**Proceedings Paper:**
Smith, Matthew, Wright, Steven A. orcid.org/0000-0001-7133-8533, Lantra, Zaman et al. (1 more author) (2025) The P3 Explorer:An Open Database of Performance, Portability, and Productivity. In: 33rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing. 33rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, 12-14 Mar 2025 , ITA

https://doi.org/10.1109/PDP66500.2025.00079

# The P3 Explorer: An Open Database of Performance, Portability, and Productivity

Matthew A. Smith, Steven A. Wright
*Department of Computer Science*
*University of York*
York, UK
{ms3408,steven.wright}@york.ac.uk

Zaman Lantra, Gihan Mudalige
*Department of Computer Science*
*University of Warwick*
Coventry, UK
{zaman.lantra,g.mudalige}@warwick.ac.uk

***Abstract*—This paper introduces a web-based tool designed to organise and present visual representations of performance, portability, and productivity (P3) data from previously published scientific studies. The P3 Explorer operates as both an open repository of application performance data and a data dashboard, providing visual heuristic analyses of performance portability and developer productivity, created using Intel's P3 Analysis library. The goal of the project is to create a community-led database of P3 studies to better inform application developers of alternative approaches to developing new applications that target high performance on diverse hardware, considering the productivity of the developer. In this paper, we evaluate our tool using a recently published study outlining a performance portable domain-specific language for particle-in-cell applications.**

***Index Terms*—High Performance, Performance Portability, Developer Productivity**

## I. Introduction

High Performance Computing (HPC) plays an important role in the field of computational science, and is used for a wide range of computationally intensive tasks across the engineering and science disciplines. For the past decade, heterogeneity has been one of the driving forces behind the push towards, and beyond, Exascale computing (i.e. $10^{18}$ FLOP/s). The largest HPC systems in the world now derive the vast majority of their computational power from accelerators, such as GPUs.

Extracting high performance from such accelerators typically requires an alternative approach to application development, such as the use of language extensions like CUDA. Porting applications to these architectures is an arduous task, and in some cases an application developed for one accelerator architecture may not run or may not achieve high performance on another accelerator architecture.

The need to maintain application portability, and in particular *performance portability* [1], has led to the development of a number of competing parallel programming paradigms that aim to deliver portability from a single-source code base [2]. Evaluating the performance of these programming approaches on a range of different hardware has been the focus of many previous scientific studies [3]–[13].

While each of these studies provides guidance to application developers, the conclusions are typically specific to an application, a platform, and/or a development approach. This paper presents a tool designed to provide a holistic view of these performance studies; firstly, through an open-source repository of scientific data, and secondly through a visualisation dashboard, designed to showcase performance, portability, and productivity (P3) data with plots and insights automatically generated using Intel's P3 Analysis Library [14]. Specifically, this paper makes the following contributions:

- We introduce an open GitHub repository for archiving performance, portability, and productivity data for HPC application from published scientific studies;
- We present the *P3 Explorer*, a data dashboard for visualising P3 data for published scientific studies [15];
- Finally, we demonstrate our tool with a recent study [16], demonstrating the insights that can be generated by the P3 Analysis Library and our visualisation pipeline.

The P3 Explorer provides an interface for application developers to assess a variety of approaches to developing new parallel software with the goal of maximising performance, portability, and productivity.

The remainder of this paper is structured as follows: Section II provides an overview of related work and an introduction to the concepts explored in this paper; Section III outlines our data repository and the P3 Explorer interface; Section IV presents a case-study of the CabanaPIC application using our exploration tool; and finally, Section V concludes this paper.

## II. Related Work

HPC hardware is diversifying, and there is a proliferation of parallel programming models that target heterogeneous systems, containing GPU accelerators from vendors such as NVIDIA, AMD, and Intel. In recent years, there has been a focus on measuring and evaluating the performance, portability, and productivity of approaches to parallel programming [1], [17], [18]. Developing and subsequently maintaining applications that achieve high performance, and are portable between architectures from different manufacturers is a difficult task, and has been the subject of numerous recent surveys [2], [19]. Moreover, developing maintainable performance portable applications has been a focus of both the Exascale Computing Project [20], [21], and the UK's ExCALIBUR programme [22].

