UK Atomic Energy Authority

ExCALIBUR

Domain-Specific Language (DSL) and Performance Portability Assessment

D3.2

The report describes work for ExCALIBUR project NEPTUNE at the point of Deliverable 3.2. It binds the technical reports “Identification of Testbed Platforms and Applications” (2047358-TN-02 [1]) and “Evaluation of Approaches to Performance Portability” (2047358-TN-03 [2]), the second and third reports produced by the consortium from the Universities of York and Warwick.

This group’s first report “Approaches to Performance Portable Applications for Fusion” (2047358-TN-01 [3]) provides a survey of the present state of exascale hardware and technologies. The key point is that exascale hardware is expected to be diverse: HPC systems will be heterogeneous or hierarchical, with GPUs accelerating host CPUs. Codes will need to be *performance portable* – able to run efficiently on a range of hardware from multiple vendors, without requiring significant source code modifications for each system. Many technologies are being developed to enable performance portability – most notably SYCL, Kokkos and RAJA – which aim for a single source code to target CPUs and GPUs (and FPGAs), and achieve comparable performance to native implementation that use CUDA, OpenACC or OpenMP for their respective targets. The reports [1, 2] assess the current state of these technologies as applied to fusion plasma software, with the overarching aim of enabling best practice for achieving performance portability in NEPTUNE software development.

TN-02: Identification of Testbed Platforms and Applications

The first of the reports included here describes the performance portability assessment methodology. It performs two main tasks. The first is to outline metrics to describe performance and portability across a range of platforms. This is a new topic that is under development. The difficulty lies in reducing very rich, multi-variate performance data into a concise but meaningful description. After discussing the single-number performance portability metric of Pennycook *et al.* [4], the authors adopt two-dimensional metrics, namely box plots and the “cascade plot” approach from Sewall *et al.* [5]. Cascade plots show a performance efficiency for each machine on the vertical axis against a range of machines on the horizontal axis. This gives a clearer sense of portability than a single score metric, but still necessarily suppresses a huge amount of data by giving a single score for each machine. For example, from such plots one cannot assess how performance changes with the number of compute nodes or with problem size. One must also bear in mind the potential sensitivity of the metric to the choice of machines used in the assessment.

The second task the report performs is to determine the sets of hardware and software to be used in the assessment. Both homogeneous and heterogeneous hardware platforms are used, and most are UK Tier 1 (Archer2) or Tier 2 machines. The software is selected from existing proxyapps, chosen to be broadly representative of NEPTUNE software, implementing algorithms that have similar computation and communication patterns to those that will be used in NEPTUNE code. Seven proxyapps are chosen: four fluid codes and three particle-in-cell (PIC) codes. The software uses a variety of approaches, with the fluid codes covering finite difference, finite element and spectral element approaches, and the PIC codes covering different algorithmic choices and structure/unstructured meshes. The fluid proxyapps cover a range of programming models – SYCL, Kokkos and RAJA, the DSL/code generation library OPS, and native implementations in MPI, OpenMP, OpenACC and CUDA. The limited availability of PIC codes however means that these are implemented only in Kokkos.

TN-03: Evaluation of Approaches to Performance Portability

The second report is an initial assessment of the platforms and applications selected in the previous report. This is intended as a living document, to be updated as more performance data becomes available. Moreover, many of the proxyapps, libraries and programming models are under active development, so conclusions are subject to change. Where possible, the report uses performance data taken from publications, but where necessary the authors have supplemented this with their own simulations.

The main conclusion is that non-portable implementations – MPI and/or OpenMP for CPUs and CUDA for NVIDIA GPUs – always outperform the portable implementations provided by SYCL, Kokkos and RAJA. The latter are at best 20% slower than native implementations on CPUs and 50% slower on GPUs. The DSL/code generation approach provided by OPS offers a compromise option between the portable and non-portable paradigms. For example, OPS is used in the fluid proxyapp TeaLeaf to generate code to target MPI, OpenMP, OpenACC, CUDA and HIP, achieving performance comparable with native CPU implementations and within a factor of two of native GPU implementations. However DSLs constrain source code to use their specific APIs, reducing the flexibility allowed to developers (in this case, OPS only targets structured grids). Moreover, the DSL inserts an additional abstraction layer to be maintained, likely by NEPTUNE developers, rather than external developers as would be the case with SYCL, Kokkos and RAJA. In any case, the conclusion at present is that ensuring best performance across a range of hardware requires the maintenance of multiple implementations of the code, one for each hardware target.

Future directions

Choice of programming model In terms of assessing the relative merits of SYCL, Kokkos, RAJA and OPS, the results presented here are preliminary and limited by available data. The conclusions only apply to the fluids proxyapps, as all the PIC proxyapps are parallelized using Kokkos. Moreover, even among the fluid proxyapps, there is no direct like-for-like comparison between programming models.

From these results, there is no clear choice of best model. While Kokkos and RAJA appear superior to SYCL, at least some of this discrepancy is due to compiler and data-collection issues. Elsewhere, Wright expresses a preference for using SYCL, though with the caveat that the choice of model may not be critical [6]. Project NEPTUNE currently has links with Codeplay, developers of one of the SYCL implementations (who are also undertaking work on SYCL backend for Kokkos), and with Lawrence Livermore National Laboratory, developers of RAJA. These links should meet the concern that SYCL (and hence NEPTUNE) may fail consequent on for example, withdrawal of support for DPC++ by Intel.

Given the performance advantages of native implementations over portable programming models, it is natural to question the value of such models. For example, if it is necessary to write and maintain bespoke backends for our SYCL or Kokkos implementation to obtain high performance on different architectures, in what sense is our code actually portable? Does this tie NEPTUNE unnecessarily into a dependence on an external library and constrain developers to using a programming model, while still essentially requiring the maintenance of multiple native implementations? Possibly! Indeed, native implementations may make more sense when code execution is limited to a small set of HPC systems, allowing intensive bespoke optimize for each. However, the aspiration of the NEPTUNE project is to be widely-used and also long-lasting. This requires the flexibility to run on as yet unseen hardware. Using programming models allows us to have a lot of portability - here no more than 20% slower on CPUs and 50% slower on GPUs without tuning - while maintaining a single source. While in absolute terms these are large degrees of performance to lose in grand challenge problems, these are acceptable figures for untuned initial simulations. They are a strong baseline to work from, allowing NEPTUNE developers to focus performance optimization for specific machines as and when it becomes appropriate through modifying specific backends. Moreover, choosing a programming model gives NEPTUNE the same advantages as choosing library implementations: it becomes able to leverage the optimization work of others, freeing NEPTUNE developers to work on NEPTUNE-specific problems.

Assessment of algorithms There is an important direction that this assessment does not treat and which need to be addressed in future work. It assesses the performance of different implementations of the same algorithm, but it does not assess the relative performance of algorithms that are meant to produce the same physics results by different numerical means. These results show us that discrepancies exist, but do not explain why that is so.

On a related note, these reports deal with assessing on-node software performance. This is a valuable exercise, as it is highly likely that the programming model at exascale will be MPI+X – MPI between nodes, and X some unknown programming model within a node. Therefore much can be learnt about exascale performance from single-node experiments on pre-exascale machines. But nonetheless this analysis does not assess the suitability of algorithms for massively multi-node simulations. It is conceivable that at scale large performance discrepancies arise between different algorithms that were not observed simply from their small-scale implementations.

As the software development plan becomes more refined, the issue of comparing algorithm performance – both on-node and off – will be a key area of focus.

Acknowledgement

The support of the UK Meteorological Office and Strategic Priorities Fund is acknowledged.

References

[1] S. Wright, B. Dudson, P. Hill, D. Dickinson, and G. Mudalige. Identification of Testbed Platforms and Applications. Technical Report 2047358-TN-02, UKAEA Project Neptune, 2021.

[2] S. Wright, B. Dudson, P. Hill, D. Dickinson, and G. Mudalige. Evaluation of Approaches to Performance Portability. Technical Report 2047358-TN-03, UKAEA Project Neptune, 2021.

[3] S. Wright, B. Dudson, P. Hill, D. Dickinson, and G. Mudalige. Approaches to Performance Portable Applications for Fusion. Technical Report 2047358-TN-01-02, UKAEA Project Neptune, 2021.

[4] S. J. Pennycook, J. D. Sewall, and V. W. Lee. Implications of a metric for performance portability. *Future Generation Computer Systems*, 92:947–958, 2019.

[5] J. Sewall, S. J. Pennycook, D. Jacobsen, T. Deakin, and S. McIntosh-Smith. Interpreting and visualizing performance portability metrics. In *2020 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*, pages 14–24. IEEE, 2020.

[6] E. Threlfall and W. Arter. Survey of code generators and their suitability for NEPTUNE. Technical Report CD/EXCALIBUR-FMS/0039-M3.2.1, UKAEA, 2021.

UKAEA REFERENCE AND APPROVAL SHEET

	Client Reference:	
	UKAEA Reference:	CD/EXCALIBUR-FMS/0049
	Issue:	1.0
	Date:	23 September 2021
Project Name: ExCALIBUR Fusion Modelling System		

	Name and Department	Signature	Date
Prepared By:	Joseph Parker Wayne Arter BD	N/A N/A	23 September 2021 23 September 2021
Reviewed By:	Rob Akers Advanced Computing Dept. Manager		23 September 2021
Approved By:	Rob Akers Advanced Computing Dept. Manager		23 September 2021

T/NA086/20
Code structure and coordination

2047358-TN-02

Identification of Testbed Platforms and Applications

Steven Wright, Ben Dudson, Peter Hill, and David Dickinson

University of York

Gihan Mudalige

University of Warwick

July 9, 2021

1 Introduction

In our previous report we provided a state-of-the-art review of available hardware and software for the development of post-Exascale applications. The report highlights two key observations:

- The hardware landscape is diversifying. At Exascale there will likely be a diverse range of hardware available, and many of the largest systems will employ heterogeneous/hierarchical parallelism, characterised predominantly by the use of GPU accelerators.
- There is a wide range of approaches to software development which allow for portability between architectures. However, whether these approaches enable us to obtain the best performance on each architecture (performance portability) without significant manual modifications (productivity) is a key question.

The focus of this project is to establish best practice for developing a new plasma-fusion application that might achieve the trinity of *Performance*, *Portability* and *Productivity*.

In this project we seek to evaluate a number of hardware platforms and associated software development methodologies. This report outlines how we will assess these factors for the remainder of the project. We will also identify a number of representative applications, in the form of mini-applications that implement algorithms of interest with similar computational/communication patterns to those likely to be present in the NEPTUNE codebase.

1.1 Evaluating Performance Portability

The basis for this investigation will be to analyse the *performance portability* of a number of software development methodologies using the established metric introduced by Pennycook et al. [1] and restated in Equation 1.

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

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 applications performance efficiency ($e_i(a,p)$). The performance efficiency for each platform can be calculated by comparing achieved performance against the best recorded (possibly non-portable) performance on each individual target platform (i.e. *the application efficiency*, or by comparing the achieved performance against the theoretical maximum performance achievable on each individual platform (i.e. *the architectural efficiency*). Should the application fail to run on one of the target platforms, a performance portability score of 0 is awarded.

The work by Pennycook et al. highlights a number of alternative measures of performance portability, highlighting their shortcomings at providing actionable insights [1]. They outline five criteria a useful metric should aspire to, and then demonstrate how their metric meets each of these criteria. Specifically, a useful metric should: (i) be measured specific to a set of platforms of interest H; (ii) be independent of the absolute performance across H; (iii) be zero if a platform in H is unsupported, and approach zero as the performance of platforms in H approach zero; (iv) increase if performance increases on any platform in H; and (v) be directly proportional to the sum of scores across H.

Since publication of this metric, it has been used extensively to assess the performance portability of a number of applications and programming models [2, 3, 4, 5, 6, 7]. This project aims to replicate this effort with a focus on applications and algorithms of interest to the plasma fusion community.

While a single, numeric metric has a number of advantages, there are also some shortcomings. For example, if a particular application fails to run on one platform then it will score 0, even if the application is performant on all other platforms. To overcome issues such as this, Sewall et al. have proposed a number of methods for visualising performance portability metrics [8].

