A Metric for Performance Portability

S. J. Pennycook, J. D. Sewall and V. W. Lee
Intel Corporation
Santa Clara, California
{john.pennycook,jason.sewall,victor.w.lee}@intel.com

Abstract—The term "performance portability" has been informally used in computing to refer to a variety of notions which generally include: 1) the ability to run one application across multiple hardware platforms; and 2) achieving some notional level of performance on these platforms. However, there has been a noticeable lack of consensus on the precise meaning of the term, and authors' conclusions regarding their success (or failure) to achieve performance portability have thus been subjective. Comparing one approach to performance portability with another has generally been marked with vague claims and verbose, qualitative explanation of the comparison. This paper presents a concise definition for performance portability, along with a simple metric that accurately captures the performance and portability of an application across different platforms. The utility of this metric is then demonstrated with a retroactive application to previous work.

I. INTRODUCTION

Developing and maintaining a separate implementation of an application for each new hardware platform of interest is a huge undertaking, and one that is unrealistic for the developers of the large and long-lived applications found in high performance computing (HPC). It is therefore unsurprising that the introduction of programming models, languages and tools that provide developers the ability to write single-source codes that can be compiled for different targets (*e.g.* OpenCL[1] [13], OpenMP[1] [21], OpenACC[1] [20]) has been met with enthusiasm from the HPC community.

However, the growth of options for ensuring *functional portability* across diverse hardware platforms naturally leads one to wonder if any of them can deliver portability without sacrificing the level of performance that developers have come to expect from traditional HPC languages (such as C, C++ and Fortran). The term *performance portability* has appeared in the literature, but the community has not agreed upon the meaning of the term nor how to measure the degree to which an application (or library, framework, programming model, etc) has become performance portable [1].

A shared lexicon and shared goals are necessary first steps in bringing performance portability researchers together to discuss and compare their findings, and to that end this paper makes the following contributions:

1) We propose a new definition of performance portability, and demonstrate how it avoids the shortcomings of previous definitions proposed in the literature;
2) We describe a novel metric for characterizing performance portability (according to our proposed definition)

and demonstrate its accuracy and utility for quantifying an application's performance *and* portability; and
3) We retroactively apply our metric to a number of published application studies, thereby highlighting the utility of a shared metric when comparing and contrasting different approaches to performance portability.

II. RELATED WORK

There have been a number of efforts to develop new programming models, languages and tools that provide users with a productive means of achieving performance portability. Some have proposed the use of domain-specific languages (DSLs), providing a limited set of high-level abstractions for a specific domain alongside a mechanism for generating optimized platform-specific binaries (*e.g.* Liszt [7], OP2 [19], [25], [26], STELLA [10], Pochoir [31], PATUS [5], Halide [24] and Nabla [4]). Others have proposed portability frameworks that take a different approach, providing a lower-level abstraction of performance-enabling hardware for developers to program against: some of these tools are truly languages or language extensions (*e.g.* OpenCL [13], OpenMP 4.0 [21], OpenACC [20], Petabricks [2] and Sequoia [9]) while others are provided as libraries (*e.g.* Kokkos [8] and RAJA [12]).

The performance portability claims of these languages and frameworks have subsequently been tested in several individual application case-studies [6], [11], [14], [15], [18], [22], [23], [26]–[28], [33]. The amount of expended development effort focused on platform-specific optimization and the level of performance that an application must achieve in order to be considered performance portable is inconsistent across studies; we explore these inconsistencies and examine the results of some of these studies in more detail later in the paper.

III. DEFINING PERFORMANCE PORTABILITY

As discussed, there is far from consensus on the meaning of *performance portability*, and one goal of this paper is to establish a significant and useful definition for the term. To begin with, we establish some basic terms: a *platform* is a particular execution environment (*i.e.* hardware, an operating system, some compilation and runtime tools); and an *application* is any suite of software that can accept a given *problem* as input and produce an output that can be validated against some existing measure of correctness (*i.e.* two completely separate source codes that solve the same problem to satisfiable accuracy are the same application).

[1]Other names and brands may be claimed as the property of others.

These are each inclusive concepts that primarily delineate components needed for further discussion. We also establish the following definitions for performance and portability:

Performance
Any measurable property of an application's correct execution of a problem on a platform.

This is a broad definition; the most common performance metrics are based on economy of time (*e.g.* time-to-solution or floating-point operations/second) or economy of energy (*e.g.* energy-to-solution or floating-point operations/watt). Also note that it is rare to be able to make a meaningful comparison between performance measurements based on distinct properties.

Portability
The ability of an application to execute a problem correctly on a given set of platforms.

An application and problem combination is either portable for a given set of platforms or not; it either runs correctly on all of them or fails for at least one of them. This is distinct from the portability of an application's source code.

A. Criteria for Useful Definitions of Performance Portability

There is a set of criteria that a useful definition of performance portability should satisfy; the definition should:

1) Reflect the individual meanings of the terms "performance" and "portability";
2) Be objective; and
3) Be measurable (and comparisons of the measured value should be meaningful).

Satisfying these criteria ensures: that usage of the term is in line with a reader's expectations (*i.e.* that performance portable applications are efficient on multiple platforms); that claims of performance portability are backed by facts and not subject to opinions of programming languages and platforms; that there is a standard way to evaluate the success of the community efforts to achieve performance portability; and that an end-user given a choice of two applications is able to reason about which is the most performance portable across the hardware platforms that are available to them.

