics. The goal is to develop recommendations for potential users when to deploy the appropriate solution. Preliminary results show deploying portability solutions is currently far from the convenience of having a standard, but there are encouraging successes using these solutions. We think without them, the scientific success of our experiments and endeavors is in danger, as software development could be expert driven and costly to be able

to run on available hardware infrastructure. We think the best solution for the community would be an extension to the C++ standard with a very low entry bar for users, supporting all hardware forms and vendors. We are very far from that ideal though. In the future, as a community, we need to request and work on portability solutions and strive to reach this ideal.

References

[1] "HEP Center for Computational Excellence (HEP-CCE)." https://www.anl.gov/hep-cce.

[2] "NVidia CUDA Toolkit." https://developer.nvidia.com/cuda-toolkit.

[3] "SYCL: C++ Single-source Heterogeneous Programming for OpenCL." https://www.khronos.org/sycl/.

[4] "HIP: C++ Runtime API and Kernel Language to create portable applications." https://github.com/ROCm-Developer-Tools/HIP.

[5] J. Duarte, S. Han, P. Harris, S. Jindariani, E. Kreinar, B. Kreis et al., *Fast inference of deep neural networks in FPGAs for particle physics*, *Journal of Instrumentation* **13** (2018) P07027.

[6] C.R. Trott, D. Lebrun-Grandié, D. Arndt, J. Ciesko, V. Dang, N. Ellingwood et al., *Kokkos 3: Programming model extensions for the exascale era*, *IEEE Transactions on Parallel and Distributed Systems* **33** (2022) 805.

[7] H.C. Edwards, C.R. Trott and D. Sunderland, *Kokkos: Enabling manycore performance portability through polymorphic memory access patterns*, *Journal of Parallel and Distributed Computing* **74** (2014) 3202 .

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

[9] E. Zenker, B. Worpitz, R. Widera, A. Huebl, G. Juckeland, A. Knüpfer et al., *Alpaka - an abstraction library for parallel kernel acceleration*, IEEE Computer Society, May, 2016, http://arxiv.org/abs/1602.08477 [1602.08477].

[10] B. Worpitz, *Investigating performance portability of a highly scalable particle-in-cell simulation code on various multi-core architectures*, Ph.D. thesis, Technische Universität Dresden, Sept., 2015. 10.5281/zenodo.49768.

[11] L. Dagum and R. Menon, *Openmp: an industry standard api for shared-memory programming*, *Computational Science & Engineering, IEEE* **5** (1998) 46.

[12] "ISO/IEC 14882:2020: Programming languages – C++." https://www.iso.org/standard/79358.html, 2020.

[13] CMS collaboration, *The CMS experiment at the CERN LHC*, *JINST* **3** (2008) S08004.

[14] DUNE collaboration, *Long-Baseline Neutrino Facility (LBNF) and Deep Underground Neutrino Experiment (DUNE): Conceptual Design Report, Volume 1: The LBNF and DUNE Projects*, 1601.05471.

[15] ATLAS collaboration, *The ATLAS Experiment at the CERN Large Hadron Collider*, *JINST* **3** (2008) S08003.

[16] A. Salzburger, P. Gessinger, F. Klimpel, M. Kiehn, B. Schlag, H. G. et al., *acts-project/acts: v17.1.0*, .

[17] "HEP-CCE Metrics for portability technologies." https://hep-cce.github.io/Metric.html.

[18] C.D. Jones, M. Paterno, J. Kowalkowski, L. Sexton-Kennedy and W. Tanenbaum, *The new CMS event data model and framework*, in *Proceedings of International Conference on Computing in High Energy and Nuclear Physics (CHEP06)*, 2006.

[19] C.D. Jones and E. Sexton-Kennedy, *Stitched together: Transitioning CMS to a hierarchical threaded framework*, *J. Phys.: Conf. Series* **513** (2014) 022034.

[20] C.D. Jones, L. Contreras, P. Gartung, D. Hufnagel and L. Sexton-Kennedy, *Using the CMS threaded framework in a production environment*, *J. Phys.: Conf. Series* **664** (2015) 072026.

[21] C.D. Jones, *CMS event processing multi-core efficiency status*, *J. Phys.: Conf. Series* **898** (2017) 042008.

[22] A. Bocci, D. Dagenhart, V. Innocente, C. Jones, M. Kortelainen, F. Pantaleo et al., *Bringing heterogeneity to the cms software framework*, *EPJ Web Conf.* **245** (2020) 05009.