### A. Evaluating Performance, Portability, and Productivity

There are numerous well-tested metrics for assessing the *performance* of a scientific application, including application runtime, FLOP/s, or memory bandwidth. Assessing the *portability* of an application, or developer *productivity* is typically much more difficult.

While portability could be considered a binary metric – an application either runs correctly or it does not – in this work we consider the more nuanced metric outlined by Pennycook et al. [1] that provides a view of *how well* an application runs across multiple platforms. In Equation 1, the *performance portability* $\mathbf{\Phi}$, of an application $a$, solving problem $p$, on a given set of platforms $H$, is calculated by finding the harmonic mean of an application's efficiency ($e_i(a,p)$). Efficiency is a measure of either the achieved performance against the best recorded (possibly non-portable) performance on each individual target platform (i.e. *the application efficiency*), or the achieved performance against the theoretical maximum performance achievable on each individual platform (i.e. *the architectural efficiency*).

$$\mathbf{\Phi}(a,p,H) = \begin{cases} \dfrac{|H|}{\displaystyle\sum_{i \in H} \dfrac{1}{e_i(a,p)}} & \text{if } i \text{ supported } \forall i \in H \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

This metric provides us with an objective view of portability, where an application that runs but performs badly will have a lower performance portability score than a more performant alternative, and should an application fail to run on a particular platform, it will achieve a score of zero.

Measuring developer productivity is more difficult still, but in our tool we use the code divergence metric proposed by Harrell et al. [18], and adopted by Intel's P3 Analysis Library [14].

$$\text{CD}(a,p,H) = \begin{pmatrix} |H| \\ 2 \end{pmatrix}^{-1} \sum_{\{i,j\} \in H \times H} d_{i,j}(a,p) \tag{2}$$

In Equation (2), code divergence is a measure of the average "distance" between the source code required to compile an application $a$, and execute problem $p$ for each pair of platforms in $H$, where $d_{i,j}(a,p)$ is any distance metric between two source codes. Pennycook et al. [17] suggest using the Jaccard distance (shown in Equation (3)), where $c_i(a,p)$ represents the set of source lines required to compile application $a$ and execute problem $p$ on a given platform $i$.

$$d_{i,j}(a,p) = 1 - \frac{|c_i(a,p) \cap c_j(a,p)|}{|c_i(a,p) \cup c_j(a,p)|} \tag{3}$$

These metrics provide a single data point for performance portability and developer productivity that may be useful in assessing a particular approach to developing performance portable applications. However, to provide a more holistic view of performance portability and developer productivity, our dashboard visualises these data using *cascade plots* and *performance-portability code-convergence* ($\mathbf{\Phi}$-*CC*) *charts* [17]. These allow users to better evaluate the P3 properties of an application visually.

### B. Previous P3 Studies

There are a number of studies focused on "the three Ps", with many of these focused on the evaluation of a particular programming model, language, or compiler. One of the earliest approaches to portable heterogeneous application development is OpenCL; Pennycook et al. evaluate the performance of the LU mini-application from the NAS Parallel Benchmarks, demonstrating the importance of selecting the right data layout for each target platform [9].

Applications from the UK Mini-App Consortium (UKMAC) have been the target of numerous studies, demonstrating the performance and portability of OpenMP, MPI, CUDA, OpenACC, Kokkos, RAJA, and the OPS and OP2 domain-specific languages [4], [8], [10], [12].

Asahi et al. evaluate the performance portability of a Vlasov code using OpenACC, OpenMP, and Kokkos across a range of CPU and GPU platforms [13]. Deakin et al. track the efficiency of various approaches to developing performance portable software over a number of years, showing the growing maturity of various software stacks [3].

The SYCL programming model has been the focus of a number of recent works, in particular evaluating the maturity of SYCL compiler toolchains [6], [7] and the performance of the programming model at Exascale on the Aurora system [5]. Of particular note, the work by Rangel et al. includes an evaluation of productivity, in addition to performance portability, showing that SYCL can deliver portability with little to no code divergence between platforms [5].