The first of these is box plots, as demonstrated on synthetic data in Figure 1.

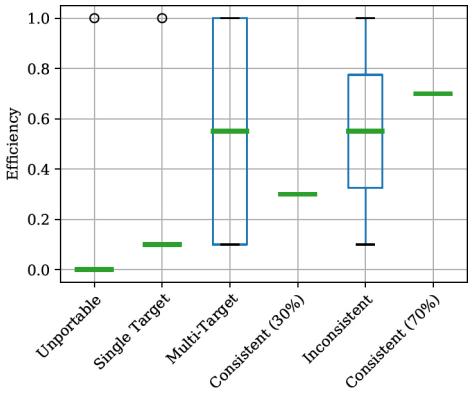


Figure 1: Visualising performance portability with box plots [8]

In both Figure 1 and Figure 2: an *unportable* application does not run on one or more of the available platforms; a *single target* solution runs well on a single platform, and achieves 10% performance on all other platforms; a *multi-target* application runs at 100% on half of the platforms, and 10% on all other platforms; an *inconsistent* solution performs increasingly better on each platform (from 10% to up 100%); and the *consistent* applications perform at 30% and 70% on all platforms, respectively.

In Figure 1, we can see that although the unportable application scores 0, there is an outlier showing that the application is performant on some of the platforms – information that is lost when evaluating based on a single numeric metric. For the other synthetic datasets, we see the performance portability ($\mathbf{\Phi}$), along with information about the range of efficiencies across all of the platforms. For some applications or kernels, it may be the case that a performance portability profile like that of the multi-target solution is acceptable, whereas for others, a consistent 70% might be more appropriate. Figures such as this can help us make these assessments, without relying on a single piece of information.

The second visualisation technique proposed by Sewall et al. is cascade plots, as shown in Figure 2. In these plots, the target platforms are labelled A-J, and plotted beneath the graph. Each application is profiled based on an increasing set of platforms (ordered from most efficient to least for each application), where the filled lines plot the platform efficiencies, and the dotted lines show

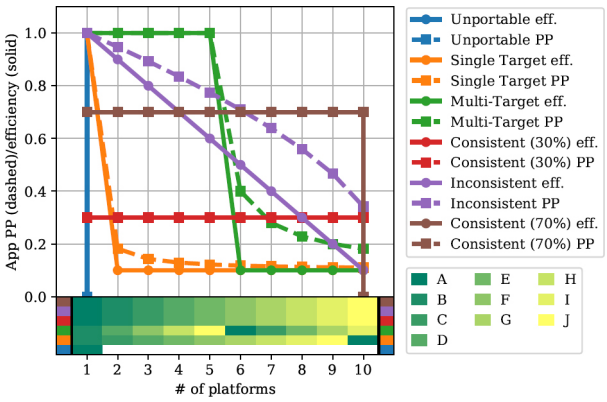


Figure 2: Visualising performance portability with cascade plots [8]

the corresponding performance portability as the platform set grows. Again, visualising performance portability data using these visual heuristics allows a developer to make a reasoned assessment about what might be acceptable for a particular application or kernel.

2 Proxy Applications

The exploratory stage of NEPTUNE includes a number of projects that are investigating the behaviour of plasmas through proxy applications. The applications currently being used broadly fall in to two categories, *fluid models* and *particle models*. In particular, T/NA078/20 is using Nektar++ to explore the performance of spectral elements, T/NA083/20 is focused on building a fluid referent model in both Bout++ and Nektar++, and T/NA079/20 is exploring the use of particle methods with the EPOCH particle-in-cell (PIC) code. It is therefore likely that the resultant NEPTUNE software stack will be a fluid model, based on the output of T/NA078/20 and T/NA083/20, coupled with a particle model, based on the output of T/NA079/20.

The three aforementioned applications are the result of many years of development and typically consist of many thousands of lines of C/C++ or Fortran. They are already widely used by the UK's scientific computing community on a diverse range of problems.

Prior to the development of the NEPTUNE software stack, it is prudent to assess the wide range of available technologies, without the associated burden of redeveloping these mature simulation applications into new programming frameworks. In this project, we have therefore decided to identify a series mini-applications that implement key computational algorithms that are similar to those used by the NEPTUNE proxy applications. These mini-applications are typically limited to a few thousand lines of code and are often available implemented in a wide range of programming frameworks already.

Notable collections of such mini-applications includes Rodinia [9], UK-MAC [10], the NAS Parallel Benchmarks [11], the ECP Proxy Apps [12] and the SPEC benchmarks [13]. In this section, we will discuss the applications we have identified from these benchmark suites, that may be relevant to our performance investigations.

2.1 Fluid Models

As previously noted, the fluid modelling aspects of the NEPTUNE project are largely focused on the use of **Bout++** [14, 15] and **Nektar++** [16]. Bout++ is a framework for writing fluid and plasma simulations in curvilinear geometry, implemented using a finite-difference method, while Nektar++ is a framework for solving computational fluid dynamics problems using the spectral element method.

Both applications are large C++ applications designed primarily for execution across homogeneous clusters. Parallelisation across a cluster in both applications is achieved using MPI, with Bout++ additionally capable of on-node parallelism with OpenMP. GPU acceleration is under development in both applications, through RAJA and HYPRE in Bout++, and through OpenACC in Nektar++ [17].

Rather than redevelop these applications, this project has instead identified a series of mini-applications that implement similar computational schemes. Specifically, we have identified a finite difference mini-app, two finite element mini-apps and one spectral element mini-app, each of which are implemented in a range of programming models for rapid evaluation of approaches to performance portability.

TeaLeaf

TeaLeaf is a finite difference mini-app that solves the linear heat conduction equation on a regular grid using a 5-point stencil. It has been used extensively in studying performance portability already [7, 2, 18, 19], and is available implemented using CUDA, HYPRE, OpenCL, PETSc and Trilinos[1].

miniFE

miniFE is a finite element mini-app, and part of the Mantevo benchmark suite [20, 21, 22, 23]. It implements an unstructured implicit finite element method and is available implemented using CUDA, Kokkos, OpenMP and OpenMP with offload[2].

Laghos

Laghos is a mini-app that is part of the ECP Proxy Applications suite [24, 25, 23]. It implements a high-order curvilinear finite element scheme on an unstructured mesh. It uses HYPRE for parallel linear algebra, and is additionally available in CUDA, RAJA and OpenMP implementations[3].

Nekbone

Nekbone is a mini-app that is representative of one of the core kernels of the incompressible Navier-Stokes solver Nek5000, from Argonne National Laboratory [26, 27, 28, 23]. Like Nek5000, it uses a high-order spectral element discretisation. The mini-app is available implemented using OpenMP, and with accelerator support via CUDA and OpenACC[4].

2.2 Particle Methods

The optimal use of particles in NEPTUNE is currently being explored using the EPOCH particle-in-cell code [29], and its associated mini-app, minEPOCH [30][5]. EPOCH is a PIC code that runs on a structured grid, using a finite differencing scheme and an implementation of the Boris push. Like Bout++ and Nektar++, EPOCH is a mature software package that is used widely by the UK science community, and thus is difficult to evaluate in alternative programming

[1] `http://uk-mac.github.io/TeaLeaf/`
[2] `https://github.com/Mantevo/miniFE`
[3] `https://github.com/CEED/Laghos`
[4] `https://github.com/Nek5000/Nekbone`
[5] `https://github.com/ExCALIBUR-NEPTUNE/minepoch`

models without a significant redevelopment effort. Furthermore, EPOCH (and minEPOCH) is developed in Fortran, making it increasingly difficult to adapt to many new programming models that are heavily based on C++.

The mini-app variant of EPOCH, minEPOCH, is likewise developed in Fortran and thus not appropriate for this study. However, there are a number of particle-based mini-apps that may be of interest to this project, that implement similar particle schemes, backed by a variety of electric/magnetic field solvers.

CabanaPIC

CabanaPIC is a structured PIC code built using the CoPA Cabana library for particle-based simulations [23]. Through the CoPA Cabana library, the application can be parallelised using Kokkos for on-node parallelism and GPU use, and with MPI for off-node parallelism[6].

VPIC/VPIC 2.0

Vector Particle-in-Cell (VPIC) is a general purpose PIC code for modelling kinetic plasmas in one, two or three dimensions, developed at Los Alamos National Laboratory [31]. VPIC is parallelised on-core using vector intrinsics, on-node through pthreads or OpenMP and off-node using MPI. VPIC 2.0 [32] adds support for heterogeneity by using Kokkos to optimise the data layout and allow execution on accelerator devices[7].

EMPIRE-PIC

EMPIRE-PIC is the particle-in-cell solver central the the ElectroMagnetic Plasma In Realistic Environments (EMPIRE) project [33]. It solves Maxwell's equations on an unstructured grid using a finite-element method, and implements the Boris push for particle movement. EMPIRE-PIC makes extensive use of the Trilinos library, and uses Kokkos as its parallel programming model [34, 35].

Each of the three particle-based mini-apps identified implement a PIC algorithm that is similar to that found in EPOCH. However, one weakness of this evaluation set is that all three applications are parallelised on-node through the Kokkos performance portability layer. Currently, we are unaware of any SYCL/DPC++-based PIC codes, however Kokkos has a range of backends

[6] `https://github.com/ECP-copa/CabanaPIC`
[7] `https://github.com/lanl/vpic`

including OpenMP target, and preliminary support for SYCL/DPC++ code generation.

Alongside the minEPOCH mini-app, there was a C++ mini-app port called miniEPOCH that is now orphaned [36], but may prove a useful evaluation vehicle should time allow a porting exercise.

Beyond the PIC method, there are other particle-based applications that we may consider as part of our evaluation, such the molecular dynamic mini-application, miniMD [37]. The evaluation set will be re-evaluated as the project progresses.

3 Evaluation Platforms

The primary focus of this project is to provide an assessment of the options available when developing Exascale-capable software. In the previous section we identified a series of mini-applications that have been implemented using a range of techniques that we believe will be important to developing future-proofed fusion simulations.

Many of these applications have already been evaluated on various platforms by others, and this project does not seek to re-run these experiments. Instead, wherever possible we will seek to collect performance data from existing studies and apply the metrics and visualisation techniques described in Section 1. Where evaluations do not already exist, we will look to evaluate performance on UK-based platforms.

Broadly speaking, we can divide UK-based HPC platforms into two categories, *homogeneous systems* and *heterogeneous systems*. The UK's only Tier-1 system, ARCHER2, is an homogeneous system with an estimated peak performance of 28 PFLOP/s. Across the UK's Tier-2 systems, there is a significant degree of diversity, offering a wide range of homogeneous and heterogeneous platforms/-partitions.

Where we require additional evaluation of these applications, we will be able to leverage access to a number of systems in the UK, such as those listed below. Evaluation on other UK or US based systems may also be possible, through existing collaborations.

3.1 Homogeneous Systems

ARHCER2

The national supercomputer, ARCHER2, is installed at the Edinburgh Parallel Computing Centre (EPCC). ARCHER2 is a Cray Shasta system interconnected with Cray Slingshot fabric. It consists of 5,848 nodes, each with two AMD EPYC Rome CPUs.

Avon

Avon will be a homogeneous cluster of 180 nodes, containing dual Intel Xeon Cascade Lake CPUs installed at the University of Warwick (expected mid-2021). It will be interconnected with Infiniband.

Isambard

The Isambard Tier-2 service is predominantly composed of Marvell ThunderX2 ARM cores, connected by a Cray Aries interconnect. Beside the ThunderX2 cabinet, Isambard also contains a cabinet of Fujistu A64FX processors.

Viking

Viking is a large Linux compute cluster supporting research needs at the University of York. It consists of approximately 170 compute nodes, each with Intel Xeon Skylake CPUs, connected via Infiniband.

Cirrus

The Cirrus cluster, installed at EPCC, consists of 280 compute nodes, each with dual Intel Xeon Broadwell processors. The cluster is connected via Infiniband fabric.

3.2 Heterogeneous Systems

Viking

The Viking cluster, at the University of York, is further bolstered by two GPU nodes, providing a small heterogeneous compute capability. The two GPU nodes each contain four NVIDIA V100 GPUs.