[23] J. Allison, K. Amako, J. Apostolakis, H. Araujo, P. Arce Dubois, M. Asai et al., *Geant4 developments and applications*, *IEEE Transactions on Nuclear Science* **53** (2006) 270.

[24] S. Agostinelli, J. Allison, K. Amako, J. Apostolakis, H. Araujo, P. Arce et al., *Geant4—a simulation toolkit*, *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* **506** (2003) 250.

[25] J. Allison, K. Amako, J. Apostolakis, P. Arce, M. Asai, T. Aso et al., *Recent developments in geant4*, *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* **835** (2016) 186.

[26] ATLAS Collaboration collaboration, *The new Fast Calorimeter Simulation in ATLAS*, Tech. Rep. ATL-SOFT-PUB-2018-002, CERN, Geneva (Jul, 2018).

[27] "Wire-cell toolkit." https://github.com/WireCell/wire-cell-toolkit.

[28] X. Qian, C. Zhang, B. Viren and M. Diwan, *Three-dimensional Imaging for Large LArTPCs*, *JINST* **13** (2018) P05032 [1803.04850].

[29] Z. Dong, K. Knoepfel, M. Lin, B. Viren and H. Yu, *Evaluation of Portable Programming Models to Accelerate LArTPC Detector Simulations*, in *20th International Workshop on Advanced Computing and Analysis Techniques in Physics Research: AI Decoded - Towards Sustainable, Diverse, Performant and Effective Scientific Computing*, 3, 2022 [2203.02479].

[30] A. Bocci, V. Innocente, M. Kortelainen, F. Pantaleo and M. Rovere, *Heterogeneous reconstruction of tracks and primary vertices with the cms pixel tracker*, *Front. Big. Data* **3** (2020) 601728 [2008.13461].

[31] "Standalone patatrack pixel tracking." https://github.com/cms-patatrack/pixeltrack-standalone/.

[32] CMS Collaboration, "TTToHadronic_TuneCP5_13TeV-powheg-pythia8 in FEVTDEBUGHLT format for 2018 collision data. CERN Open Data Portal.." doi:10.7483/OPENDATA.CMS.GOB0.0LEW, 2019.

[33] "Light-weight mini-app which performs the track propagation in radial direction and kalman update kernels in track reconstruction." https://github.com/kakwok/p2r-tests.

[34] S. Lantz, K. McDermott, M. Reid, D. Riley, P. Wittich, S. Berkman et al., *Speeding up particle track reconstruction using a parallel kalman filter algorithm*, *Journal of Instrumentation* **15** (2020) P09030–P09030.

[35] R.E. Kalman, *A New Approach to Linear Filtering and Prediction Problems*, *Journal of Basic Engineering* **82** (1960) 35.

[36] "oneAPI Math Kernel Library (oneMKL) Interfaces." https://github.com/oneapi-src/oneMKL.

[37] H. Kaiser, P. Diehl, A.S. Lemoine, B.A. Lelbach, P. Amini, A. Berge et al., *Hpx - the c++ standard library for parallelism and concurrency*, *Journal of Open Source Software* **5** (2020) 2352.

[38] "Vulkan is a cross-platform industry standard enabling developers to target a wide range of devices with the same graphics API." https://www.vulkan.org.

[39] J.E. Stone, D. Gohara and G. Shi, *Opencl: A parallel programming standard for heterogeneous computing systems*, *Computing in Science Engineering* **12** (2010) 66.

[40] "RAJA Performance Portability Layer." https://github.com/LLNL/RAJA.

[41] V.R. Pascuzzi and M. Goli, *Achieving near native runtime performance and cross-platform performance portability for random number generation through SYCL interoperability*, *arXiv e-prints* (2021) arXiv:2109.01329 [2109.01329].

[42] M. Krainiuk, M. Goli and V.R. Pascuzzi, *oneapi open-source math library interface*, in *2021 International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*, pp. 22–32, 2021, DOI.

[43] Z. Dong, H. Gray, C. Leggett, M. Lin, V.R. Pascuzzi and K. Yu, *Porting hep parameterized calorimeter simulation code to gpus*, *Frontiers in Big Data* **4** (2021) .

[44] "oneAPI Threading Building Blocks." https://github.com/oneapi-src/oneTBB.