Interpreting and Visualizing Performance Portability Metrics

Jason Sewall*, S. John Pennycook*, Douglas Jacobsen*, Tom Deakin† and Simon McIntosh-Smith†
*Intel Corporation, Santa Clara, USA
Email: {jason.sewall, john.pennycook, douglas.w.jacobsen}@intel.com
†Department of Computer Science, University of Bristol, UK
Email: {tom.deakin, s.mcintosh-smith}@bristol.ac.uk
All authors have equal contribution to this work

***Abstract*—Recent work has introduced a number of tools and techniques for reasoning about the interplay between application performance and portability, or "performance portability". These tools have proven useful for setting goals and guiding high-level discussions, but our understanding of the performance portability problem remains incomplete. Different views of the same performance efficiency data offer different insights into an application's performance portability (or lack thereof): standard statistical measures such as the mean and standard deviation require careful interpretation, and even metrics designed specifically to measure performance portability may obscure differences between applications.**

This paper offers a critical assessment of existing approaches for summarizing performance efficiency data across different platforms, and proposes visualization as a means to extract useful information about the underlying distribution. We explore a number of alternative visualizations, outlining a new methodology that enables developers to reason about the performance portability of their applications and how it might be improved. This study unpicks what it might mean to be "performance portable" and provides useful tools to explore that question.

***Index Terms*—high performance computing; performance portability**

I. Introduction

There has been a growing volume of work in the deeply interconnected space of performance, productivity, and portability that aims to provide useful tools and answers to developers. This is a fledgling area of research, and each contribution measurably grows our ability to reason about these complex notions.

In previous work, we proposed a formal definition for performance portability and a metric based on this definition for characterizing an application's performance portability [1], [2]. Although this metric has proven an effective way to discuss performance portability in objective terms [3]–[7], it cannot answer all questions that a developer might have about their application's performance portability [8]–[12].

This paper therefore seeks to explore this problem space more deeply. What does it mean to ask for performance, portability and productivity? How does it vary between developers, and what do the current metrics and tools offer in service of these questions? In attempting to provide answers to these questions, we make the following contributions:

1) We propose a set of six synthetic data sets, designed to assess the intuition gained from different performance portability metrics and visualizations.
2) We provide a critical assessment of existing approaches to summarize performance portability, including measures of average performance efficiency and consistency.
3) We demonstrate a number of novel visualizations for expressing the performance efficiency of applications while preserving insight into the underlying distribution of performance efficiency across platforms.

II. Background / Motivation

A. Applications, Platforms, Problems, and Semantic Pitfalls

The definition of performance portability proposed in [1], [2] relies on specific interpretations of certain fundamental terms so as to be precise. It is worth reviewing these terms and how they figure into a comprehensive study of an application's performance portability.

Problem: A task with a pass/fail correctness metric and for which quantitative performance may be measured.

Application: A collection of software capable of running a given *problem* with measurable correctness and performance.

Platform: A collection of software and hardware on which an *application* may run a *problem*.

Performance Efficiency: Either: *architectural* efficiency, computed relative to the theoretical capabilities of a platform (*e.g.* peak instruction throughput or memory bandwidth); or *application* efficiency, computed relative to the best performance previously demonstrated to be achievable on a platform (*i.e.* by another implementation of the same application).

These terms are intended to accommodate broad inputs as well as unburden readers from preconceived notions of what they mean. When critically viewing performance portability results, it is of the highest importance that the realizations of these terms be understood in the context of these definitions and what the author of the results has ascribed to them.

To emphasize some of the meaningful subtleties, consider a hypothetical molecular dynamics package. This package is capable of running a variety of molecular simulation scenarios, and considerable work has been put into supporting a wide variety of hardware and operating systems. In some cases,

TABLE I: Examples of meaningful interpretations of terms. Entries marked † represent varying components of the study.

Problem(s)	Application(s)	Platform(s)
Protein-folding scenario	MD package	CPUs, GPUs†
Protein-folding scenario	MD package	CPUs†
Protein-folding scenario	MD package	Compilers†
Various MD problems†	MD package	CPUs, GPUs†
Protein-folding scenario	Implementations†	CPUs, GPUs†
Suite of MD codes	Frameworks/languages†	CPUs, GPUs†

large amounts of code specific to a certain class of hardware (for example, GPUs) has been written.

There are many different studies we might wish to perform using this package, examples of which are listed in Table I. Perhaps the most obvious is to think of a particular input scenario as the problem, the whole of the package as the application, and a small group of different systems with varied hardware and operating systems as the platforms. A semantic shift can produce novel studies that recycle the same components. Keeping the problem the same, we could consider the language frameworks and their respective underlying hardware as platforms and treat the individual, disparate implementations within the package as applications. This would reveal the aggregate efficiencies of the language frameworks, rather than hide that within a monolithic package-level application. Alternatively, one could combine several codes into one suite (*e.g.* a benchmark suite) and use some aggregated performance metric to measure success/failure across the suite as a whole.

B. Challenges

It is important to acknowledge that developers may be interested in performance portability for different reasons, and that there are several aspects of an application's performance portability that they may wish to explore:

1) **Is it performance portable?**
   This seemingly simple question may be asked by developers seeking to measure progress towards a goal of performance portability. While metrics such as those in [1], [2], [10], [11] provide ways to *measure* application performance portability, whether something *is* performance portable or not is based on inherently subjective criteria – the set of platforms that must be supported, and the performance efficiency that must be achieved on each, will vary by developer and application.
2) **What performance does it achieve "on average"?**
   A developer may not have full control over where their applications are ultimately executed. For example, researchers may be assigned time on multiple machines of different architectures. Understanding the expected performance across a set of candidate machines may therefore assist developers in ensuring that their applications run "fast enough" wherever they are scheduled, or help end-users to choose which libraries to leverage.
3) **How consistently does it perform across platforms?**
   The answer to Question 2 does not address how the performance efficiencies of an application are *distributed*, obscuring an application's affinity for different platforms and inputs – applications with similar average efficiencies may be tuned for different architectures [12] (e.g. CPU or GPU) or exhibit different scaling behaviors [10].
4) **What performance is expected on new platforms?**
   Many developers interested in performance portability are looking to future-proof their codebase against unknown changes in architecture. Predicting the level of performance on a hypothetical new platform is clearly impossible (since the target architecture is unknown), but analyzing an application's sensitivity to changes in platforms may be a suitable proxy. Analytical performance modelling is another approach for this prediction.

Hardware-agnostic languages and frameworks enabling developers to support multiple platforms with a single source code (such as OpenCL [13], SYCL [14], Kokkos [15] and RAJA [16] to name but a few) have been developed and deployed in service of performance portability. The nature of these languages/frameworks leads to a fifth question:

5) **How difficult is it to write/maintain?**
   Maintaining a separate highly-tuned implementation for every platform is the most obvious way to guarantee high performance efficiency everywhere, but the amount of effort required by this approach is unacceptable to many developers. We believe that separating measures of performance from measures of productivity (*e.g.* code divergence [8]) is a more promising direction than a combined performance-portability-productivity metric [11], and will not explore this question further.