## III. The P3 Explorer

The P3 Explorer consists of two components: (i) a P3 data repository for storing study data alongside detailed artifact descriptions; and (ii) a data dashboard, displaying visual representations of P3 data for comparison and analysis [15]. Application developers can explore the dashboard to gain an insight into historic data to better inform their own approach to application development – primarily focused on which programming approaches to apply to maximise efficiency over a broad range of hardware, while considering the impact on developer productivity. The dashboard is available at https://p3-explorer.github.io.

### A. The Data Repository

The data repository stores user-submitted data, consisting of performance and productivity results, and associated metadata. It is stored on the `main` branch of a GitHub repository, with all submitted data located within a "submissions" directory. This directory is structured as shown in Figure 1, with a directory for each scientific application.

Each unique application has its own directory, containing an associated TOML file of metadata, such as an application description, scientific domain classifications, and references to application sources. Within each application directory there is a data directory containing study data. Each unique study is stored based on the year, month, and first author surname, and data should contain performance data in CSV, and an artifact

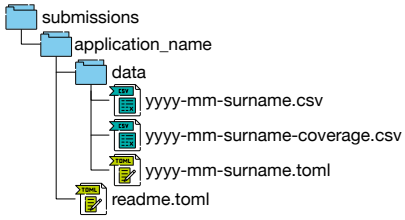


Fig. 1: An illustration of the submissions directory structure

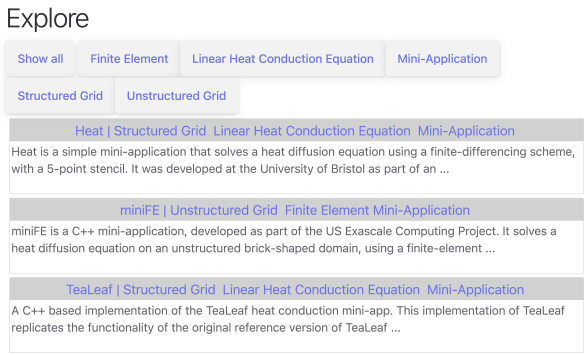


Fig. 2: An example of the database exploration page

description in an associated TOML file. Code coverage data can optionally be included in CSV format (see the P3 Analysis Library for further information [14]).

### *B. The Explorer Interface*

The P3 Explorer dashboard is hosted through GitHub Pages, and renders visualisations of the performance studies stored within the data repository. The website is updated via GitHub Actions on merged pull requests to the data repository.

On the dashboard, studies are organised by application, and categorised by scientific domain(s). Figure 2 shows an example of the exploration interface. Each application has its own subpage, which lists all associated performance studies by published article title.

Each study subpage contains plots generated from performance and productivity data, and metadata such as an artifact description, links to sources, and a DOI reference. Plots generated from performance data can be found at the top of each study page, as shown in Figure 3. These plots include a bar chart of the raw performance data and P3 visualisations, generated with the P3 Analysis Library [14]. In all cases a *cascade plot* is generated, and where code coverage data is present, a $\Phi$-*CC chart* is generated [17].

### *C. Contributing*

The intent for the P3 Explorer is to be a community-led effort to collate data from scientific studies in a single GitHub repository, to enable easier data analysis and study replication.

Contributors should first fork the `main` branch of the p3-explorer from GitHub. Next, a contributor should find their application, or create a new directory for their application, within the `submissions` directory. They should then create or update the required CSV and TOML files (as illustrated in Figure 1). The final step for contributing data to the repository