Bede

The Bede system, installed at the University of Durham, has an architecture similar to that found on Summit and Sierra. Bede is a single cabinet of IBM POWER9 CPUs each supporting four NVIDIA V100 GPUs.

Isambard

Alongside the two ARM-based partitions on the Isambard system is the Multi-Architecture Comparison System (MACS). MACS contains four nodes each with two NVIDIA P100 GPUs, and four nodes each with an NVIDIA V100 GPU. It also contains four nodes with AMD EPYC Rome CPUs, and four nodes with Intel Xeon Cascade Lake CPUs. Finally, there are also eight Intel Xeon Phi nodes, and two IBM Power9 nodes with NVIDIA V100 GPUs.

CSD3

The Cambridge Service for Data Driven Discovery (CSD3) provide two supercomputers under EPSRC Tier-2. Peta4 is a system comprising predominantly of Intel Xeon Skylake CPUs, with a small number of Intel Xeon Phil nodes. Wilkes2 provides the largest GPU enabled system in the UK, comprising of 90 nodes each with four NVIDIA P100 GPUs.

Baskerville

The Baskerville system will be the University of Birmingham's Tier-2 cluster. There are 46 compute nodes, each with four NVIDIA A100 GPUs alongside Intel Xeon Ice Lake CPUs.

4 Conclusions

This report has identified a number of mini-applications that implement similar numerical methods to those of interest in the NEPTUNE project. These applications will be our focus for the remainder of this project, using these applications to evaluate approaches to developing performance portable fusion applications.

Our next report will gather performance data from available sources, and will begin the process of evaluating performance across a range of architectures using performance portability metrics and visualisation techniques. Where data is not available it will be gathered from the systems we have at our disposal.

This analysis will allow us to make comparisons between differing programming models and in turn make well reasoned recommendations for the NEPTUNE programme.

References

[1] S.J. Pennycook, J.D. Sewall, and V.W. Lee. Implications of a metric for performance portability. *Future Generation Computer Systems*, 92:947 – 958, 2019.

[2] R. O. Kirk, G. R. Mudalige, I. Z. Reguly, S. A. Wright, M. J. Martineau, and S. A. Jarvis. Achieving Performance Portability for a Heat Conduction Solver Mini-Application on Modern Multi-core Systems. In *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 834–841, Sep. 2017.

[3] Daniela F. Daniel and Jairo Panetta. On applying performance portability metrics. In *2019 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*, pages 50–59, 2019.

[4] S. L. Harrell, J. Kitson, R. Bird, S. J. Pennycook, J. Sewall, D. Jacobsen, D. N. Asanza, A. Hsu, H. C. Carrillo, H. Kim, and R. Robey. Effective performance portability. In *2018 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*, pages 24–36, Nov 2018.

[5] 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)*, pages 0–12, Nov 2018.

[6] Simon McIntosh-Smith. Performance Portability Across Diverse Computer Architectures. In *P3MA: 4th International Workshop on Performance Portable Programming models for Manycore or Accelerators*, 2019.

[7] Tom Deakin, Simon McIntosh-Smith, James Price, Andrei Poenaru, Patrick Atkinson, Codrin Popa, and Justin Salmon. Performance portability across diverse computer architectures. In *2019 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*, pages 1–13, 2019.

[8] Jason Sewall, S. John Pennycook, Douglas Jacobsen, Tom Deakin, and Simon McIntosh-Smith. Interpreting and visualizing performance portabil-

ity metrics. In *2020 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*, pages 14–24, 2020.

[9] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W. Sheaffer, Sang-Ha Lee, and Kevin Skadron. Rodinia: A benchmark suite for heterogeneous computing. In *2009 IEEE International Symposium on Workload Characterization (IISWC)*, pages 44–54, 2009.

[10] UK Mini-App Consortium. Uk-mac. `http://uk-mac.github.io` (accessed April 20, 2021), 2021.

[11] David H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, Horst D. Simon, V. Venkatakrishnan, and S. K. Weeratunga. The NAS Parallel Benchmarks. *International Journal of High Performance Computing Applications*, 5(3):63–73, 1991.

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

[13] Guido Juckeland, William Brantley, Sunita Chandrasekaran, Barbara Chapman, Shuai Che, Mathew Colgrove, Huiyu Feng, Alexander Grund, Robert Henschel, Wen-Mei W. Hwu, Huian Li, Matthias S. Müller, Wolfgang E. Nagel, Maxim Perminov, Pavel Shelepugin, Kevin Skadron, John Stratton, Alexey Titov, Ke Wang, Matthijs van Waveren, Brian Whitney, Sandra Wienke, Rengan Xu, and Kalyan Kumaran. SPEC ACCEL: A Standard Application Suite for Measuring Hardware Accelerator Performance. In Stephen A. Jarvis, Steven A. Wright, and Simon D. Hammond, editors, *High Performance Computing Systems. Performance Modeling, Benchmarking, and Simulation*, pages 46–67. Springer International Publishing, 2015.

[14] Benjamin Daniel Dudson, Peter Alec Hill, David Dickinson, Joseph Parker, Adam Dempsey, Andrew Allen, Arka Bokshi, Brendan Shanahan, Brett Friedman, Chenhao Ma, David Schwörer, Dmitry Meyerson, Eric Grinaker, George Breyiannia, Hasan Muhammed, Haruki Seto, Hong Zhang, Ilon Joseph, Jarrod Leddy, Jed Brown, Jens Madsen, John Omotani, Joshua Sauppe, Kevin Savage, Licheng Wang, Luke Easy, Marta Estarellas, Matt Thomas, Maxim Umansky, Michael Løiten, Minwoo Kim, M Leconte, Nicholas Walkden, Olivier Izacard, Pengwei Xi, Peter Naylor, Fabio Riva,

Sanat Tiwari, Sean Farley, Simon Myers, Tianyang Xia, Tongnyeol Rhee, Xiang Liu, Xueqiao Xu, and Zhanhui Wang. BOUT++, 10 2020.

[15] B D Dudson, M V Umansky, X Q Xu, P B Snyder, and H R Wilson. BOUT++: A framework for parallel plasma fluid simulations. *Computer Physics Communications*, 180:1467–1480, 2009.

[16] C.D. Cantwell, D. Moxey, A. Comerford, A. Bolis, G. Rocco, G. Mengaldo, D. De Grazia, S. Yakovlev, J.-E. Lombard, D. Ekelschot, B. Jordi, H. Xu, Y. Mohamied, C. Eskilsson, B. Nelson, P. Vos, C. Biotto, R.M. Kirby, and S.J. Sherwin. Nektar++: An open-source spectral/hp element framework. *Computer Physics Communications*, 192:205–219, 2015.

[17] Jan Eichstädt. Implementation of High-performance GPU Kernels in Nektar++, 2020.

[18] Matthew Martineau, Simon McIntosh-Smith, and Wayne Gaudin. Assessing the performance portability of modern parallel programming models using tealeaf. *Concurrency and Computation: Practice and Experience*, 29(15):e4117, 2017.

[19] Simon McIntosh-Smith, Matthew Martineau, Tom Deakin, Grzegorz Pawelczak, Wayne Gaudin, Paul Garrett, Wei Liu, Richard Smedley-Stevenson, and David Beckingsale. TeaLeaf: A Mini-Application to Enable Design-Space Explorations for Iterative Sparse Linear Solvers. In *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 842–849, 2017.

[20] Richard Frederick Barrett, Li Tang, and Sharon X. Hu. Performance and Energy Implications for Heterogeneous Computing Systems: A MiniFE Case Study. 12 2014.

[21] Alan B. Williams. Cuda/GPU version of miniFE mini-application. 2 2012.

[22] Meng Wu, Can Yang, Taoran Xiang, and Daning Cheng. The research and optimization of parallel finite element algorithm based on minife. *CoRR*, abs/1505.08023, 2015.

[23] David F. Richards, Yuri Alexeev, Xavier Andrade, Ramesh Balakrishnan, Hal Finkel, Graham Fletcher, Cameron Ibrahim, Wei Jiang, Christoph Junghans, Jeremy Logan, Amanda Lund, Danylo Lykov, Robert Pavel,

Vinay Ramakrishnaiah, et al. FY20 Proxy App Suite Release. Technical Report LLNL-TR-815174, Exascale Computing Project, September 2020.

[24] J. C. Camier. Laghos summary for CTS2 benchmark. Technical Report LLNL-TR-770220, Lawrence Livermore National Laboratory, March 2019.

[25] Robert Anderson, Julian Andrej, Andrew Barker, Jamie Bramwell, Jean-Sylvain Camier, Jakub Cerveny, Veselin Dobrev, Yohann Dudouit, Aaron Fisher, Tzanio Kolev, Will Pazner, Mark Stowell, Vladimir Tomov, Ido Akkerman, Johann Dahm, David Medina, and Stefano Zampini. Mfem: A modular finite element methods library. *Computers & Mathematics with Applications*, 81:42–74, 2021. Development and Application of Open-source Software for Problems with Numerical PDEs.

[26] Ilya Ivanov, Jing Gong, Dana Akhmetova, Ivy Bo Peng, Stefano Markidis, Erwin Laure, Rui Machado, Mirko Rahn, Valeria Bartsch, Alistair Hart, and Paul Fischer. Evaluation of parallel communication models in nekbone, a nek5000 mini-application. In *2015 IEEE International Conference on Cluster Computing*, pages 760–767, 2015.

[27] Stefano Markidis, Jing Gong, Michael Schliephake, Erwin Laure, Alistair Hart, David Henty, Katherine Heisey, and Paul Fischer. Openacc acceleration of the nek5000 spectral element code. *The International Journal of High Performance Computing Applications*, 29(3):311–319, 2015.

[28] Jing Gong, Stefano Markidis, Erwin Laure, Matthew Otten, Paul Fischer, and Misun Min. Nekbone performance on gpus with openacc and cuda fortran implementations. *The Journal of Supercomputing*, 72(11):4160–4180, 2016.

[29] T D Arber, K Bennett, C S Brady, A Lawrence-Douglas, M G Ramsay, N J Sircombe, P Gillies, R G Evans, H Schmitz, A R Bell, and C P Ridgers. Contemporary particle-in-cell approach to laser-plasma modelling. *Plasma Physics and Controlled Fusion*, 57(11):113001, sep 2015.

[30] Michael Bareford. minEPOCH3D Performance and Load Balancing on Cray XC30. Technical Report eCSE03-1, Edinburgh Parallel Computer Centre, 2016.

[31] K. J. Bowers, B. J. Albright, B. Bergen, L. Yin, K. J. Barker, and D. J. Kerbyson. 0.374 Pflop/s Trillion-Particle Kinetic Modeling of Laser Plasma

Interaction on Roadrunner. In *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, SC '08. IEEE Press, 2008.

[32] Robert Bird, Nigel Tan, Scott V Luedtke, Stephen Harrell, Michela Taufer, and Brian Albright. VPIC 2.0: Next Generation Particle-in-Cell Simulations. *IEEE Transactions on Parallel and Distributed Systems*, pages 1–1, 2021.

[33] Matthew T. Bettencourt and Sidney Shields. EMPIRE Sandia's Next Generation Plasma Tool. Technical Report SAND2019-3233PE, Sandia National Laboratories, March 2019.

[34] Matthew T. Bettencourt, Dominic A. S. Brown, Keith L. Cartwright, Eric C. Cyr, Christian A. Glusa, Paul T. Lin, Stan G. Moore, Duncan A. O. McGregor, Roger P. Pawlowski, Edward G. Phillips, Nathan V. Roberts, Steven A. Wright, Satheesh Maheswaran, John P. Jones, and Stephen A. Jarvis. EMPIRE-PIC: A Performance Portable Unstructured Particle-in-Cell Code. *Communications in Computational Physics*, x(x):1–37, March 2021.

[35] Dominic A.S. Brown, Matthew T. Bettencourt, Steven A. Wright, Satheesh Maheswaran, John P. Jones, and Stephen A. Jarvis. Higher-order particle representation for particle-in-cell simulations. *Journal of Computational Physics*, 435:110255, 2021.