Throughout, it is important to remember that the platform set of interest to a developer may evolve over time. As this set diversifies, a developer's definition of *acceptable* performance portability may also change. For example, a drop in average performance might be acceptable if it allows a more diverse platform set to be utilized or makes an application more amenable to supporting new architectures in the future. We make no attempt in this paper to define what an acceptable level of performance portability is; our focus is on assisting developers in making determinations about their own codes.

C. Data Sets

The real-world performance data in this paper is taken from a study by Deakin et al. [17]. We include results from three codes, written in five programming models: OpenMP, OpenCL, Kokkos, OpenACC and CUDA. Note that by the definitions in Section II-A, each combination of code and programming model constitutes an "application". All combinations were tested on a wide-ranging set of twelve platforms.

We have additionally constructed a number of synthetic data sets (**unportable**, **single target**, **multi-target**, **consistent** and **inconsistent**), shown as a heatmap in Figure 1. Each column represents a different data set and its performance efficiency over a range of hypothetical platforms (A-J). Each data set is designed and named to reflect the characteristics of

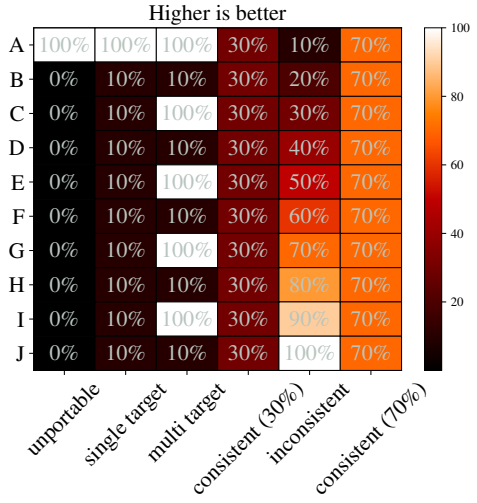


Fig. 1: Heatmap of synthetic application efficiencies.

a distinct efficiency distribution that could realistically arise during application development; taken together, these data sets can be used as a litmus test for the intuition gained from different metrics and visualizations. We believe that synthetic distributions like these could serve a similar role to mathematical basis functions or classifiers in the space of performance efficiency distributions, but leave this investigation to future work. A detailed description of each data set is given below.

Unportable: Applications in this class are written to support only a single target platform, possibly using a proprietary programming model that excludes some classes of platforms. This is represented by an efficiency of zero on one or more platforms (where the application does not run).

Single Target: Applications may be written in a portable model but with significant optimization efforts applied to a limited set of platforms. For example, if an application is only ever expected to run on one platform, that platform will be the development team's optimization focus. However, the application remains capable of achieving non-zero efficiencies (*i.e.* it will run correctly) on the remaining platforms.

Multi-Target: A multi-target application shares many similarities with one targeting a single platform, but expands the target to a larger set of platforms. For example, the application might be optimized specifically for one class of architectures (such as GPUs). The result is often a bimodal distribution of efficiencies, with two distinct groups: one high and one low.

Consistent: Some applications achieve similar performance efficiencies on all platforms. This may result from the nature of the application (*e.g.* a micro-benchmark intended to stress a particular architectural feature) or project goals (*e.g.* a desire for a code to attain similarly high levels of performance across all platforms [17]). We use two distributions to represent such applications, separating cases where the efficiency is consistently low (30%) and high (70%), to determine whether the approaches tested can differentiate between them.

Inconsistent: An application that prioritizes portability over performance is unlikely to achieve consistent levels of performance efficiency. Such applications may be continually adapted as they move between platforms and developers add new features, resulting in wide-ranging results with high variance. We represent this as a linearly increasing performance efficiency, but could have drawn these numbers from a uniform distribution.

TABLE II: "Averages" of synthetic efficiency data.

	Unportable	Single Target	Multi Target	Consistent (30%)	Inconsistent	Consistent (70%)
Minimum	0.0	10.00	10.00	30.0	10.00	70.0
Arith. Mean	10.0	19.00	55.00	30.0	55.00	70.0
Geo. Mean	0.0	12.59	31.62	30.0	45.29	70.0
Har. Mean	NaN	10.99	18.18	30.0	34.14	70.0
Median	0.0	10.00	55.00	30.0	55.00	70.0
$\mathcal{P}$	0.0	10.99	18.18	30.0	34.14	70.0

III. Single Number Metrics

Representing the performance portability of an application using a single numeric value is highly desirable, providing a simple way to compare the distributions of performance efficiency results across applications.

When confronted with a slew of data, it is tempting to reach for familiar statistical measures (*e.g.* average and standard deviation); Question 2 and Question 3 may be answered directly, with Question 1 answered by defining a value (or range of values) representing a sufficiently performance portable application. However, as we will see, common statistical tools may not provide the insight we're looking for. It is also highly unlikely that any single number could answer all three of these questions simultaneously; combining multiple metrics – or using different metrics for different analyses – may be necessary.

A. Average Performance Efficiency

Table II compares several "averages" computed for our synthetic data set. In addition to standard statistical averages (arithmetic mean, geometric mean, harmonic mean, median), we include the minimum value and the value of the performance portability metric from [1], [2], calculated as:

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

that is, the harmonic mean of application a's performance efficiency $e(a,p)$ when executing problem p on a set of platforms H. $\mathcal{P}$ differs from the harmonic mean only in that the latter is not defined if the input data contains a zero (since it would require the computation of $1/0$).

The ranking of the synthetic data sets according to the performance portability metric is the most aligned with the authors' intuition of which hypothetical application is the most performance portable (from least to most): 1) **unportable**;

TABLE III: "Consistencies" of synthetic efficiency data.

	Unportable	Single Target	Multi Target	Consistent (30%)	Inconsistent	Consistent (70%)
SD	31.62	28.46	47.43	0.0	30.27	0.0
Har. SD[1]	NaN	3.44	15.68	0.0	32.32	0.0
Har. SD[2]	NaN	3.16	15.81	0.0	40.86	0.0
MAD	0.00	0.00	45.00	0.0	25.00	0.0
Range	100.00	90.00	90.00	0.0	90.00	0.0

[1]As calculated in [18], [19]; [2]As calculated in [20]

2) **single target**; 3) **multi-target**; 4) **consistent (30%)**; 5) **inconsistent**; 6) **consistent (70%)**. However, whilst we are confident that the rankings of the extremes of this list are intuitive, the middle rankings are less obvious – for example, ordering **inconsistent** and **consistent (30%)** depends on the importance an individual developer assigns to consistency (since the **inconsistent** application has higher absolute performance on all but two of the platforms).

For the most part, the other metrics agree on which of the hypothetical applications in our synthetic data sets have the best and worst performance portability. However, as expected, some of the metrics are unable to distinguish between the different distributions: the minimum value is identical for three of the distributions; and the arithmetic mean is identical for the **multi-target** and **inconsistent** distributions.

The arithmetic mean also highlights the difficulty of handling "did not run" results with standard statistical measures. Performance efficiencies are naturally constrained to the range $[0, 1]$, where the two extremes convey extra information (*i.e.* 0 represents "did not run" and 1 represents "best performance"). Whether or not an application is portable is key to its performance portability, and giving "did not run" results the same weighting as poor performance results is misleading. The sensitivity of the arithmetic mean to large values makes it most liable to obscure performance efficiencies of 0, but the median can suffer similarly. Handling these problematic results is a theme that we will revisit throughout the paper.