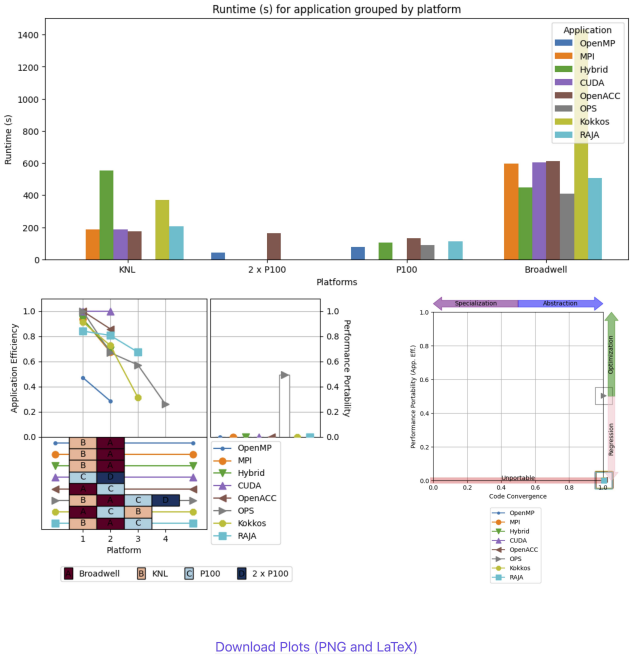


Fig. 3: An example experimental data page for Kirk et al. [4]

```
title       = "TeaLeaf"
appDomain   = ["Structured Grid", ...]
sources     = [["GitHub", "https://github.com/..."]]
description = """A C++ based implementation of the ..."""
```

Fig. 4: An example TOML file for a new application

```
title       = "Achieving Performance Portability for ..."
authors     = ["R. O. Kirk" , "G. R. Mudalige", ...]
sources     = [
                ["TeaLeaf Repository", ...],
                ["OPS APPS Repository", ...],
                ...
              ]
doi         = "10.1109/CLUSTER.2017.122"
fom         = "Runtime (s)"
tags        = ["Structured Grid", ...]
description = """In this paper, we investigate ..."""
```

Fig. 5: An example TOML file for a new data submission

is to commit these changes to the forked repository, and then create a *Pull Request* through GitHub to the p3-explorer repository.

For a new application, a TOML readme file is required in the application's root directory containing relevant application information. Figure 4 shows an example for the TeaLeaf application [4].

As a minimum, scientific study data submissions should include performance data in CSV format (with column labels matching the P3 Analysis Library [14]), and a TOML file with information about the submission. Figure 5 shows an example study submission TOML file, where the description explains how the performance data was collected (e.g. computational environment, compiler versions, compiler flags).

TABLE I: Runtime (s) comparison of CabanaPIC using Kokkos and OP-PIC across different devices and architectures

| Platform | Architecture | Kokkos | OP-PIC (Backend) |
|---|---|---|---|
| Intel Xeon 6252 24-Core | Cascade Lake | 66.039 | 48.197 (OpenMP) |
| | | | 41.811 (MPI) |
| Intel Xeon 5418Y 24-Core | Sapphire Rapids | 35.813 | 33.213 (OpenMP) |
| | | | 29.258 (MPI) |
| AMD EPYC 7543 32-Core | Zen3 (Milan) | 29.786 | 27.986 (OpenMP) |
| | | | 28.199 (MPI) |
| AMD EPYC 9334 32-Core | Zen4 (Genoa) | 30.080 | 31.816 (OpenMP) |
| | | | 40.074 (MPI) |
| NVIDIA P100 | Pascal | 22.406 | 22.749 (CUDA) |
| NVIDIA V100 | Volta | 15.128 | 15.631 (CUDA) |
| NVIDIA H100 | Hopper | 9.949 | 10.097 (CUDA) |
| AMD MI100 | GFX908 | 562.239 | 32.983 (HIP, seg-reductions) |
| | | | 587.505 (HIP, atomics) |
| AMD MI210 | GFX90A | 8.492 | 8.113 (HIP, unsafe atomics) |

All submissions must adhere to the submission guidelines and format, and each pull request is reviewed prior to acceptance into the repository. Datasets can be updated at any time, overwriting old or incorrect data if necessary. The P3 Explorer dashboard automatically updates all data pages and plots on a commit or merge.