[36] Robert F Bird, Patrick Gillies, Michael R Bareford, Andy Herdman, and Stephen Jarvis. Performance Optimisation of Inertial Confinement Fusion Codes using Mini-applications. *The International Journal of High Performance Computing Applications*, 32(4):570–581, 2018.

[37] S. J. Pennycook and S. A. Jarvis. Developing performance-portable molecular dynamics kernels in opencl. In *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*, pages 386–395, 2012.

T/NA086/20
Code structure and coordination

Report 2047358-TN-03

Evaluation of Approaches to Performance Portability

Steven Wright, Ben Dudson, Peter Hill, and David Dickinson

University of York

Gihan Mudalige

University of Warwick

July 28, 2021

Contents

1 Introduction

The focus of the *code structure and coordination* work package is to establish a series of "best practices" on how to develop simulation applications for Exascale systems that are able to obtain high performance on each architecture (i.e. are performance portable) without significant manual porting efforts.

In the past decade, a large number of approaches to developing performance portable code have been developed. In this report we will begin to report on our evaluation of some of these approaches through the execution of a small number of mini-applications that implement methods similar to those likely to be required in NEPTUNE.

These applications are detailed in report 2047358-TN-02, but are summarised below for convenience:

TeaLeaf
: A finite difference mini-app that solves the linear heat conduction equation on a regular grid using a 5-point stencil[1].

miniFE
: A finite element mini-app, and part of the Mantevo benchmark suite[2].

Laghos
: A high-order curvilinear finite element scheme on an unstructured mesh[3].

Nekbone
: A high-order spectral element application for solving the incompressible Navier-Stokes equations.[4].

CabanaPIC
: A structured PIC code built using the CoPA Cabana library for particle-based simulations[5].

VPIC/VPIC 2.0
: A general purpose PIC code for modelling kinetic plasmas in one, two or

[1] `http://uk-mac.github.io/TeaLeaf/`
[2] `https://github.com/Mantevo/miniFE`
[3] `https://github.com/CEED/Laghos`
[4] `https://github.com/Nek5000/Nekbone`
[5] `https://github.com/ECP-copa/CabanaPIC`

three dimensions, developed at Los Alamos National Laboratory[6].

EMPIRE-PIC

An unstructured PIC code that uses the finite-element method.

The selected applications broadly represent the algorithms of interest for the NEPTUNE project and fall in to two categories – fluid-methods and particle-methods. Within the fluid-method tranche, the applications are available implemented in a wide range of programming models, allowing us a good opportunity to evaluate the effect of programming model on the performance, and importantly the *performance portability* of that particular approach to application development. There are a relatively small number of particle-in-cell mini-applications available, and thus the selected particle-methods applications are only available implemented using Kokkos. However, this still allows us an opportunity to evaluate the appropriateness of Kokkos as a programming model for performance portable application development.

As stated previously, we will evaluate the performance portability of these applications using the metric introduced by Pennycook et al. [1], and use the visualisation techniques outlined by Sewall et al. [2].

Where possible, performance data has been taken from previously published works. Where no data exists, the data has been collected from the UK's Tier-2 platforms, in particular Isambard's Multi-Architecture Comparison System (MACS), ThunderX2 system and A64FX system.

As many of the applications, libraries and programming models used in this report are under active development, the data presented here is subject to change. New data is being collected all the time and analysed, and will be updated in the future where necessary. This document should therefore be considered a living document, reflecting the current state of performance portable application development focused on applications of interest for the simulation of plasma physics.

[6] `https://github.com/lanl/vpic`

2 Application Evaluations

In this section we present performance data for a number of mini-applications, across a range of architectural platforms, using a range of different approaches to performance portability.

The applications chosen in each case are broadly representative of some of the algorithms of interest to NEPTUNE. In particular, the fluid-method based mini-apps implement algorithms that range from finite-difference (like Bout++ [3]) to high-order finite element or spectral element (like Nektar++ [4]). Similarly, the particle-methods mini-apps all implement the particle-in-cell method (like EPOCH [5]).

2.1 TeaLeaf

TeaLeaf is a finite difference mini-app that solves the linear heat conduction equation on a regular grid using a 5-point stencil, developed as part of the UK-MAC (UK Mini-App Consortium) project.

It has been used extensively in studying performance portability already [6, 7, 8, 9], and is available implemented using CUDA, OpenACC, OPS, RAJA, and Kokkos, among others[7]. The results in this section are extracted from two of these studies, namely one by Kirk et al. [7] and one by Deakin et al. [6].

In both studies, the largest test problem size (`tea_bm_5.in`) is used, a 4000×4000 grid.

2.1.1 Performance

The study by Kirk et al. executes 8 different implementation/configurations of TeaLeaf across 3 platforms, a dual Intel Broadwell system, an Intel KNL system and an NVIDIA P100 system. The raw runtime figures are presented in Figure 1. Note that in the study, some results are missing due to incompatibility (e.g. CUDA on Broadwell/KNL).

[7] `http://uk-mac.github.io/TeaLeaf/`

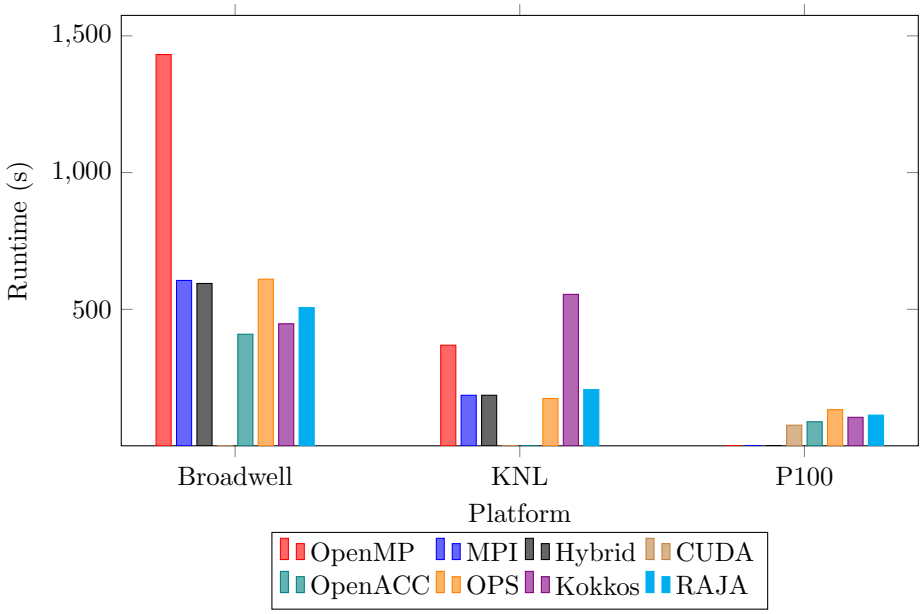


Figure 1: TeaLeaf runtime data from Kirk et al. [7]

The study by Deakin et al. is more recent, using a C-based implementation of TeaLeaf as its base. It consequently evaluates less programming models, but over a wider range of hardware, including a dual Intel Skylake system, both NVIDIA P100 and V100 systems, AMDs Naples CPU, and the Arm-based ThunderX2 platform. Runtime results are provided in Figure 2.

2.1.2 Performance Portability

Both studies evaluate some portable and non-portable implementations. In most cases, there is a non-portable implementation that achieves the lowest runtime, however this places a restriction on the hardware that it can target.

For study by Kirk et al. [7], Figures 3 and 4 allow us to visualise the performance portability of each approach to application development. The figures show a clear divide between portable approaches (Kokkos, OPS and RAJA), and the non-portable approaches (CUDA, OpenMP and MPI). While typically higher performance can be achieved non-portably, each of these programming models tightly binds developers to particular architectures.

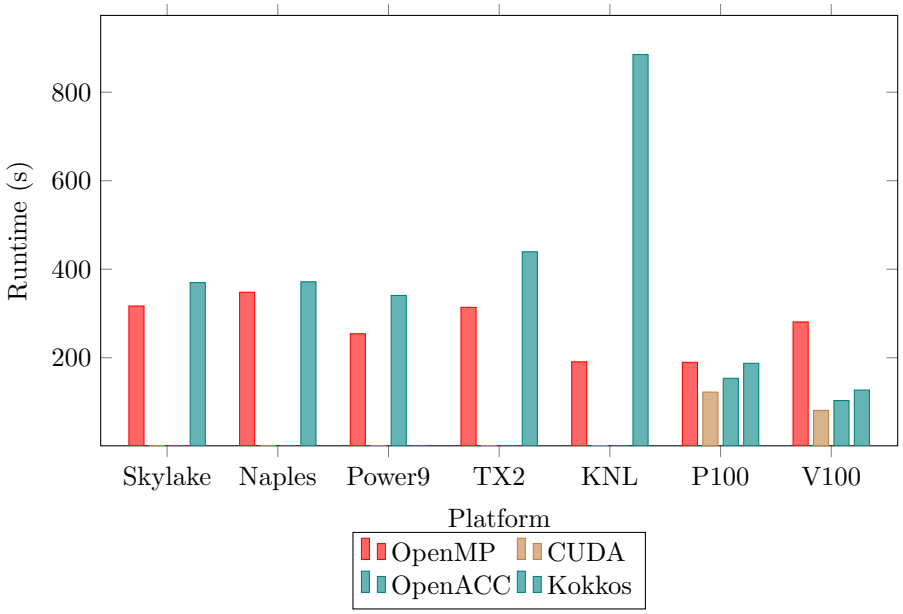


Figure 2: TeaLeaf runtime data from Deakin et al. [6]

Of the portable approaches, the best portability is achieved by RAJA, followed by Kokkos, and then the domain specific language, OPS. Referring back to Figure 1, we can see that on the Intel KNL system, the Kokkos performance is double that of other performance portable approaches, and thus skews the portability calculation. It is likely that this is the result an unidentified issue in TeaLeaf or Kokkos at the time of evaluation. Otherwise, these three programming models each achieve similar levels of performance and, importantly, portability across different architectures.

Figures 5 and 6 show the same visualisations for the data from Deakin et al. [6]. Again, the non-portable programming model (CUDA) achieves the highest performance on its target architecture. For CPU architectures OpenMP produces the highest result, and using offload directives, portability is available to GPU devices. It should be noted that to support the use of GPU devices, there are two OpenMP implementations that must be maintained (with and without offload directives), though these results are presented together here. Much like in the previous study, the performance portability of Kokkos is affected by an anomalous result on the Intel KNL platform.