B. Consistent Performance Efficiency

Some definitions of performance portability [17], [21], [22] talk of achieving "similar" performance rather than (or in addition to) a high average performance. The focus in such discussions is not just how good the performance efficiency of an application is (although it doesn't necessarily exclude that), but whether the performance efficiency it achieves is *consistent* across platforms.

Table III compares several common "consistency" measures computed for our synthetic data set: the standard deviation (SD), median absolute deviation (MAD) and the range of the data. We also include two variants of the standard deviation calculated from the harmonic mean (Har. SD), as proposed in [18]: the first is calculated as in [18], [19], and the second as in the original paper by Lam et al. [20]. Exactly how these values should be interpreted is unclear, but we include them in our comparison for completeness.

It is clear from the results that consistency alone is insufficient to meaningfully summarize the performance portability of applications. The two **consistent** data sets have the same degree of consistency according to all the measures tested, despite the fact that any reasonable user would prefer consistently high performance over consistently low performance.

But even pairing an average with a measure of consistency may obscure important and interesting information. Since each of the consistency metrics considered here represent how close the values are to a specific average, the degree to which they are affected by large or small performance efficiencies is dependent on the properties of the selected average. For example, the standard deviation and harmonic standard deviation paint very different pictures of consistency: for the **single target** data, the harmonic standard deviation is more reflective of consistently bad results; while for the **multi-target** data, the standard deviation is more reflective of the large difference in platform support.

Based on previous case studies [2], [3], [21], [23], we conjecture that applications are more likely to demonstrate a preference for certain architecture types than to exhibit normally distributed performance efficiencies across platforms, complicating the interpretation of simple statistical measures further. If data sets like **single target** and **multi-target** are as common as we believe, then how far our data points lie from an average value is unlikely to be the question that performance portability analyses should be trying to answer.

C. Beyond Single Number Metrics

We maintain that defining performance portability in terms of single number metrics can be a useful way to set project goals and make high-level comparisons, but such binary qualification of performance portability does not show the whole picture. Without insight into *why* an application's performance portability is what it is, or *how* the underlying performance efficiencies really differ across the platforms of interest, a developer cannot take appropriate action to resolve the issue(s). The remainder of this paper explores visualizations as a means to summarize performance portability, with an aim to preserve as much information about the raw data as possible.

IV. Distribution Across Platforms

In seeking to improve an application's performance portability, a developer must ultimately gain an understanding of how performance efficiency results are distributed across platforms. Poor efficiency on a small number of platforms might be root-caused to a performance bug or some combination of architecture and tool immaturity. Large deviations in efficiency and/or bimodal results may suggest that specializing for different platforms will prove beneficial, while improving tightly clustered results may require much broader optimizations.

However, drawing such insight from raw performance efficiency data can be challenging, and grows increasingly more so with each additional data point. Ideally, we seek an effective means of visualizing distributions that: does not distort or obscure results; permits multiple applications to be plotted

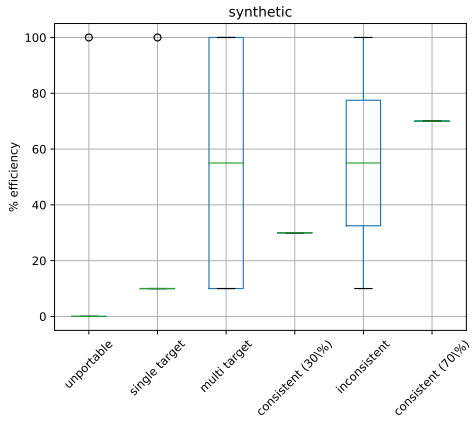


Fig. 2: Box plots of synthetic performance data.

together and compared; and is simple enough for developers to intuit the performance portability of an application at a glance. Such visualizations would enable developers to quickly (albeit subjectively) answer Question 1, Question 2 and Question 3.

A. Box Plots

Box plots are a common, well-understood figure showing the spread of data around the median, and are an obvious candidate for summarizing the distribution of performance efficiency data. The graph consists of a box formed by the lower and upper quartiles, which is divided by the median. Many software packages produce box plots with whiskers at 1.5 times the interquartile range from the box edge, and plot outliers beyond this range as circles.

Figure 2 shows box plots for the synthetic data introduced in Section II-C. This data is intended to stress test the approaches, and we can clearly see that the box plots fail to show useful information for many of these data sets. In the **multi-target** case, the fact that the data has two clusters is not represented in any way. The large box shows that the data is spread far from the median, but doesn't provide insight into the amount of data around these points. Likewise, the **inconsistent** data shows a fairly large box and a similar median value to the **multi-target** data, as we found in Table II. The data is evenly spread throughout the entire range, but this is not represented. The **consistent** data sets do not utilize the visible space on the graph well, but the lack of visible boxes conveys that the data is highly clustered around the median. Additionally, the difference in absolute performance between the two consistent data sets (30% and 70%) is clearly represented. For the **unportable** and **single target** data sets, the lack of boxes reflects the clustering around the low performance efficiency values. The single platforms with high performance efficiency are represented as outliers, reflecting that these results are not characteristic of this application – however, it is important to note that the decision to label these results as outliers is under user control, and therefore subject to abuse.

Figure 3 shows box plots for the real-world applications described in Section II-C. Each chart in the figure pertains to one code, with different box plots for each application (programming model). The first two box plots for BabelStream show clearly that much of the efficiency data is consistently high; however, it is easy to miss that some platforms did not run (represented by the outliers at zero), and the number of unsupported platforms is obscured (by nature of all outliers being at the same point). The other plots for this code do not yield much information as to the quality of performance portability; the boxes all cover the complete range $[0, 100]$, and we are left only with the median to make comparisons. Many of the box plots shown draw the median line at zero: most of the efficiency results are classified as not portable (*i.e.* most applications did not run on most platforms). It is difficult to see results where the data is non-zero.

When performance efficiencies are clustered around the median, box plots intuitively represent the extent of that clustering. However, in more general cases it can be challenging to understand the number and effect of outliers. In particular, bimodal distributions (like **multi-target**) appear severely distorted and indistinguishable from other distributions. Box plots therefore suffer from many of the same problems as the metrics discussed in Section III, and do not provide a clear way to intuit a ranking of applications.

B. Histograms

Another classic way to visualize the distribution of data is to produce a histogram. Data are grouped into categories (bins) and plotted as a bar chart showing the number of items in each bin, highlighting which bins are highly populated. A histogram also shows all the data directly (albeit smoothed into categories), preserving outliers and intermediate values occurring between regions of high density.

In selecting the bins, it is important to remember the meaning that we have ascribed to 0% performance efficiency (*i.e.* that an application did not run or produced an incorrect result). This is distinct from $(0 + \epsilon)\%$, which indicates that an application ran correctly, but with very low efficiency. As such, we recommend separating "did not run" results into their own bin, so as to distinguish them from low efficiencies. This is a special case of a common problem in constructing histograms: using too few bins hides useful information; but using too many bins does nothing to summarize the data. The significance of being in one bin or another is also open to interpretation: one might feel that efficiencies of 69% and 71% are equivalent, yet these results may fall into distinct bins.