## IV. Case Study

To demonstrate the application of the P3 Explorer, we present a case study using the CabanaPIC mini-application, which is a 3D electromagnetic Particle-in-Cell (PIC) code. The original application is developed as a part of the Exascale Computing Project's Co-design center for Particle Applications (CoPA) using the performance portable Cabana library, based on Kokkos [23].

Our case study is based on a recent publication by Lantra et al. assessing the performance of OP-PIC, a new domain-specific language (DSL) for unstructured-mesh PIC simulations [16]. The OP-PIC DSL is aimed at gaining performance portability on current and emerging, massively parallel architectures using source-to-source translation to generate platform-specific optimisations.

The original CabanaPIC implementation consists of shared memory parallelisations using the Kokkos C++ template library to target CPUs (with OpenMP) and GPUs (with CUDA, HIP, or SYCL). Conversely, OP-PIC uses code generation to target CPUs and GPUs, using combinations of OpenMP, MPI, CUDA, and HIP, while also having the ability to generate code optimised to particular generations of hardware.

Although CabanaPIC is a structured-mesh PIC application, OP-PIC is a DSL for unstructured-mesh PIC, and so the OP-PIC implementation uses unstructured-mesh mappings, solving the same physics as the original.

For the comparisons presented here, we investigate the performance on a single CPU socket or a single GPU. Table I lists the achieved runtimes of the Kokkos and OP-PIC implementations of CabanaPIC across a range of CPU and GPU architectures, from Intel, NVIDIA, and AMD. For the simulation, the configuration is a mesh of 96,000 cells and 72 million particles for 200 iterations in double-precision (FP64).

One known bottleneck of the PIC algorithm is when particles try to accumulate current/charge to cells. To handle the data races that occur when multiple particles deposit to the same cell, the original CabanaPIC uses scatter-views (which may use atomics underneath for GPUs); conversely, the OP-PIC implementation can use either atomics or *segmented reductions* depending on the platform.

The study shows that atomics on NVIDIA GPU hardware provides good performance and causes little serialisation; however, to gain performance on AMD GPUs, an alternative approach is required. Using the `-munsafe-fp-atomics` compiler flag means that comparable performance can be seen on the AMD MI210 GPU with both Kokkos and OP-PIC. However, unsafe-atomics is not supported on the AMD MI100 GPU, and using atomics leads to an uncompetitive runtime. OP-PIC provides an alternative optimisation, known as segmented reductions, that brings down the runtime significantly, but at the expense of higher memory consumption; this is an implementation not available through the Kokkos-backed original code.

### A. Exploring the Performance, Portability, and Productivity

To analyse this data with respect to the performance portability, we prepare the data in Table I as a CSV file using the appropriate column headings for the P3 Analysis Library [14]. For the Kokkos programming model, there is a single runtime for each platform; for OP-PIC, there may be numerous, based on the backend used. In our CSV dataset we include the OP-PIC results along with the backend used, and additionally we duplicate the best OP-PIC result for each platform to an additional "OP-PIC (Best)" data set. To complete our data submission, we prepare appropriate TOML files for the CabanaPIC application and to document the source of our study data [16]. Figures 6, 7, and 8 show the plots generated by the P3 Explorer for our dataset.

Figure 6 visualises the results in Table I, highlighting that each platform has a comparable runtime for both Kokkos and OP-PIC, with the exception of the AMD MI100. In this case, as previously mentioned, the Kokkos implementation relies on double-precision hardware atomic operations that are poorly supported on this platform. A similar result is achieved