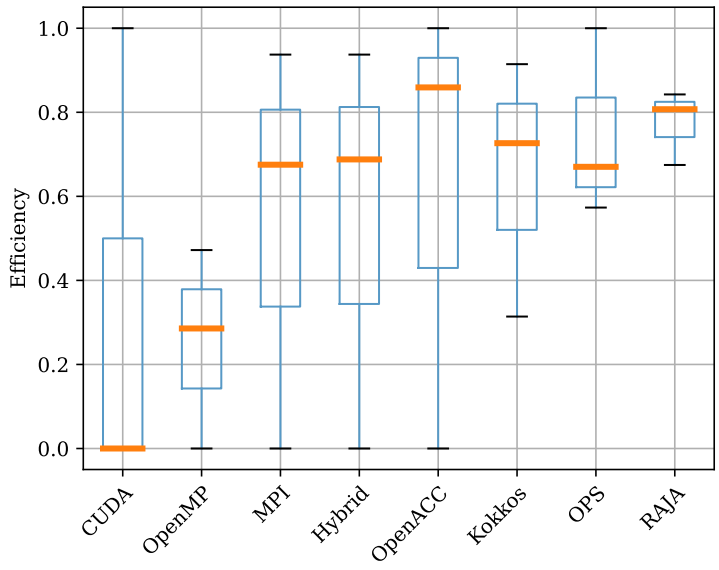


Figure 3: Box plot visualisation of performance portability from Kirk et al. [7]

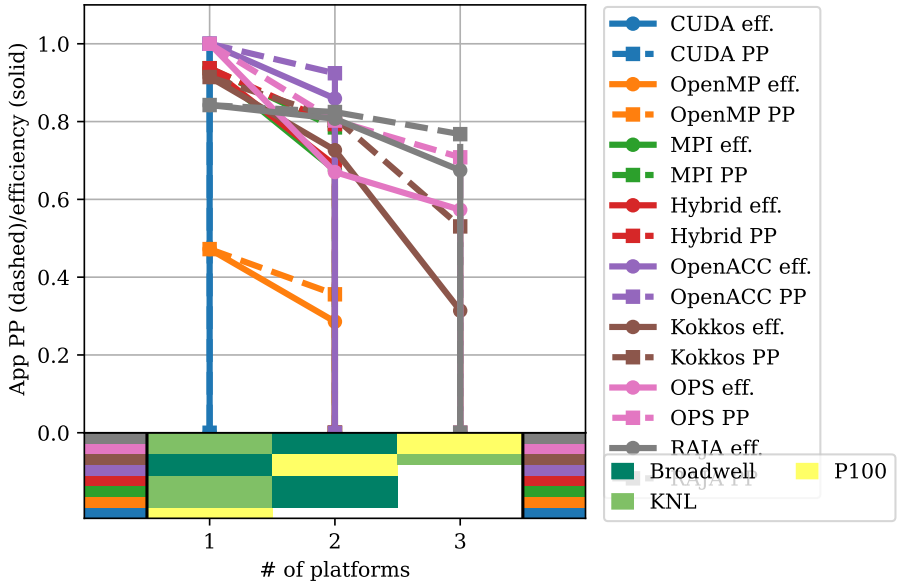


Figure 4: Cascade visualisation of performance portability from Kirk et al. [7]

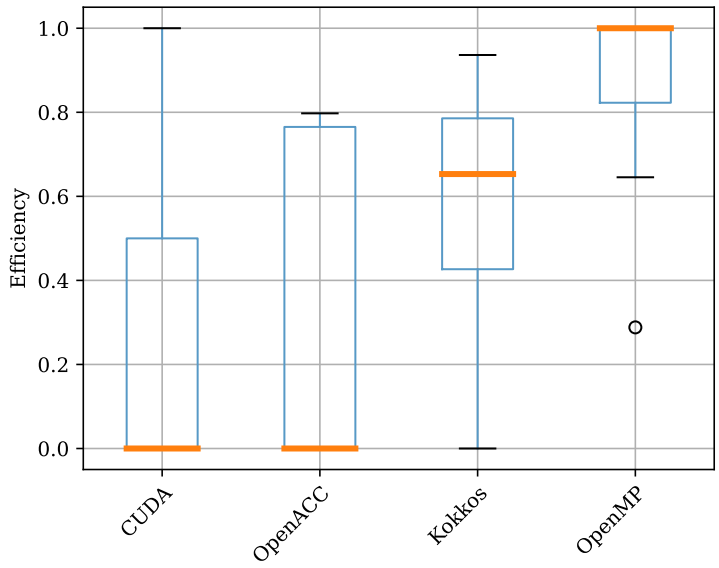


Figure 5: Box plot visualisation of performance portability from Deakin et al. [6]

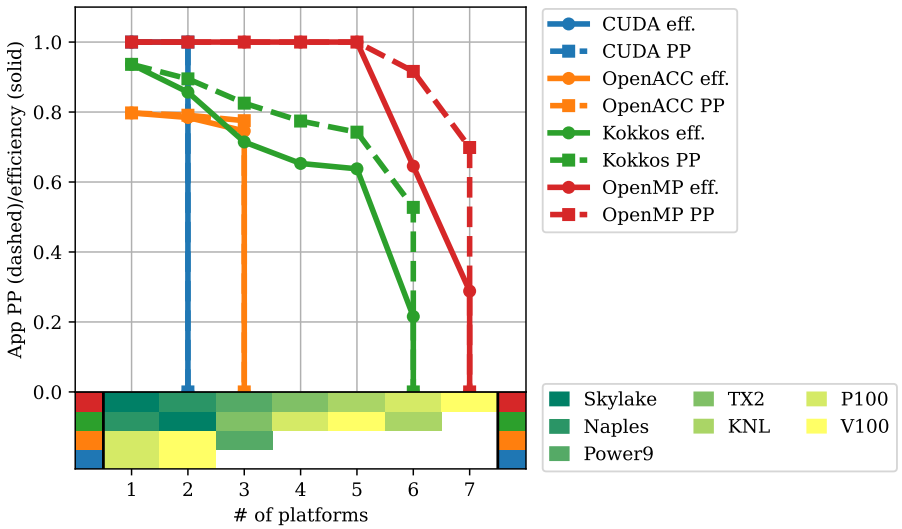


Figure 6: Cascade visualisation of performance portability from Deakin et al. [6]

2.2 miniFE

miniFE is a finite element mini-app, and part of the Mantevo benchmark suite [10, 11, 12, 13]. It implements an unstructured implicit finite element method

and has versions available in CUDA, Kokkos, OpenMP (3.0+ and 4.5+) and SYCL[8].

While there are a number of data sources for miniFE data, many of these are limited in scope, and so to ensure consistency, all data presented in this section has been newly gathered. In all cases, a $256 \times 256 \times 256$ problem size has been used, and all runs have been conducted on the platforms available on Isambard.

2.2.1 Performance

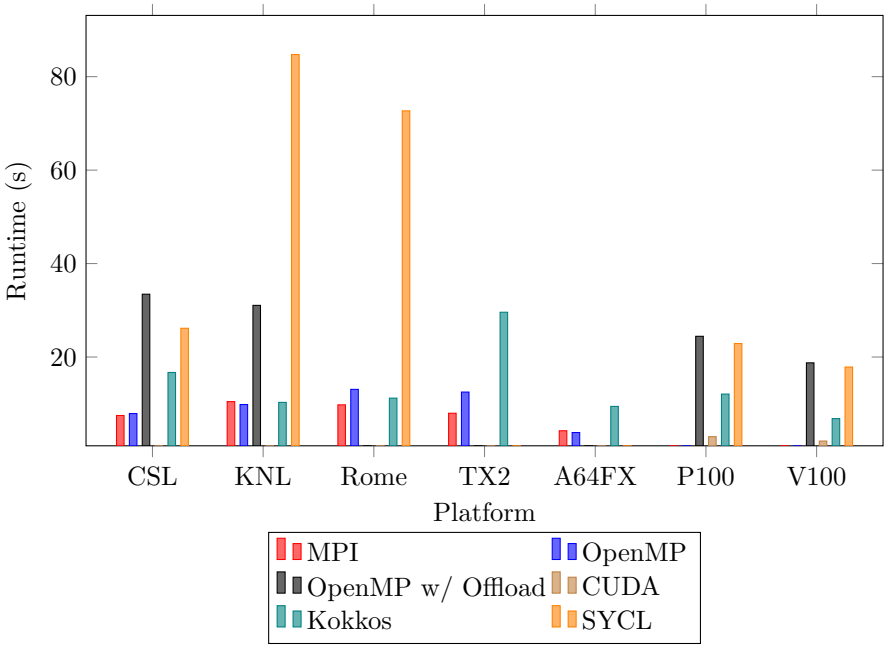


Figure 7: miniFE runtime data

The raw runtime results for these runs can be seen in Figure 7. It should be noted that the SYCL data is gathered from a miniFE port that can be found as part of the oneAPI-DirectProgramming github repository[9], and is compiled using hipSYCL and GCC. Data has not yet been collected for the ARM-based systems with SYCL. The OpenMP with offload variant of miniFE runs successfully on both AMD Rome and Cavium ThunderX2 platforms, but the runtimes

[8] `https://github.com/Mantevo/miniFE`
[9] `https://github.com/zjin-lcf/oneAPI-DirectProgramming/tree/master/miniFE-sycl`

are several orders of magnitude greater than all other platforms (likely due to an bug in the compiled code), and so have been removed.

In many of the miniFE ports available, only the conjugate solver has been parallelised effectively, so the results presented here represent only the timing from this kernel.

2.2.2 Performance Portability

Figures 8 and 9 present visualisations of the performance portability of miniFE, through various approaches.

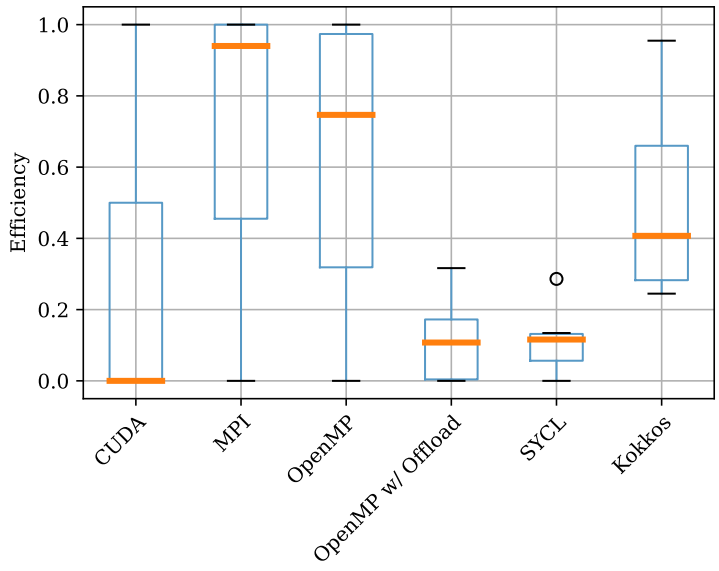


Figure 8: Box plot visualisation of performance portability of miniFE

Again, non-portable approaches (CUDA, OpenMP and MPI) often provide the best performance, but this leads to a significant restriction on the platforms that can be used. Figure 9 shows how the performance portability of miniFE evolves as more platforms are added for the Kokkos variant. While the performance is lower than native implementations, it has the advantage of being able to target every platform from a single codebase.

Likewise, both OpenMP with Offload and SYCL can target every platform,

though in some cases, we have not yet collected data (SYCL on Arm platforms), and in some cases, compiler issues are preventing reasonable performance.

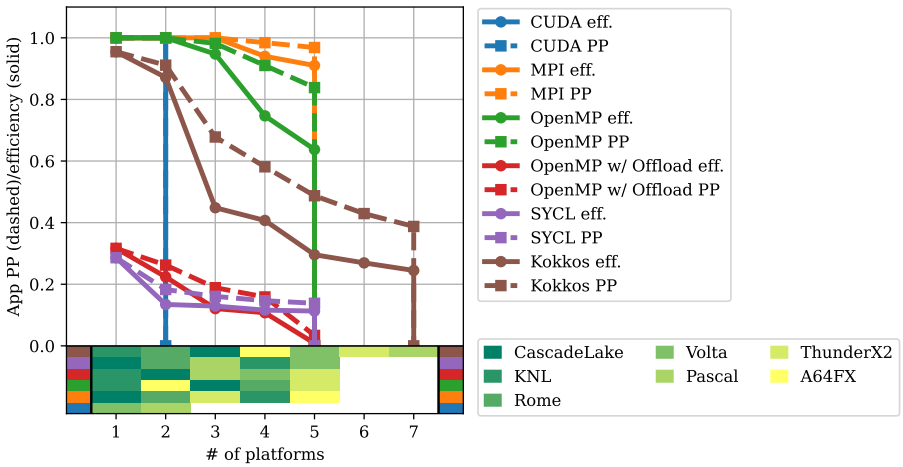


Figure 9: Cascade visualisation of performance portability of miniFE

2.3 Laghos

Laghos is a mini-app that is part of the ECP Proxy Applications suite [14, 15, 13]. It implements a high-order curvilinear finite element scheme on an unstructured mesh. The majority of the computation is performed by the HYPRE and MFEM libraries, and can thus use any programming model that is available for these libraries[10].

The results presented below have all been collected from the Isambard platform.

2.3.1 Performance

Figure 10 shows the runtime for Laghos, running problem #1 (Sedov blast wave), in three dimensions, up to 1.0 second of simulated time, using partial assembly (i.e., `./laghos -p 1 -dim 3 -rs 2 -tf 1.0 -pa -f`).