Histograms for the synthetic data are shown in Figure 4a. We show all the data sets on the same graph for brevity also to allow direct comparison between them. Given the limited range of the data, it is important to plot the different data sets as independent bars side-by-side on the chart; in practice, overlaying them almost always obscures data points.

These histograms capture the characteristics of the data sets effectively. The two **consistent** data sets show strong peaks in the bins corresponding to 30% and 70% efficiency, and the two peaks of the **multi-target** data set are similarly intuitive. The presence of many low frequency bins for the **inconsistent** data

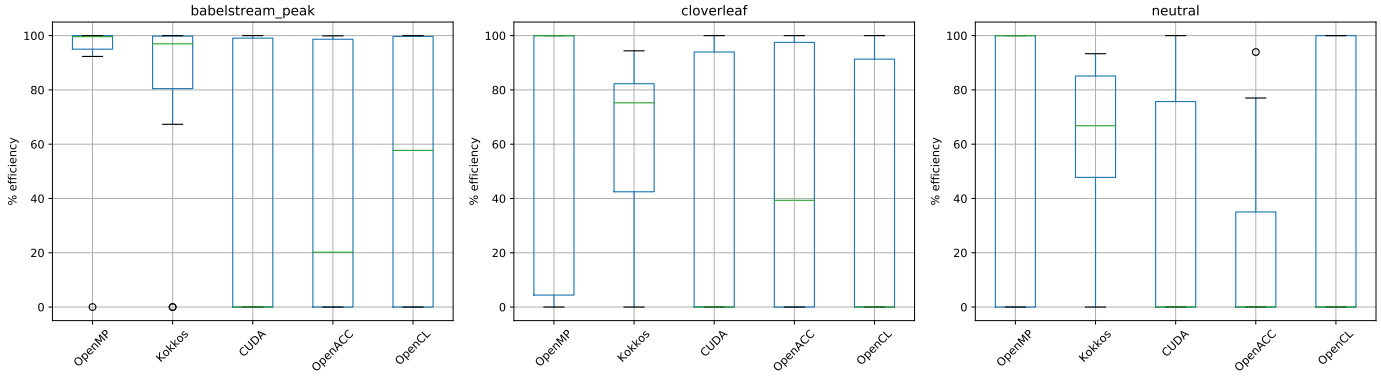
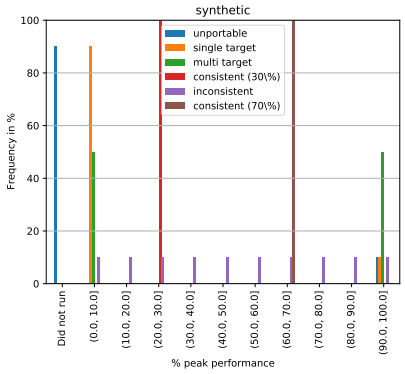
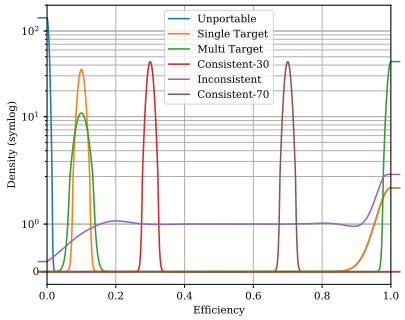


Fig. 3: Box plots of application efficiency for the real-life data sets.



(a) Histogram.



(b) Estimated probability density function.

Fig. 4: Comparative visualizations of the synthetic data set.

set reflects the wide spread of data. An approximate ordering of applications by performance portability can be derived by examining whether the largest peaks occur for low or high efficiency (reading left to right on the figure); however, such an approach highlights the challenge of ranking the **inconsistent** data set, which contains no peaks.

The corresponding histograms for the real application data are shown in Figure 5a. These figures are again plotted using frequency to normalize against differences in the number of platforms in the data set. Although the distribution of performance efficiencies for specific applications remains fairly clear, it is harder in the real data sets to compare different distributions. This is particularly true in areas of relatively low density (approximately $(0 - 60]$ for all applications shown), where a given bin is not guaranteed to have non-zero frequency for all applications; a reader seeking to understand the underlying distributions must effectively average over multiple bins by eye in order to make a meaningful comparison.

C. Probability Density Function

Each performance efficiency result can be thought of as a *sample* from some unknown underlying distribution. We can use these samples to construct a continuous density function with desirable properties using kernel density estimation techniques. These methods are heavily used in statistics, having first been introduced in the mid-20th century by Rosenblatt [24] and Parzen [25].

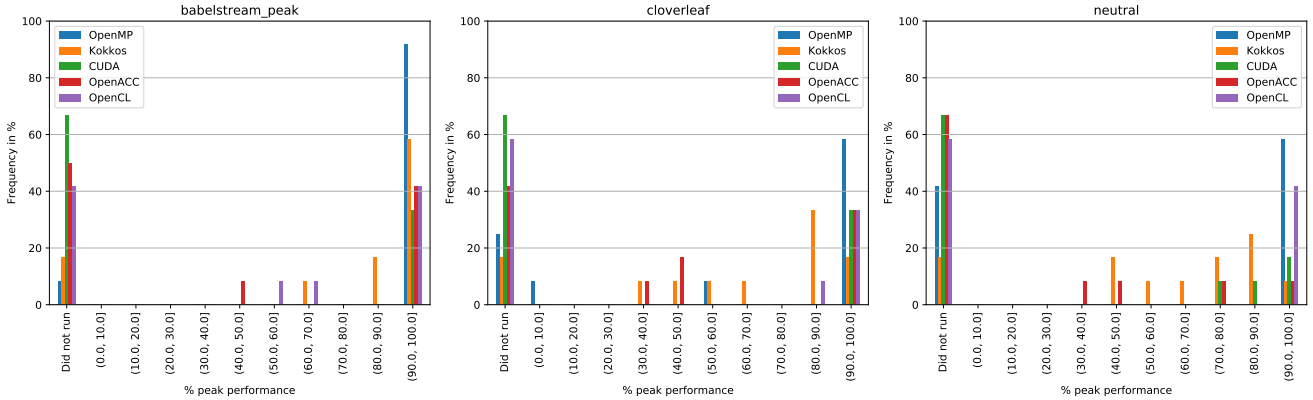
Most commonly, these methods additively apply some normalized continuous kernel function at sample points to reconstruct a continuous density estimation of the data set. The primary challenge is the choice of kernel bandwidth to apply in the process; a fixed width is often used based on a variety of heuristics. Most analysis of kernel density estimation methods is based on the asymptotic mean squared error, which considers the behavior of the estimators as the number of samples approaches infinity. While this is suitable for many applications, we must consider that performance portability studies generally have small, sparse sample sizes.

For data sets with sparse and unevenly distributed data, no fixed bandwidth is guaranteed to yield satisfactory results – isolated peaks may be smoothed out excessively, or closely placed samples may show as high frequency spikes. Work has been done on so-called "adaptive" or "variable" bandwidth kernel density estimation, where a different bandwidth is applied across the reconstruction space. The are many variations of this technique, which are reviewed in [26] and [27].