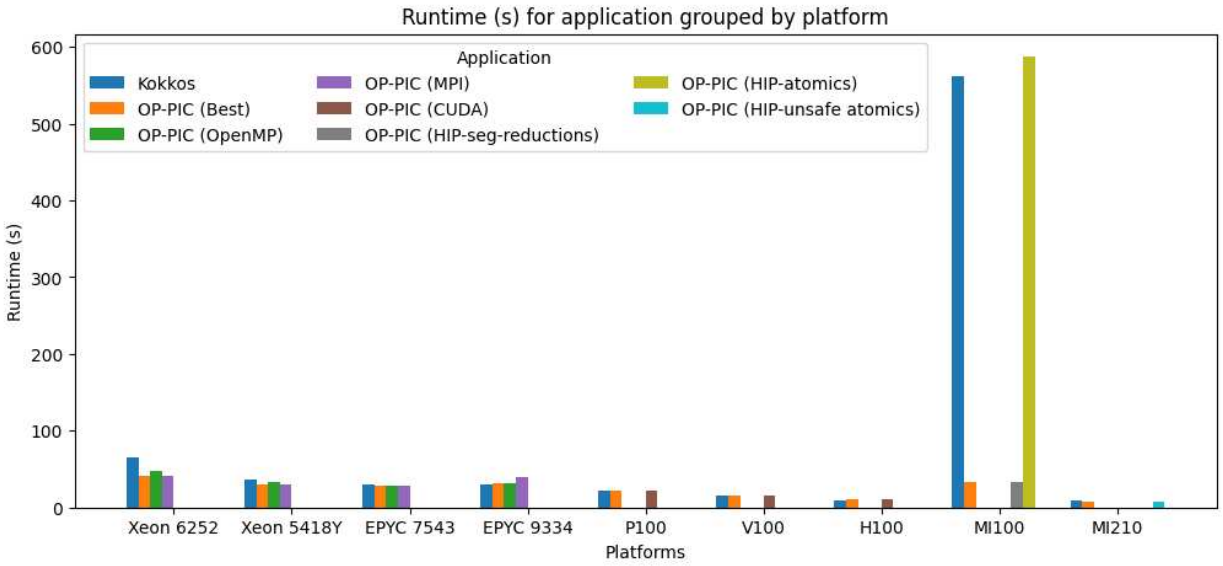


Fig. 6: A bar chart showing the runtime performance of CabanaPIC across 9 platforms

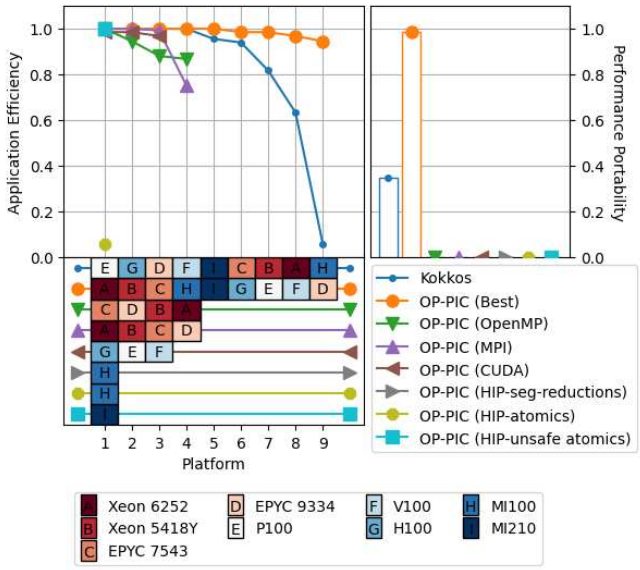


Fig. 7: A P3 cascade plot generated for CabanaPIC

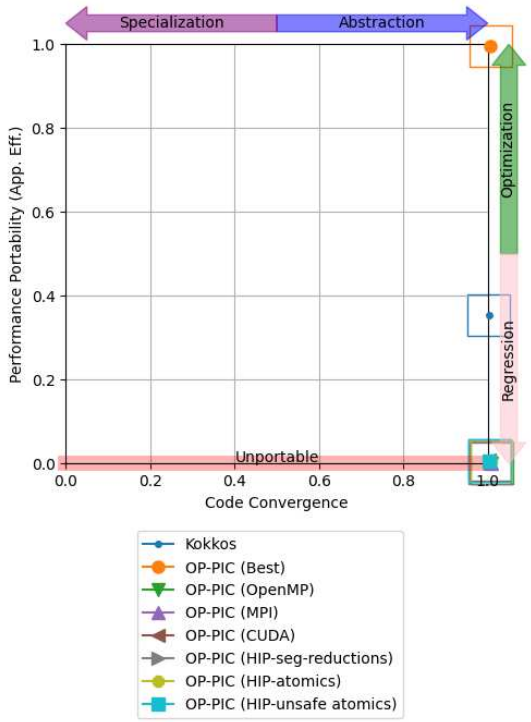


Fig. 8: A P3 $\Phi$-*CC* chart generated for CabanaPIC

with OP-PIC using hardware atomics, but significantly better performance is possible with an alternative backend.