[10] `https://github.com/CEED/Laghos`

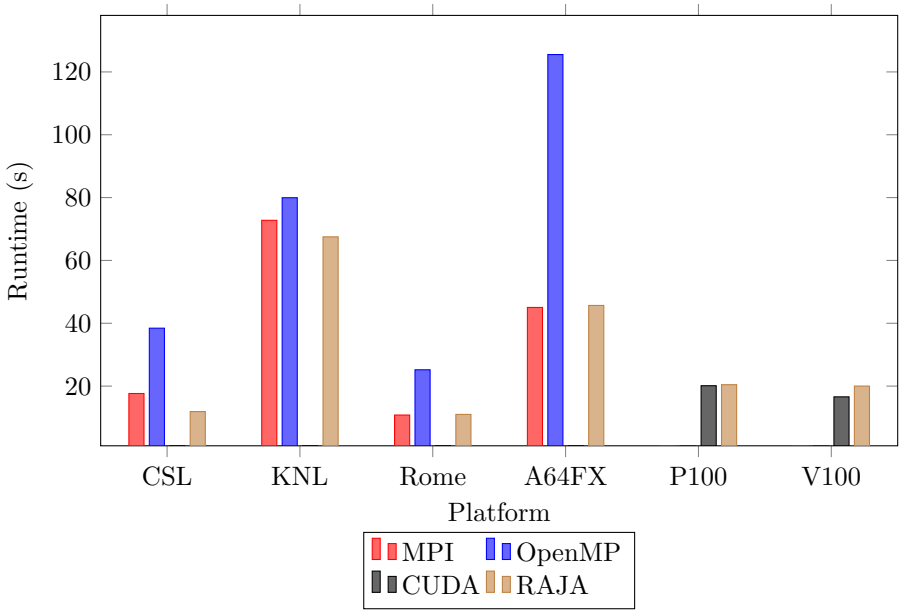


Figure 10: Laghos runtime data

2.3.2 Performance Portability

Portability visualisations of each implementation of Laghos are provided in Figures 11 and 12.

Again, it is clear that the highest achievable performance is often not from a portable approach. However, RAJA achieves the highest performance portability, due to being able to span each of the platforms, and provide near equal performance to the CUDA and MPI variants, and much better performance than the OpenMP implementation.

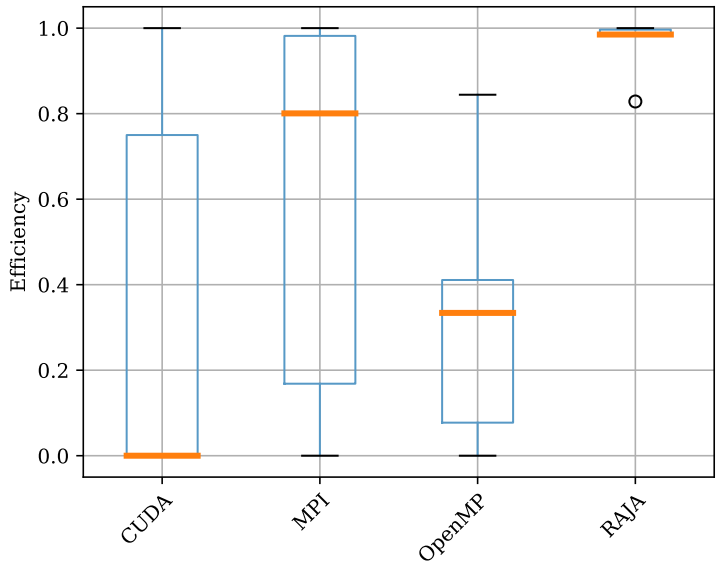


Figure 11: Box plot visualisation of performance portability of Laghos

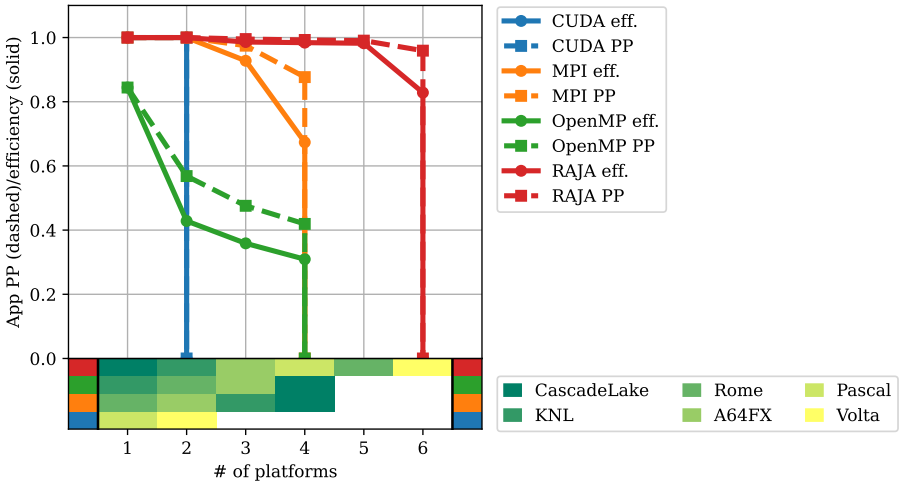


Figure 12: Cascade visualisation of performance portability of Laghos

2.4 CabanaPIC

CabanaPIC is a structured PIC demonstrator application built using the Co-PA/Cabana library for particle-based simulations [13]. The application uses Kokkos as its programming model for on-node parallelism and GPU use, and uses MPI for off-node parallelism[11].

2.4.1 Performance

Since there is only a single implementation of CabanaPIC, it is not possible for us to evaluate how the programming model affects its performance portability, however, we can show how its performance changes between architectures.

Figure 13 shows the achieved runtime for CabanaPIC across four of Isambard's platforms, running a simple 1D 2-stream problem with 6.4 million particles.

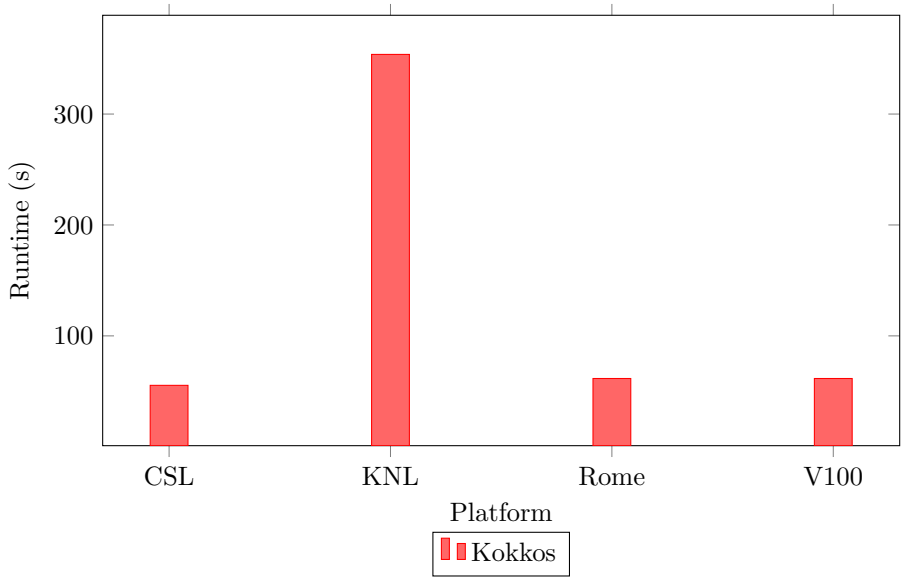


Figure 13: CabanaPIC data

Approximately equivalent performance can be seen on the CascadeLake, Rome and V100 systems. Similar to our previous Kokkos results on KNL, the runtime

[11] `https://github.com/ECP-copa/CabanaPIC`

is significantly worse than expected, possibly indicating a Kokkos bug, or a configuration issue.

2.5 VPIC

Vector Particle-in-Cell (VPIC) is a general purpose PIC code for modelling kinetic plasmas in one, two or three dimensions, developed at Los Alamos National Laboratory [16]. VPIC is parallelised on-core using vector intrinsics and on-node through a choice of pthreads or OpenMP. It can additionally be executed across a cluster using MPI[12].

Recently, a VPIC 2.0 [17] has been developed that adds support for heterogeneity by using Kokkos to optimise the data layout and allow execution on accelerator devices.

2.5.1 Performance

Figure 14 shows the runtime for the three variants of the VPIC code running on seven platforms[13]. This data is taken from the VPIC 2.0 study, comparing the non-vectorised, vectorised and Kokkos variants of the VPIC code. In each case, the runtime is the time taken for 500 time steps, with 66 millions particles.

2.5.2 Performance Portability

In terms of the performance portability of VPIC, we can see that the original and vectorised variants are only viable on the CPU architectures. Figures 15 and 16 visualise how the performance portability varies as more platforms are evaluated.

The highest performance in most cases comes from the vectorised variant of VPIC, as it achieves the best performance on all CPU platforms (except the ThunderX2, where no data is provided). However, Figure 15, when evaluating

[12] `https://github.com/lanl/vpic`
[13] `https://globalcomputing.group/assets/pdf/sc19/SC19_flier_VPIC.pptx.pdf`

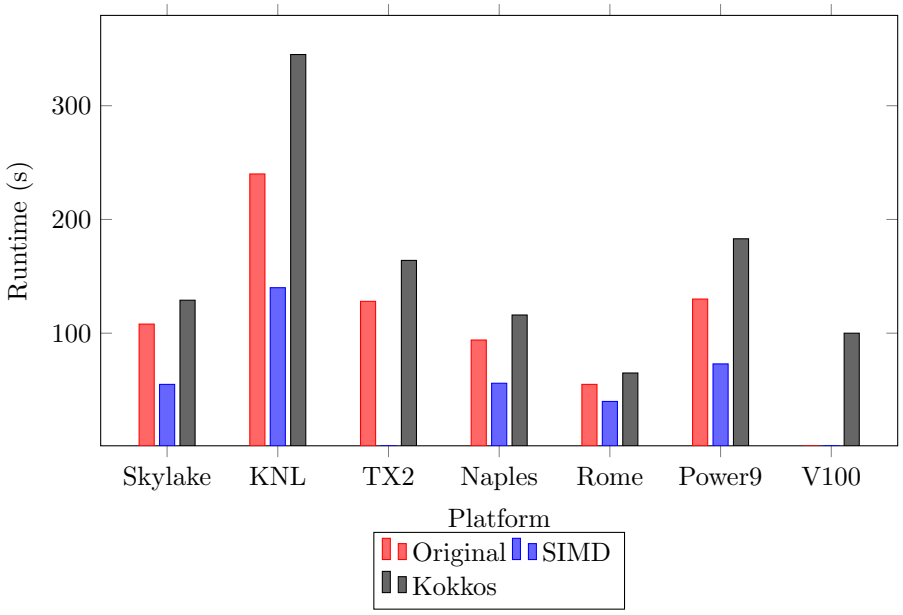


Figure 14: VPIC runtime data from Bird et al. [17]

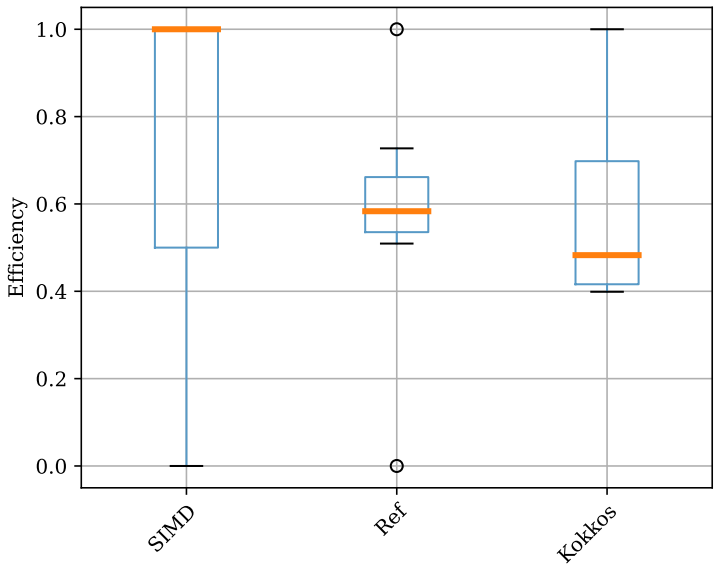


Figure 15: Box plot visualisation of performance portability of VPIC

the entire set of platforms, its performance portability would be 0, due to non-

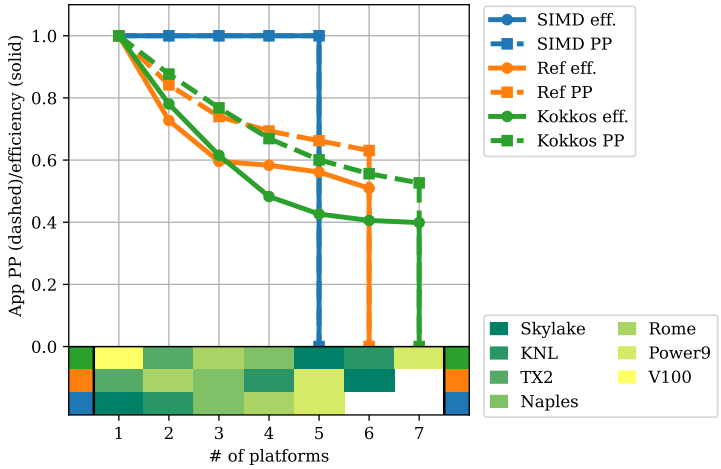


Figure 16: Cascade visualisation of performance portability of VPIC

execution on the V100 platform.