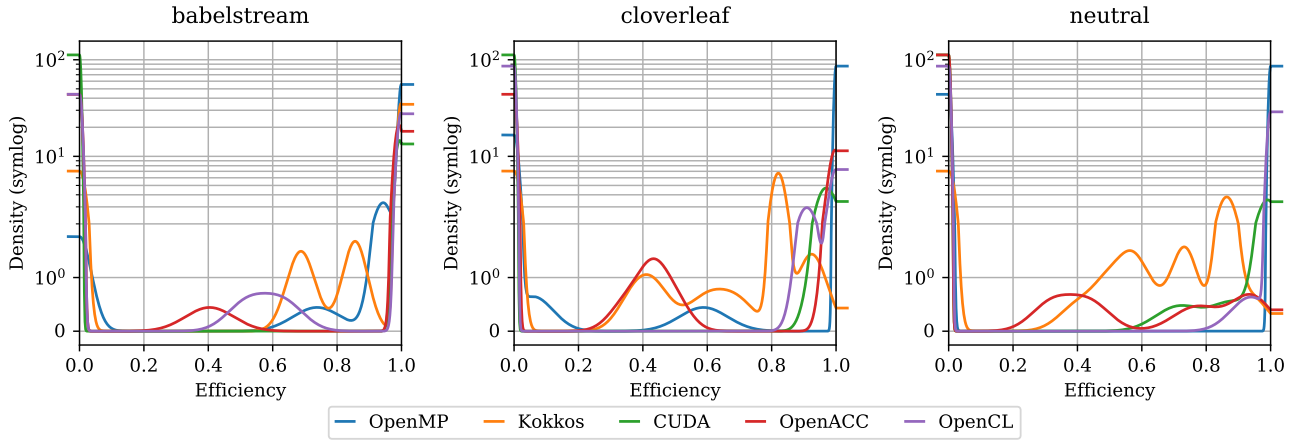
To construct a probability distribution function $f(x)$ from n efficiency samples $S = \{s_0, s_1, \ldots, s_{n-1}\}$, we use unit Gaussian kernels applied at sample locations:

$$f(x) = \frac{1}{n} \sum_{i \in [0,n)} \frac{1}{h(s_i)} K\left(\frac{x - s_i}{h(s_i)}\right) \tag{1a}$$

where $K(x)$ is the unit Gaussian and $h(x)$ is the density-to-bandwidth formula of Abramson [28] that operates on



(a) Histogram.



(b) Estimated probability density function.

Fig. 5: Comparative visualizations of the application data set.

estimated density $d(x)$ using a tuneable scaling factor V:

$$K(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2} \quad \text{(1b)} \qquad h(x) = \frac{V}{\sqrt{d(x)}} \quad \text{(1c)}$$

Estimating Density The formulae above are incomplete: we need to estimate the density $d(x)$ based on s_i. Of course, a continuous representative density is precisely what we are trying to compute with f; this suggests we should use an iterative refinement algorithm. We construct an initial fixed-bandwidth estimate $f_0(x)$ using Equation (1a), with $h(x) = C$. C may be chosen from among the popular heuristics (*e.g.* Silverman's Rule of Thumb [29]). Then we iteratively compute $f_i(x)$, using Equation (1c) with $d(x) = f_{i-1}(x)$. We continue until $\int_0^1 (f_i(x) - f_{i-1}(x))^2 \, dx$ has decreased below some threshold ϵ, at which point we may be satisfied of convergence. Figure 6 shows this process for an artificial data set.

Conserving Density There is one issue with the process described with Equation (1a) that must be addressed: the Gaussian kernel in Equation (1b) has infinite support, so each sample will contribute some volume to the entire real line. This is at odds with our knowledge that only efficiencies in $[0, 1]$ are meaningful – if we limit our domain of interest to this interval, we will lose density by "clipping" contributions.

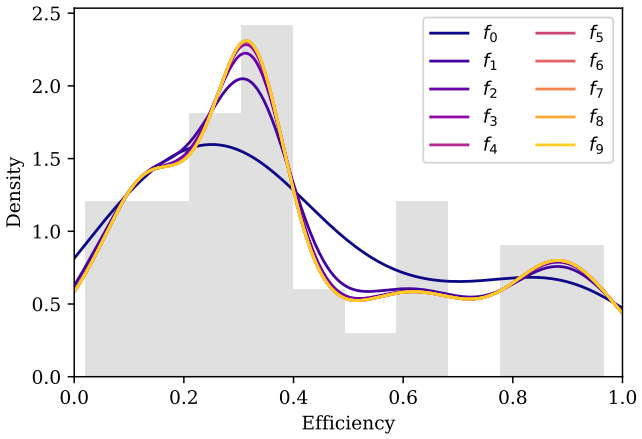


Fig. 6: Example of adaptive kernel density estimation refinement on an artificial data set. The gray bars are a density-normalized histogram of the samples. The line plots show iterates $f_0(x)$ through $f_9(x)$; note the progressive convergence.

To correct this, we propose a modification to Equation (1a)

to adjust the applied kernels $K(x)$ such that they contribute unit density to the interval of interest.

$$f(x) = \frac{1}{n} \sum_{i \in [0,n)} \frac{Y_a^b\left(s_i, h(s_i)\right)}{h(s_i)} K\left(\frac{x - s_i}{h(s_i)}\right) \tag{2}$$

The scaling factor $Y_a^b(s_i, h(s_i))$ adjusts the kernel applied at s_i with bandwidth $h(s_i)$ to integrate to unity over $[a, b]$. This can be derived as follows (abbreviating $Y_a^b(s_i, h(s_i))$ for clarity):

$$Y_a^b \int_a^b \frac{1}{h(s_i)} K\left(\frac{x - s_i}{h(s_i)}\right) dx = 1 \tag{3a}$$

$$Y_a^b \int_{a'}^{b'} K(t) dt = 1 \tag{3b}$$

$$Y_a^b \left(\int_{-\infty}^{b'} K(t) dt - \int_{-\infty}^{a'} K(t) dt \right)^{-1} = 1 \tag{3c}$$

Where we have employed a change of variable $t = \frac{x - s_i}{h(s_i)}$. $a' = \frac{a - s_i}{h(s_i)}$ and $b' = \frac{b - s_i}{h(s_i)}$ and the knowledge that the improper integrals of K do not diverge. Now, when K is the standard normal distribution, we know that the integrals found in Equation (3c) are given by the cumulative distribution function for the standard normal distribution:

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{1}{2}t^2} dt = \frac{1}{2}\left(1 + \operatorname{erf} \frac{x}{\sqrt{2}}\right) \tag{3d}$$

We combine Equations (3c) and (3d) to obtain:

$$\begin{aligned} Y_a^b &= 2\left(\operatorname{erf} \frac{b'}{\sqrt{2}} - \operatorname{erf} \frac{a'}{\sqrt{2}}\right)^{-1} \\ &= 2\left(\operatorname{erf} \frac{b - s_i}{\sqrt{2} h(s_i)} - \operatorname{erf} \frac{a - s_i}{\sqrt{2} h(s_i)}\right)^{-1} \end{aligned} \tag{3e}$$

Since $b > a$ and erf monotonically increases from -1 to 1, $Y_a^b > 1$. If samples lie within an interval of interest – which is the only scenario that is realistic for our purposes – the scaling effect is largest for samples on the boundary of the domain.