Figure 7 shows a P3 cascade plot for this data, showing how the portability of OP-PIC compares to Kokkos as we add more platforms to our evaluation set. Overall, we can see that OP-PIC is able to achieve better portability across our 9 platforms based on its code generation technology – achieving a $\Phi \approx 0.987$, compared to $\Phi \approx 0.347$ for Kokkos. However, we can see from Figure 7 that platform H (MI100) is the source of the largest degradation in portability for Kokkos, where hardware atomics are not well supported. We can additionally observe that platforms A and B (both Intel Xeon CPUs) are another source of portability degradation for the Kokkos implementation, where both the OpenMP and MPI implementations from OP-PIC perform better. Besides these platforms, the performance of Kokkos and OP-PIC are largely comparable.

Figure 8 shows a $\Phi$-*CC* chart for our dataset. Since both Kokkos and OP-PIC are single-source approaches to software development, no code divergence is seen, hence the plot repeats the performance portability data from Figure 7.

The P3 Explorer allows us to conduct such portability feasibility studies, and then observe trends in the data visually, to better inform development efforts.

## V. Conclusion

Performance, portability, and productivity are vital when designing and developing new HPC applications for heterogeneous systems [2]. In this paper we have outlined an open-source database of P3 study results, alongside a visualisation framework for analysing P3 data. We have applied this framework to a recent study of the CabanaPIC application by Lantra et al. [16], demonstrating the analysis that can be performed using the P3 Analysis Library and our visualisation pipeline.

Our repository provides an open archive of scientific data from past studies focused on "the three Ps", and we aim for this database to become a community effort to collect data and artifact descriptions to better inform future development efforts across the various HPC application domains.

The success of this project depends on community participation in building an effective database, with appropriate metadata to enable reproducibility. The primary ongoing work will therefore be to collect, collate, and validate study data and metadata, and to encourage wide participation in the project.

## Acknowledgement

Matthew Smith was funded under EPSRC's Vacation Internships scheme.

This work used the ARCHER2 UK National Supercomputing Service (https://www.archer2.ac.uk).

## References

[1] S. J. Pennycook, J. D. Sewall, and V. W. Lee, "Implications of a metric for performance portability," *Future Generation Computer Systems*, vol. 92, 2019.

[2] S. A. Wright, C. Ridgers *et al.*, "Developing performance portable plasma edge simulations: A survey," *Computer Physics Communications*, vol. 298, 2024.

[3] T. Deakin, A. Poenaru, T. Lin, and S. McIntosh-Smith, "Tracking Performance Portability on the Yellow Brick Road to Exascale," in *The IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*, 2020, pp. 1–13.

[4] R. O. Kirk, G. R. Mudalige *et al.*, "Achieving Performance Portability for a Heat Conduction Solver Mini-Application on Modern Multi-core Systems," in *The IEEE International Conference on Cluster Computing*, Sep. 2017.

[5] E. M. Rangel *et al.*, "A Performance-Portable SYCL Implementation of CRK-HACC for Exascale," in *IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*, 2023.

[6] W. Shilpage and S. A. Wright, "An Investigation into the Performance and Portability of SYCL Compiler Implementations," *Lecture Notes in Computer Science (LNCS)*, vol. 13999, Aug. 2023.

[7] W.-C. Lin, T. Deakin, and S. McIntosh-Smith, "On Measuring the Maturity of SYCL Implementations by Tracking Historical Performance Improvements," in *International Workshop on OpenCL (IWOCL'21)*, 2021, pp. 1–13.