Figure 16 shows that while Kokkos performs worse than the vectorised implementation, its performance is similar the non-vectorised variant, but is also capable of execution on the V100 platform.

It should be noted that this data is from a study based on the initial implementation of VPIC using Kokkos. It is likely that these performance figures will be improved in future, potentially closing the performance gap on the vectorised implementation, while maintaining portability to heterogeneous architectures.

2.6 EMPIRE-PIC

EMPIRE-PIC is the particle-in-cell solver central the the ElectroMagnetic Plasma In Realistic Environments (EMPIRE) project [18]. It solves Maxwell's equations on an unstructured grid using a finite-element method, and implements the Boris push for particle movement. EMPIRE-PIC makes extensive use of the Trilinos library, and subsequently uses Kokkos as its parallel programming model [19, 20].

2.6.1 Performance

The EMPIRE-PIC application is export controlled, and thus the results in this section come from the study by Bettencourt et al. [19], looking specifically at the particle kernels within EMPIRE-PIC.

Figure 17 shows the runtime of the Accelerate, Weight Fields, Move and Sort kernels within EMPIRE-PIC for an electromagnetic problem with 16 million particles (8 million H+, 8 million e-). The geometry for this problem is the tet mesh that can be seen in Figure 7 in Bettencourt et al. [19].

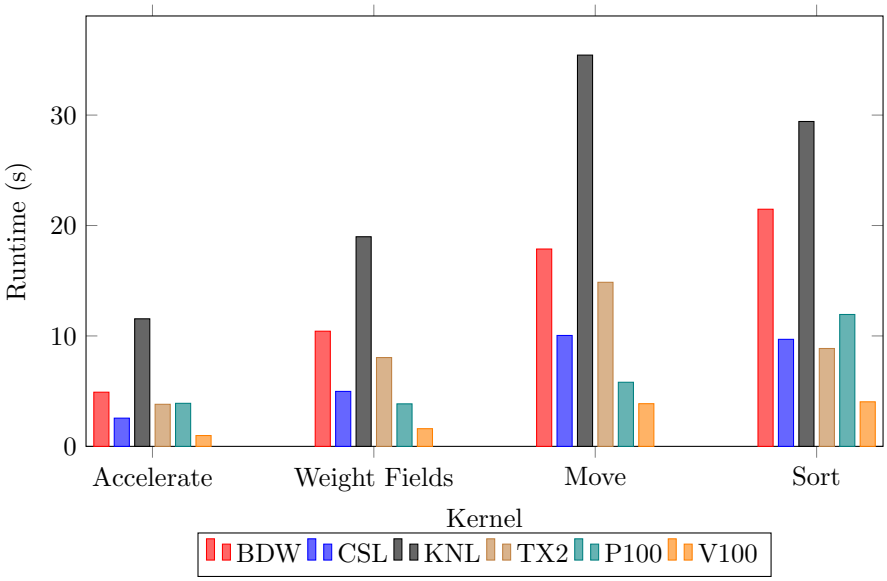


Figure 17: EMPIRE-PIC runtime data

2.6.2 Performance Portability

While there is only a single programming model implementation of EMPIRE-PIC, we can use the equations given in Table 2 of Bettencourt et al. [19] to calculate the FLOP/s achieved and compare this to each machines maximum performance, thus calculating the architectural efficiency. The equations presented assume the best case performance, whereby particles are evenly distributed across the domain, there is no particle migration throughout the simulation, and they are sorted at the start of the simulation. Nevertheless, they

provide a useful opportunity to analyse the performance portability of Kokkos for particle-based kernels.

Figures 18 and 19 provide visualisations of EMPIRE-PIC's performance portability across six platforms.

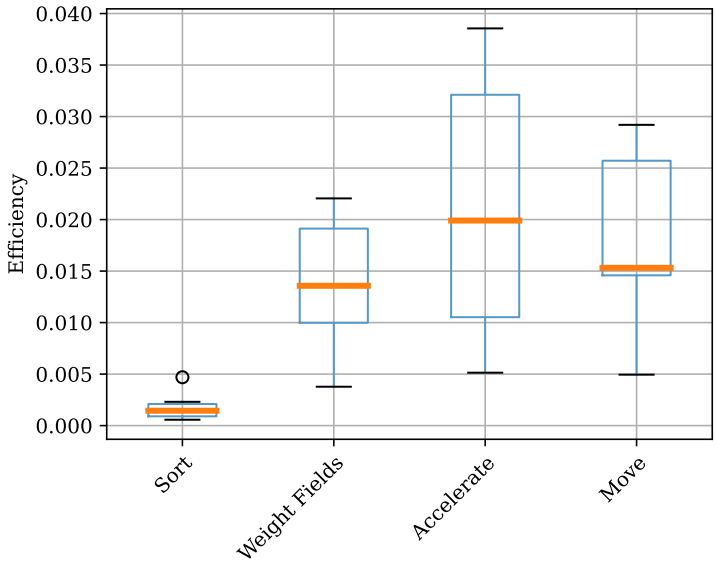


Figure 18: Box plot visualisation of performance portability for four particle kernels in EMPIRE-PIC

It is important to note that although Figure 18 shows incredibly low efficiency, this is compared to each platform's peak performance, where a vectorised fused-multiply-add instruction must be executed each clock cycle. Achieving less than 10% of this peak performance is not unusual for a real application. In the case of the Sort kernel, the efficiency is lower still, as this is not a kernel that is bound by floating point performance.

What is clear from the performance portability visualisations is that the variance in achieved efficiency between platforms is not large, indicating that Kokkos is able to achieve a similar portion of the available performance for EMPIRE-PIC's particle kernels. Achieved efficiency is higher on the ThunderX2 and Broadwell systems, due to less reliance on well vectorised code, and a lower available peak performance.

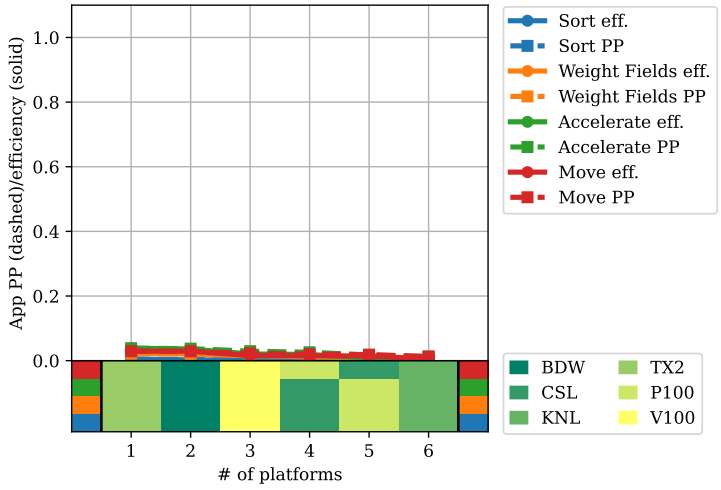


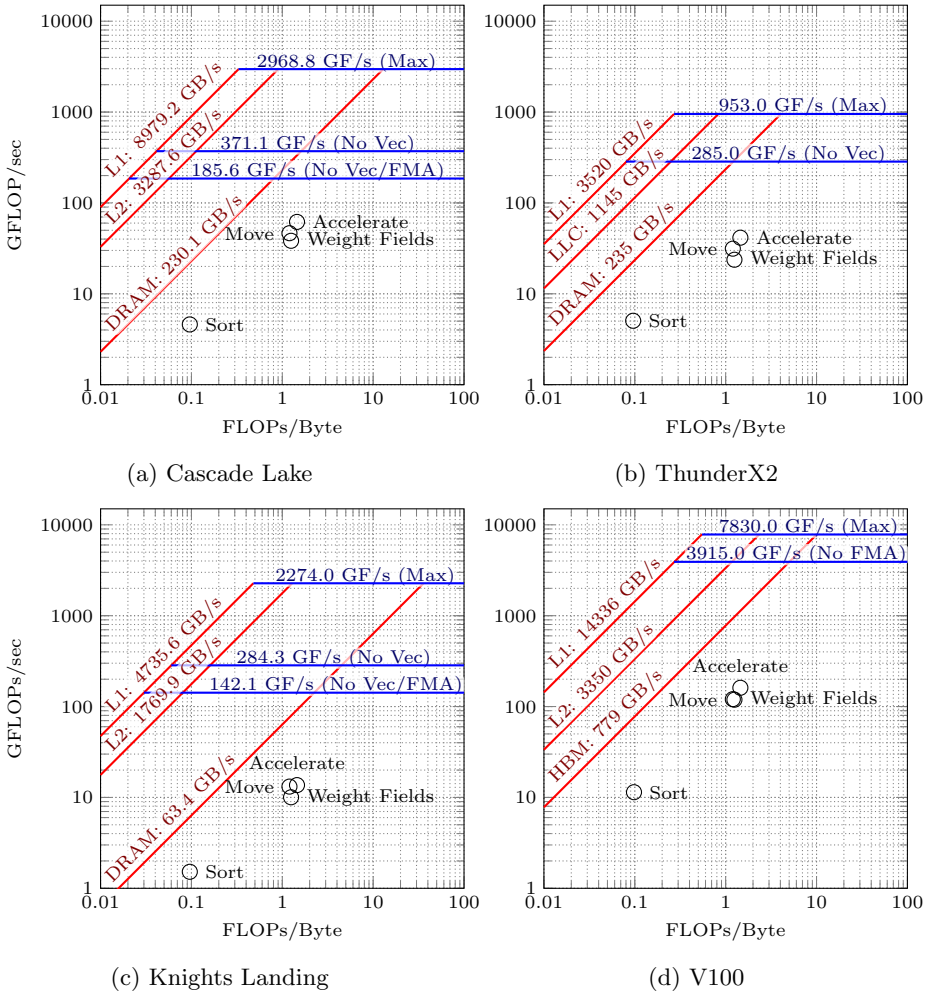
Figure 19: Cascade visualisation of performance portability for four particle kernels in EMPIRE-PIC

This data perhaps suggests that EMPIRE-PIC is not able to fully exploit the available on-core parallelism available through vectorisation. We can further support this analysis using Roofline models for the Cascade Lake, KNL, ThunderX2 and V100 systems. Figure 20 shows roofline models for these four platforms, with the four particle kernels plotted according to their arithmetic intensity and achieved FLOPs/s.

In all cases, we can see that the application is not successfully using vectorisation (and this is confirmed by compiler reports). As stated in Bettencourt et al. [19], the control flow required to handle particles crossing element boundaries leads to warp divergence on GPUs and makes achieving vectorisation difficult on CPUs. Nonetheless, on the Cascade Lake and ThunderX2 platforms, we are within an order of magnitude of the non-vectorised peak performance for the three main kernels, and we can observe that all four kernels are memory bound. For the two many-core architectures (KNL and V100), floating-point performance is further from the peak, but all kernels are likewise memory bound by available DRAM/HBM bandwidth.