Applying Estimated Efficiency PDFs Figure 4b shows these techniques applied to the synthetic data; contrast with the histogram in Figure 4a. The aforementioned boundary effects are prominent, and compounded by the tendency of data sets to have many points at 0 and 1. High frequencies at these locations also encourage narrow bandwidths to be applied, resulting in large spikes. These are best plotted on a semilog scale that treats the densities $[0, 1]$ linearly and applies logarithmic scales to $(1, \infty)$. We have added small handles that visually extend the PDFs beyond $[0, 1]$ in x to make it clear where they intercept the efficiency = 0 and efficiency = 1 lines. We can derive the ordering of the synthetic data sets using the same approach used for the histograms in Section IV-B.

Figure 5b shows the techniques applied to the real-world data sets; again a semilog scale for density has been used, along with handles to show intersections with the boundaries of efficiency. The tendency of efficiency data to cluster around the endpoints is pronounced here.

The density plots provide similar insight into an application's performance portability as the histograms in Section IV-B. However, the density plots are an improvement in two areas: extracting trends is easier; and multiple applications may be plotted without resulting in crowded clusters of bars.

Violin Plots Violin plots can be used to visualize the same curves as those produced from kernel density estimation. For this data they are the same curves as seen in Figure 5b, reflected into a symmetrical shape and presented as the box plots. As they display the same data in much the same way, we do not expand on them further in this paper. Their main use would be in comparing two similar data sets side by side, however would be limited to comparing two data sets, unlike those in Figure 5b which is clear for many data sets.

V. Impact of Platform Selection

A critical part of discussing performance portability is the set of platforms that an application is expected to support. The approaches in Section III and Section IV treat all platforms equally, but it is highly likely that different subsets of platforms will have higher or lower priorities during an application's lifetime: for example, purchasing a new machine may increase a platform's priority temporarily, while an existing machine being retired may permanently reduce interest in maintaining support for related (historical) platforms.

Whatever the reason, selecting a different platform set can have a significant impact upon performance portability analyses. For example, single number metrics such as P evaluate to zero in the presence of any single "did not run" result, and the distribution of performance efficiencies may be distorted if a platform set includes many similar platforms [2]. In this section, we explore an alternative view of performance portability, by characterizing an application's sensitivity to changes in platform set. Understanding this sensitivity may go some way towards answering Question 4.

A. Plotting P Against H

The study by Deakin et al. [17] contains plots of performance portability (as measured by P) against the platform set (H) used to compute it. In these plots, the x-axis for all applications is sorted to represent decreasing subsets of platforms: the leftmost point includes all platforms, and each subsequent point removes the least supported platform. Applications with better performance portability appear closer to the top-left of the graph, achieving higher P scores for larger platform sets.

The study defined the least supported platform as the one with the fewest non-zero performance efficiencies across all applications (programming models) being compared. As a result, the approach cannot be directly applied to studies of a single application (where the number of non-zero performance efficiencies for each platform will always be either 0 or 1). We refine that approach here, by considering alternative constructions of the platform domain (the x-axis).

B. Efficiency Cascade Plots

Consider an application with efficiencies E_0 observed for a set of platforms H_0. One or more platforms has the

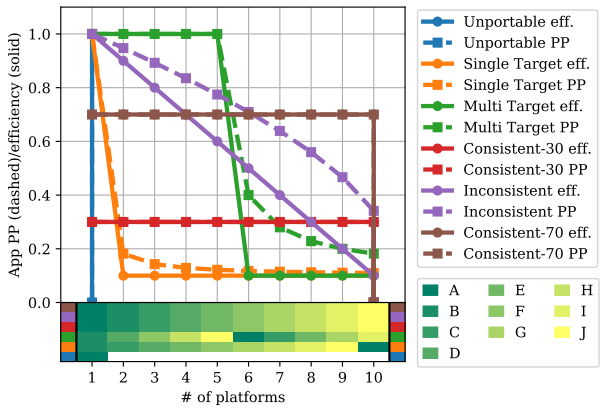


Fig. 7: Efficiency cascade plot for synthetic data sets, along with platform chart.

minimum efficiency in E_0; we record this $\min E_0$ and the cardinality of H_0, then remove any one platform among those with the minimum efficiency to construct a new set of platforms H_1 with corresponding efficiencies E_1. We continue for $n = |H_0|$ steps in this fashion until we obtain $H_n = \emptyset$. We then plot $|H_{n-1}|, |H_{n-2}|, \ldots, |H_0|$ against $\min E_{n-1}, \min E_{n-2}, \ldots, \min H_0$ (*i.e.* increasing number of platforms vs. minimum efficiencies among each subset) to obtain a visualization of how precipitously an application's support for various platforms degrades.

This is necessarily non-increasing, and so we designate these plots as *efficiency cascade plots*. It is trivial to compute $\mathcal{P}$ using E_i for each $|H_i|$ and to superimpose this with the efficiency cascade. Multiple applications/problems may be aggregated onto a single efficiency cascade plot by winnowing the platform sets as described above *individually*; the H_i at each tick on the x-axis will not necessarily be the same across applications, but the plotted cardinalities are shared. Figure 7 demonstrates such a plot for the synthetic data sets, with solid and dashed lines representing the minimum efficiency and $\mathcal{P}$ values respectively.

Efficiency cascade plots may be easily constructed by individually sorting each application's efficiencies across all platforms in decreasing order, then plotting them piecewise-linear against the sequence number of the platform in this ordering. Reading a platform number from the x-axis and consulting the plotted efficiency (for a specific application) gives the number of platforms with at least that level of efficiency. Conversely, reading right from an efficiency or $\mathcal{P}$ value on the y-axis to where it intercepts a plotted value gives the number of platforms that have an efficiency or $\mathcal{P}$ greater than the chosen y value.

Because the platform sets for each application are sorted separately, we must resist the temptation to draw any conclusions about specific comparative platform performance across applications in efficiency cascade plots. The exception is for the rightmost point for each application, which shows the minimum efficiency and $\mathcal{P}$ calculated across all platforms.

Platform Charts It is possible to include information about the H_i chosen by adding a row of color-coded boxes for each application for each $|H_i|$ plotted on an efficiency cascade; this we term a *platform chart*. Where an application does not run on a platform, the space is left blank. Due to the construction of efficiency cascades, we can always expect the blank areas for any given application to be contiguous and on the right-hand side of the plot.

When application efficiency is used and the data set is "closed" (*i.e.* the peak efficiency for each platform is contained in the shown data, as in the data used here) we expect the efficiency cascade to begin with a series of peak efficiency values. Similarly, we can expect that the leftmost appearance(s) of a platform in a platform chart contains the peak efficiency for that platform among the data.

By nature of the harmonic mean, the $\mathcal{P}$ for an application on a given set of platforms is never lower than the minimum efficiency. For the **consistent** data sets, the $\mathcal{P}$ and efficiency as seen in Figure 7 are identical. The bimodal nature of the **single target** and **multi-target** data sets are also reflected in the efficiency cascade, with clear transitions between two levels of support marked by sharp drops in efficiency.