[8] D. Truby, S. A. Wright, R. Kevis, S. Maheswaran, J. A. Herdman, and S. A. Jarvis, "BookLeaf: An Unstructured Hydrodynamics Mini-Application," in *2018 IEEE International Conference on Cluster Computing (CLUSTER)*, Sep. 2018, pp. 615–622.

[9] S. J. Pennycook, S. D. Hammond, S. A. Wright, J. A. Herdman, I. Miller, and S. A. Jarvis, "An Investigation of the Performance Portability of OpenCL," *Journal of Parallel and Distributed Computing (JPDC)*, vol. 73, no. 11, pp. 1439–1450, November 2013.

[10] J. A. Herdman, W. P. Gaudin, S. McIntosh-Smith, M. Boulton, D. A. Beckingsale, A. C. Mallinson, and S. A. Jarvis, "Accelerating Hydrocodes with OpenACC, OpenCL and CUDA," in *Proceedings of the 2012 SC Companion: High Performance Computing, Networking Storage and Analysis*, ser. SCC '12. USA: IEEE Computer Society, 2012, p. 465–471.

[11] M. T. Bettencourt, D. A. S. Brown *et al.*, "EMPIRE-PIC: A Performance Portable Unstructured Particle-in-Cell Code," *Communications in Computational Physics*, vol. 30, no. 4, pp. 1–37, Mar. 2021.

[12] T. R. Law, R. Kevis, S. Powell, J. Dickson, S. Maheswaran, J. A. Herdman, and S. A. Jarvis, "Performance Portability of an Unstructured Hydrodynamics Mini-application," in *2018 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*, Nov 2018, pp. 0–12.

[13] Y. Asahi, G. Latu, J. Bigot, and V. Grandgirard, "Optimization strategy for a performance portable Vlasov code," in *The International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*, 2021, pp. 79–91.

[14] S. J. Pennycook *et al.*, "Performance, Portability and Productivity Analysis Library," 10.5281/zenodo.7733678, Mar. 2023.

[15] M. A. Smith, S. A. Wright, Z. Lantra, and G. R. Mudalige, "The P3 Explorer: Exploring the Performance, Portability, and Productivity Wilderness," in *The IEEE/ACM International Conference for High Performance Computing, Networking, Storage, and Analysis (SC24)*, 2024.

[16] Z. Lantra, S. A. Wright, and G. R. Mudalige, "OP-PIC – an Unstructured-Mesh Particle-in-Cell DSL for Developing Nuclear Fusion Simulations," ser. ICPP '24, 2024, pp. 294—304.

[17] S. J. Pennycook *et al.*, "Navigating Performance, Portability, and Productivity," *Computing in Science & Engineering*, vol. 23, no. 5, 2021.

[18] S. L. Harrell, J. Kitson *et al.*, "Effective performance portability," in *IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*, 2018, pp. 24–36.

[19] I. Z. Reguly and G. R. Mudalige, "Productivity, performance, and portability for computational fluid dynamics applications," *Computers & Fluids*, vol. 199, pp. 1–10, 2020.

[20] Exascale Computing Project, "ECP Proxy Applications," https://proxyapps.exascaleproject.org/ (accessed April 20, 2021), 2021.

[21] T. M. Evans, A. Siegel, E. W. Draeger, J. Deslippe, M. M. Francois, T. C. Germann, W. E. Hart, and D. F. Martin, "A survey of software implementations used by application codes in the Exascale Computing Project," *The International Journal of High Performance Computing Applications*, vol. 36, no. 1, pp. 5–12, 2022.

[22] E. J. Threlfall, R. J. Akers *et al.*, "Software for Fusion Reactor Design: ExCALIBUR Project NEPTUNE:Towards Exascale Plasma Edge Simulations," in *29th IAEA Fusion Energy Conference*, October 2023.

[23] S. M. Mniszewski, J. Belak *et al.*, "Enabling particle applications for exascale computing platforms," *The International Journal of High Performance Computing Applications*, vol. 35, no. 6, pp. 572–597, 2021.