Figure 20 demonstrates how vital efficient memory accesses are for achieving high performance in PIC codes, due to the relatively low arithmetic intensity of the kernels when compared to the amount of bytes that need to be moved

to and from main memory. An alternative approach to the FEM-PIC method has been explored using EMPIRE-PIC by Brown et al. [20], whereby complex particle shapes are supported using virtual particles based on quadrature rules. Using virtual particles in this manner can increase the arithmetic intensity of particle kernels without requiring significantly more data to be moved from and to main memory.



(a) Cascade Lake

(b) ThunderX2

(c) Knights Landing

(d) V100

Figure 20: Roofline plots on four platforms, gathered using the Empirical Roofline Toolkit [21]

3 Conclusions

This report serves as a living document of the performance of applications that implement algorithms of interest to the NEPTUNE project. For each of the applications in this report, there are typically a number of alternative implementations, solving the same algorithm but using a different parallel programming model. This allows us an opportunity to assess these programming models and their appropriateness for the NEPTUNE project, with the goal of creating a set of best practices to developing plasma physics applications that are both *performant* and *portable*.

The results presented in the previous section show that in most cases, OpenMP and/or MPI provide the best performance on CPU platforms, while CUDA typically provides the best performance on NVIDIA GPUs. However, these programming models significantly affect the portability of these applications, with the former unable to use accelerators, and the latter unable to use host platforms. Developing an application that can exploit all available parallelism that is likely to be present on post-Exascale systems would therefore require developers to maintain multiple implementations of a code – potentially one for each class/generation of host or accelerator platforms.

For fluid codes, there are a number of domain specific languages (DSLs) that provide abstractions for grid-based algorithms. OPS is one such DSL targeted at structured mesh applications, and capable of code generation targeting MPI, OpenMP, OpenACC, CUDA and HIP. Our study with TeaLeaf shows that it is able to provide performance that in many cases is on par with native OpenMP and MPI, and within 2× native CUDA performance on a P100. However, such DSLs often reduce the flexibility afforded to a developer.

Besides code generation from a higher-level abstraction, GPUs can be targeted using pragma-based language extensions such as OpenMP 4.5 and OpenACC. Both offer similar functionality, but only OpenMP 4.5 allows portability between accelerator and non-accelerator platforms. However, our evaluation has shown that although OpenMP 4.5 allows us to target GPUs, different pragmas are often required to achieve sufficient performance on accelerators when compared to host systems, meaning that multiple implementations would likely need to be maintained. This is well demonstrated by our miniFE results, where the

OpenMP with offload code does successfully execute on the CPU architectures but offers significantly worse performance than OpenMP itself.

The template libraries, Kokkos and RAJA are both capable of providing full portability across all architectures, and in most cases offer good performance. The significant exception from our results is for the Intel Knights Landing platform, where Kokkos performance is typically poor. This performance gap is likely the result of a bug or memory configuration issue, but will not be investigated further due to the discontinuation of the KNL architecture. Regardless, where we are able to compare Kokkos or RAJA to a native programming model, they are typically able to achieve a runtime that is no more than 20% greater than the native programming model on CPUs and no more than 50% greater than the native programming model on GPUs, but from a single code base.

Another approach that is gaining traction is that of SYCL/DPC++. In our current benchmark set, only a single application is available implemented in SYCL (miniFE), and that implementation has been generated using Intel's DPC++ Compatibility Toolkit. The resulting application is portable across platforms but in most cases has performance that is only slightly better than the available OpenMP 4.5 implementation. This warrants additional exploration to account for this performance difference; for such an immature programming model, it is likely that choice of compiler, and some very simple optimisations will bring performance more inline with other approaches to portability. As this project progresses, hopefully more applications will be available for evaluation, and compiler support will evolve.

For the particle methods tranche of applications, they are predominantly available using Kokkos as a parallel programming model. This does allow portable execution across all available platforms, but makes it difficult to compare performance against native implementations. In the case of VPIC, we can see that Kokkos provides performance that is inline with the original, unvectorised implementation on all platforms, and allows us to extend our platform set to include GPU devices. However, the greatest performance comes from using non-portable vector intrinsics, which in this case means maintaining an implementation for each set of vector instructions (i.e. SSE, AVX, AVX-2, Altivec, etc.).

3.1 Limitations

The work presented in this report represents our initial evaluation of approaches to performance portability. We intend that this document is continually updated as new data becomes available, and as applications and implementations are developed. Currently, the data in this report contains a few limitations that we aim to rectify in future.

Firstly, due to its immaturity relative to other approaches, there are a lack of relevant fluid and particle-in-cell applications available that use the SYCL/DPC++ programming model. This means, that with the exception of miniFE, it is difficult to assess its appropriateness as an approach to performance portable application development. A recent study has by Reguly et al. shows that for a computational fluid dynamic application SYCL may be able to achieve comparable performance, though this may require different code paths for different hardware [22].

Secondly, the PIC codes assessed in this report all use the Kokkos programming model. Again, this limits our ability to reason about the appropriateness of this approach for PIC codes, but we can use the VPIC data to show that while we cannot match native, hand-vectorised performance, it can provide performance that is similar to the original implementation, and can be extended to heterogeneous architectures.

Finally, we have not currently evaluated performance on any AMD Radeon Instinct or Intel Xe hardware, due to availability of test platforms. We aim to add these platforms in the near future, when available, either through the COSMA8 system at Durham University, or through Amazon EC2 instances.

References

[1] S.J. Pennycook, J.D. Sewall, and V.W. Lee. Implications of a metric for performance portability. *Future Generation Computer Systems*, 92:947 – 958, 2019.

[2] Jason Sewall, S. John Pennycook, Douglas Jacobsen, Tom Deakin, and Simon McIntosh-Smith. Interpreting and visualizing performance portabil-

ity metrics. In *2020 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*, pages 14–24, 2020.

[3] B D Dudson, M V Umansky, X Q Xu, P B Snyder, and H R Wilson. BOUT++: A framework for parallel plasma fluid simulations. *Computer Physics Communications*, 180:1467–1480, 2009.

[4] C.D. Cantwell, D. Moxey, A. Comerford, A. Bolis, G. Rocco, G. Mengaldo, D. De Grazia, S. Yakovlev, J.-E. Lombard, D. Ekelschot, B. Jordi, H. Xu, Y. Mohamied, C. Eskilsson, B. Nelson, P. Vos, C. Biotto, R.M. Kirby, and S.J. Sherwin. Nektar++: An open-source spectral/hp element framework. *Computer Physics Communications*, 192:205–219, 2015.

[5] T D Arber, K Bennett, C S Brady, A Lawrence-Douglas, M G Ramsay, N J Sircombe, P Gillies, R G Evans, H Schmitz, A R Bell, and C P Ridgers. Contemporary particle-in-cell approach to laser-plasma modelling. *Plasma Physics and Controlled Fusion*, 57(11):113001, sep 2015.

[6] Tom Deakin, Simon McIntosh-Smith, James Price, Andrei Poenaru, Patrick Atkinson, Codrin Popa, and Justin Salmon. Performance portability across diverse computer architectures. In *2019 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*, pages 1–13, 2019.

[7] R. O. Kirk, G. R. Mudalige, I. Z. Reguly, S. A. Wright, M. J. Martineau, and S. A. Jarvis. Achieving Performance Portability for a Heat Conduction Solver Mini-Application on Modern Multi-core Systems. In *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 834–841, Sep. 2017.

[8] Matthew Martineau, Simon McIntosh-Smith, and Wayne Gaudin. Assessing the performance portability of modern parallel programming models using tealeaf. *Concurrency and Computation: Practice and Experience*, 29(15):e4117, 2017.

[9] Simon McIntosh-Smith, Matthew Martineau, Tom Deakin, Grzegorz Pawelczak, Wayne Gaudin, Paul Garrett, Wei Liu, Richard Smedley-Stevenson, and David Beckingsale. TeaLeaf: A Mini-Application to Enable Design-Space Explorations for Iterative Sparse Linear Solvers. In *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 842–849, 2017.

[10] Richard Frederick Barrett, Li Tang, and Sharon X. Hu. Performance and Energy Implications for Heterogeneous Computing Systems: A MiniFE Case Study. 12 2014.

[11] Alan B. Williams. Cuda/GPU version of miniFE mini-application. 2 2012.

[12] Meng Wu, Can Yang, Taoran Xiang, and Daning Cheng. The research and optimization of parallel finite element algorithm based on minife. *CoRR*, abs/1505.08023, 2015.

[13] David F. Richards, Yuri Alexeev, Xavier Andrade, Ramesh Balakrishnan, Hal Finkel, Graham Fletcher, Cameron Ibrahim, Wei Jiang, Christoph Junghans, Jeremy Logan, Amanda Lund, Danylo Lykov, Robert Pavel, Vinay Ramakrishnaiah, et al. FY20 Proxy App Suite Release. Technical Report LLNL-TR-815174, Exascale Computing Project, September 2020.

[14] J. C. Camier. Laghos summary for CTS2 benchmark. Technical Report LLNL-TR-770220, Lawrence Livermore National Laboratory, March 2019.

[15] Robert Anderson, Julian Andrej, Andrew Barker, Jamie Bramwell, Jean-Sylvain Camier, Jakub Cerveny, Veselin Dobrev, Yohann Dudouit, Aaron Fisher, Tzanio Kolev, Will Pazner, Mark Stowell, Vladimir Tomov, Ido Akkerman, Johann Dahm, David Medina, and Stefano Zampini. Mfem: A modular finite element methods library. *Computers & Mathematics with Applications*, 81:42–74, 2021. Development and Application of Open-source Software for Problems with Numerical PDEs.

[16] K. J. Bowers, B. J. Albright, B. Bergen, L. Yin, K. J. Barker, and D. J. Kerbyson. 0.374 Pflop/s Trillion-Particle Kinetic Modeling of Laser Plasma Interaction on Roadrunner. In *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, SC '08. IEEE Press, 2008.

[17] Robert Bird, Nigel Tan, Scott V Luedtke, Stephen Harrell, Michela Taufer, and Brian Albright. VPIC 2.0: Next Generation Particle-in-Cell Simulations. *IEEE Transactions on Parallel and Distributed Systems*, pages 1–1, 2021.

[18] Matthew T. Bettencourt and Sidney Shields. EMPIRE Sandia's Next Generation Plasma Tool. Technical Report SAND2019-3233PE, Sandia National Laboratories, March 2019.

[19] Matthew T. Bettencourt, Dominic A. S. Brown, Keith L. Cartwright, Eric C. Cyr, Christian A. Glusa, Paul T. Lin, Stan G. Moore, Duncan A. O. McGregor, Roger P. Pawlowski, Edward G. Phillips, Nathan V. Roberts, Steven A. Wright, Satheesh Maheswaran, John P. Jones, and Stephen A. Jarvis. EMPIRE-PIC: A Performance Portable Unstructured Particle-in-Cell Code. *Communications in Computational Physics*, x(x):1–37, March 2021.

[20] Dominic A.S. Brown, Matthew T. Bettencourt, Steven A. Wright, Satheesh Maheswaran, John P. Jones, and Stephen A. Jarvis. Higher-order particle representation for particle-in-cell simulations. *Journal of Computational Physics*, 435:110255, 2021.

[21] Yu Jung Lo, Samuel Williams, Brian Van Straalen, Terry J. Ligocki, Matthew J. Cordery, Nicholas J. Wright, Mary W. Hall, and Leonid Oliker. Roofline Model Toolkit: A Practical Tool for Architectural and Program Analysis. In Stephen A. Jarvis, Steven A. Wright, and Simon D. Hammond, editors, *High Performance Computing Systems. Performance Modeling, Benchmarking, and Simulation*, pages 129–148. Springer International Publishing, 2015.

[22] Istvan Z. Reguly, Andrew M. B. Owenson, Archie Powell, Stephen A. Jarvis, and Gihan R. Mudalige. Under the Hood of SYCL – An Initial Performance Analysis with An Unstructured-Mesh CFD Application. In Bradford L. Chamberlain, Ana-Lucia Varbanescu, Hatem Ltaief, and Piotr Luszczek, editors, *High Performance Computing*, pages 391–410. Springer International Publishing, 2021.