Figure 8 shows efficiency cascade plots for the real-world data. There are numerous distinct patterns that help to quickly assess application (*i.e.* language/framework) behavior. The number of supported platforms is marked by a drop to zero efficiency. There are some applications that show high efficiency for a subset of platforms after which efficiency precipitously drops, reminiscent of the **single target** data set.

Other observations are notable for requiring subjective evaluation. In all data sets, OpenMP leads or ties all other applications through most of the platforms. In some cases, Kokkos supports more platforms, or supports latter platforms with higher efficiency than OpenMP. For additional discourse on this subject, we refer the reader to the study of Deakin et al. [17]. It is an interesting exercise for the developer or application user to consider whether they prefer performance or portability; in some cases, it is most important that as many platforms be supported as possible; while in other cases, the higher efficiency may be more desirable.

By choosing colors in the platform chart that convey meaningful groupings, we can gain insight into how individual applications handle different types of platforms. In Figure 8, we have chosen distinct color families for GPUs and CPUs, highlighting that CUDA supports only GPUs, while both OpenCL and OpenACC have weak support for CPUs in the data presented.

It is reasonable to wonder how the data presented in an efficiency cascade plot coincides (or does not) with one's idea of performance portability, quantitative or qualitative. Since $\mathcal{P}$ is featured in the plots, these questions are simple to answer: the highest point in the rightmost column is the application with the highest $\mathcal{P}$ across all platforms in H_0. Comparing points to the left can be misleading, since the H_i at each of these points is not necessarily the same across applications.

Insights into the qualitative question are readily available in

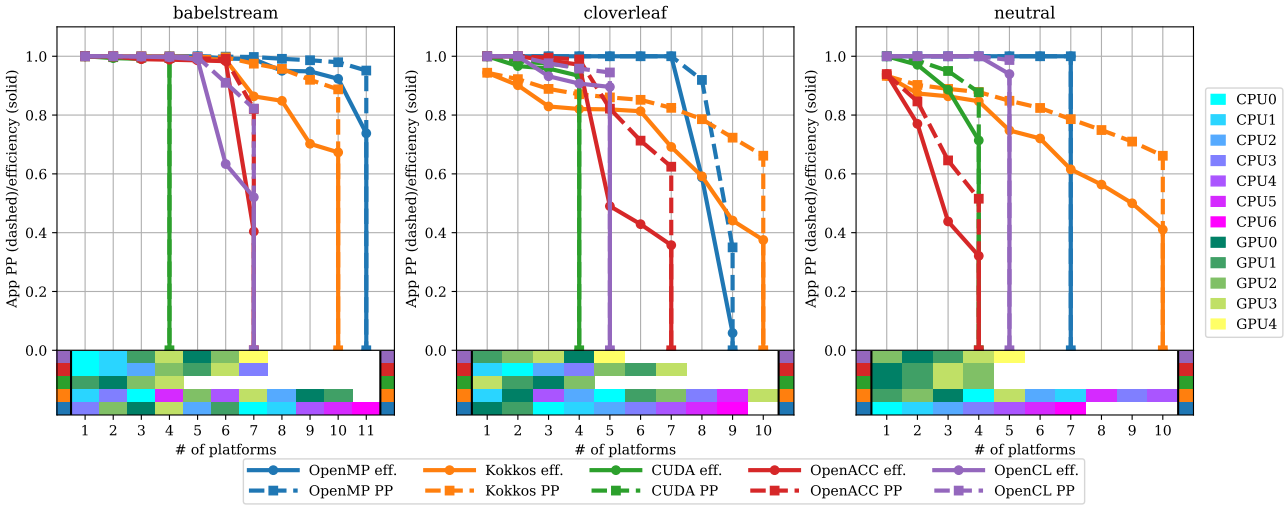


Fig. 8: Efficiency cascade plots for the real-life data sets, along with platform charts.

efficiency cascades. Imagine a line of the form $y = -\alpha x + c$ sweeping from the top-right of the plot (*i.e.* by decreasing c); the first application efficiency plot that is intersected can reasonably be argued to be the most performance portable. For $\alpha \gg 1$, the line becomes steeper – the first intersected point will be the application with the highest $\mathcal{P}$ across the most platforms, as described above. For smaller α (> 0) this sweeping line may intersect an application that supports fewer platforms but does so at a higher net efficiency than the application that maximizes absolute $\mathcal{P}$ over all platforms. This process may be used progressively to sort applications by their performance portability.

For example, in Figure 7, **consistent (70%)** has the highest $\mathcal{P}$ across all platforms. If a developer values performance on a subset of platforms more than what $\mathcal{P}$ expresses over a larger set, **multi-target** may satisfy. Likewise, in Figure 8, Kokkos has the highest $\mathcal{P}$ for Cloverleaf and Neutral, but valuing the highest maximum for these applications may result in OpenMP appearing the most performance portable. These valuations can quickly become subjective, but remain useful in guiding discussions.

C. Assigning Platform Weights

The cascade plots in Section V-B order platforms along the x-axis differently for each application, implicitly assigning an equal weight to each platform. This is similar to the distribution analyses of Section IV, where the platform dimension is also not exposed explicitly. This formulation of the cascade plots therefore assumes that a developer's ultimate goal is to support all platforms with high performance efficiency, but as discussed previously, there are many circumstances in which a developer may assign different priorities to supporting different platforms.

One could imagine sorting the x-axis for all applications to align with a developer's priorities; however, the data points for each application would no longer be guaranteed to be non-increasing, and interpretation of the plots would follow [17]. A more promising approach would be to assign colors in the platform chart beneath the plot based on priority; this is a straightforward extension to the existing plots, and one that we intend to explore in future work.

VI. Conclusion

Understanding performance portability data is a challenging and multi-faceted problem. To this end, we have identified questions which a developer might ask of their applications in their quest for performance portability.

This paper has demonstrated that looking at "average" performance efficiency or the "consistency" of performance efficiency alone can be misleading, and doesn't provide actionable insight into an application's performance portability.

Visualizing the distribution of performance efficiency across the platform set of interest has proven much more intuitive in this regard. Specifically, we have identified two promising directions for novel visualizations that effectively summarize the performance portability of applications:

1) Estimated probability density functions (see Section IV-C), which summarize the data distribution in a familiar and histogram-like fashion; and
2) Efficiency cascade plots (see Section V-B), which highlight an application's sensitivity to platform selection and provide insight into how individual efficiency values impact the calculation of $\mathcal{P}$.

Although not the original intention of this work, we find that $\mathcal{P}$ remains a useful summary metric that is aligned with our intuition of what it means for an application to be performance portable. We believe that augmenting $\mathcal{P}$ values with one or more of the visualizations discussed in this paper effectively overcomes the metric's shortcomings, and provides the insight that developers have been missing.

All scripts used to produce the plots in this paper are open source, available online (see Appendix A), and straightforward to apply to new data sets. Work is ongoing to combine the scripts into a more comprehensive software package.

Acknowledgments

This work was in part funded by the EPSRC ASiMoV project (EP/S005072/1).

Disclaimers

If accepted, the camera-ready version of this paper will include disclaimers.

Appendix A
Artifact Description Appendix: Interpreting and Visualizing Performance Portability Metrics

A. Abstract

This paper contains no computational results. The plotting scripts used to produce the visualizations in this paper, along with the synthetic and real-world data sets, are available online under the MIT License at https://github.com/UoB-HPC/performance-portability.

References

[1] S. J. Pennycook, J. D. Sewall, and V. W. Lee, "A Metric for Performance Portability," *CoRR*, vol. abs/1611.07409, 2016. [Online]. Available: http://arxiv.org/abs/1611.07409 1, 2, 3

[2] ——, "Implications of a Metric for Performance Portability," *Future Generation Computer Systems*, aug 2017. [Online]. Available: https://doi.org/10.1016/j.future.2017.08.007 1, 2, 3, 4, 8

[3] S. J. Pennycook, J. D. Sewall, and J. R. Hammond, "Evaluating the Impact of Proposed OpenMP 5.0 Features on Performance, Portability and Productivity," in *2018 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*, 2018, pp. 37–46. 1, 4

[4] C. Yang, R. Gayatri, T. Kurth, P. Basu, Z. Ronaghi, A. Adetokunbo, B. Friesen, B. Cook, D. Doerfler, L. Oliker, J. Deslippe, and S. Williams, "An Empirical Roofline Methodology for Quantitatively Assessing Performance Portability," in *2018 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*, 2018, pp. 14–23. 1

[5] A. Hsu, D. N. Asanza, J. A. Schoonover, Z. Jibben, N. N. Carlson, and R. Robey, "Performance Portability Challenges for Fortran Applications," in *2018 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*, 2018, pp. 47–58. 1

[6] T. Zhao, S. Williams, M. Hall, and H. Johansen, "Delivering Performance-Portable Stencil Computations on CPUs and GPUs Using Bricks," in *2018 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*, 2018, pp. 59–70. 1

[7] I. Z. Reguly, "Performance Portability of Multi-Material Kernels," in *2019 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*, 2019, pp. 26–35. 1

[8] 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)*, 2018, pp. 24–36. 1, 2

[9] B. Siklosi, I. Z. Reguly, and G. R. Mudalige, "Heterogeneous CPU-GPU Execution of Stencil Applications," in *2018 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*, 2018, pp. 71–80. 1

[10] D. F. Daniel and J. Panetta, "On Applying Performance Portability Metrics," in *2019 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*, 2019, pp. 50–59. 1, 2

[11] A. Sedova, J. D. Eblen, R. Budiardja, A. Tharrington, and J. C. Smith, "High-Performance Molecular Dynamics Simulation for Biological and Materials Sciences: Challenges of Performance Portability," in *2018 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*, 2018, pp. 1–13. 1, 2

[12] H. Dreuning, R. Heirman, and A. L. Varbanescu, "A Beginner's Guide to Estimating and Improving Performance Portability," in *High Performance Computing*, R. Yokota, M. Weiland, J. Shalf, and S. Alam, Eds. Cham: Springer International Publishing, 2018, pp. 724–742. 1, 2

[13] Khronos OpenCL Working Group, *The OpenCL Specification*, 2nd ed., March 2016. 2

[14] Khronos SYCL Working Group, *SYCL Specification*, 1st ed., April 2020. 2

[15] H. C. Edwards, C. R. Trott, and D. Sunderland, "Kokkos: Enabling Manycore Performance Portability Through Polymorphic Memory Access Patterns," *Journal of Parallel and Distributed Computing*, vol. 74, no. 12, pp. 3202 – 3216, 2014, domain-Specific Languages and High-Level Frameworks for High-Performance Computing. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0743731514001257 2

[16] R. D. Hornung and J. A. Keasler, "The RAJA Portability Layer: Overview and Status," Lawrence Livermore National Laboratory, Tech. Rep. LLNL-TR-661403, September 2014. 2

[17] T. Deakin, S. McIntosh-Smith, J. Price, A. Poenaru, P. Atkinson, C. Popa, and J. Salmon, "Performance Portability across Diverse Computer Architectures," in *2019 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*, 2019, pp. 1–13. 2, 3, 4, 8, 9, 10

[18] C. Bertoni, J. Kwack, T. Applencourt, Y. Ghadar, B. Homerding, C. Knight, B. Videau, H. Zheng, V. Morozov, and S. Parker, "Performance Portability Evaluation of OpenCL Benchmarks across Intel and NVIDIA Platforms," in *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2020, pp. 330–339. 4

[19] M. Martinez and M. Bartholomew, "What Does It "Mean"? A Review of Interpreting and Calculating Different Types of Means and Standard Deviations," *Pharmaceutics*, vol. 9, no. 4, p. 14, Apr 2017. [Online]. Available: http://dx.doi.org/10.3390/pharmaceutics9020014 4

[20] F. Lam, C. Hung, and D. Perrier, "Estimation of Variance for Harmonic Mean Half-Lives," *Journal of Pharmaceutical Sciences*, vol. 74, no. 2, pp. 229 – 231, 1985. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0022354915465803 4

[21] S. McIntosh-Smith, M. Boulton, D. Curran, and J. Price, "On the Performance Portability of Structured Grid Codes on Many-Core Computer Architectures," in *Supercomputing: 29th International Conference, ISC 2014, Leipzig, Germany, June 22-26, 2014. Proceedings*, J. M. Kunkel, T. Ludwig, and H. W. Meuer, Eds. Cham: Springer International Publishing, 2014, pp. 53–75. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-07518-1_4 4

[22] DOE Office of Science. (2017, August) Definition - Performance Portability. [Online]. Available: https://performanceportability.org/perfport/definition/ 4

[23] A. Sabne, P. Sakdhnagool, S. Lee, and J. S. Vetter, "Evaluating Performance Portability of OpenACC," in *Languages and Compilers for Parallel Computing: 27th International Workshop, LCPC 2014, Hillsboro, OR, USA, September 15-17, 2014, Revised Selected Papers*, J. Brodman and P. Tu, Eds. Cham: Springer International Publishing, 2015, pp. 51–66. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-17473-0_4 4

[24] M. Rosenblatt, "Remarks on Some Nonparametric Estimates of a Density Function," *The Annals of Mathematical Statistics*, vol. 27, no. 3, pp. 832–837, 1956. 6

[25] E. Parzen, "On Estimation of a Probability Density Function and Mode," *The Annals of Mathematical Statistics*, vol. 33, no. 3, pp. 1065–1076, 1962. 6

[26] W. Jarosz, "Efficient Monte Carlo Methods for Light Transport in Scattering Media," Ph.D. dissertation, UC San Diego, September 2008, Appendix C: Density Estimation. [Online]. Available: https://cs.dartmouth.edu/~wjarosz/publications/dissertation/ 6

[27] G. R. Terrell and D. W. Scott, "Variable Kernel Density Estimation," *The Annals of Statistics*, pp. 1236–1265, 1992. 6

[28] I. S. Abramson, "On Bandwidth Variation in Kernel Estimates - A Square Root Law," *The Annals of Statistics*, pp. 1217–1223, 1982. 6

[29] B. W. Silverman, *Density Estimation for Statistics and Data Analysis*. CRC press, 1986, vol. 26. 7