Although *productivity* is one of the driving factors behind community interest in solutions to performance portability [1], we deliberately exclude it from our definition criteria because it is at odds with objectivity and measurability – an application developer's productivity heavily depends upon their skillset, and common productivity metrics (*e.g.* lines/words of code) do not reflect the reality that a library/framework may be more productive to use than it is to develop and/or maintain. Whether or not there exists a meaningful and useful definition for *productive* performance portability is beyond the scope of this work, but keeping productivity considerations independent of performance portability is compatible with existing efforts to measure the impact of different performance portability frameworks on programmer productivity [11], [16], [32].

B. Existing Definitions

In this section, we test a number of existing definitions for performance portability against our criteria. While we respect the intentions behind these definitions and agree with aspects of all of them, they each have shortcomings and would benefit from further refinement. This is by no means an exhaustive list of definitions – rather, we have selected a set of representative perspectives from both literature and informal discussions at conferences and translated them into definitions.

Definition 1
An approach to application development, in which developers focus on providing portability between platforms without sacrificing performance.

This definition serves as an important reminder: if a developer is working to support multiple platforms and to improve performance across the platforms they support, then they should be able to claim some level of performance portability. While we agree with this aspect of the definition, it is highly subjective and sets a very low bar.

Definition 2 [15]
The ability of the same source code to run productively on a variety of different architectures.

This definition implies that maintaining separate code paths for each platform is not productive, and should therefore be disallowed. However, the number of device-specific optimizations and other source code changes (*e.g.* additional pragmas) that are permitted before a code is no longer "the same" as it was originally is subjective.

Definition 3 [34]
$P_n^{\mathscr{P}}(b \to t) = \frac{S_n{}^t}{S_n{}^b} \times 100\%$ for program $\mathscr{P}$, base system b, target system t and speed-up on n nodes S_n.

This definition is clearly objective and measurable, but comparison of these measurements is complicated (and may not be meaningful) because the scaling behavior of two applications does not necessarily reflect their absolute performance.

Definition 4 [18]
The ability of an application to achieve a similar high fraction of peak performance across target devices.

"Peak performance" here is the theoretical maximum performance of a platform, and the fraction of this that an application can achieve is an objective and useful quantity that we refer to as *architectural efficiency*. This definition is measurable and comparable, but its use of the terms "similar" and "high" make it subjective. Furthermore, the aspiration that similar architectual efficiencies be achieved across platforms is often unrealistic and penalizes sets of platforms with microarchitectural differences (*e.g.* applications with an imbalanced ratio of multiplications and additions are more efficient on platforms without fused multiply-add instructions).

Definition 5 [8]
The ability of an application to obtain the same (or nearly the same) performance as a variant of the code that is written specifically for that device.

This definition meets all of our criteria except for objectivity and measurability: "nearly the same" is subjective, and different development teams are likely to have different opinions on how similar performance must be before they have achieved performance portability.

Definition 6
> The ability of an application to execute with a performance difference of less than $2\times$ on two different systems, without significant software changes.

This definition initially appears to satisfy all of our criteria, but it has counterintuitive properties that do not reflect absolute performance. For example, consider an application that executes on Platform A in four seconds and on Platform B in one second; a speed-up of $2\times$ on Platform A *or* a slow-down of $2\times$ on Platform B both result in a state that satisfies the definition. Like Definition 4, it also fails to account for architectural differences between platforms that may prevent an application from meeting the definition.

C. Our Definition

Our proposed definition of performance portability is a refinement of the definitions above:

Performance Portability
> A measurement of an application's performance efficiency for a given problem that can be executed correctly on all platforms in a given set.

An application that cannot execute a given problem correctly across a given set of platforms is not performance portable (for that problem across that set of platforms). We separate application and problem (*i.e.* input parameters) to acknowledge that an application's behavior and performance is largely dependent on the problem being solved, often to the point that a different algorithm may be more appropriate.

Our definition builds directly on our earlier definitions of performance and portability, and remains objective by specifying that performance efficiency (*i.e.* a ratio of observed performance relative to some proven, achievable level of performance such as a previous architectural study or a performance model) for each platform be used, rather than absolute performance. This precludes the use of terms like "good performance". It is not obvious from the definition alone that it satisfies the third of our criteria; the next section presents a new metric for performance portability to support our proposed definition.

IV. MEASURING PERFORMANCE PORTABILITY

As with our definition, we have designed our performance portability metric around specific criteria. A useful metric should:

1) Be measured specific to a set of platforms of interest H.
2) Be independent of the absolute performance across H.
3) Be zero if a platform in H is unsupported, and approach zero as the performance of platforms in H approach zero.
4) Increase if performance increases on any platform in H.
5) Be directly proportional to the sum of scores across H.

Satisfying these criteria ensures that the metric is easy to understand: that there are no assumptions about the number of or type of platforms; that an application cannot claim performance portability in situations that it couldn't claim portability; and that comparisons of performance portability reflect comparisons of performance.

Our proposed metric is the harmonic mean of an application's performance efficiency observed across a set of platforms. If the application fails on any measured platform(s), then we define the performance portability to be 0.

Formally, for a given set of platforms H, the performance portability $\mathcal{P}$ of an application a solving problem p is:

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

where $e_i(a,p)$ is the performance efficiency of application a solving problem p on platform i.

The harmonic mean has been previously demonstrated as a useful way to aggregate multiple performance numbers [30]. Unlike the geometric and arithmetic means, the harmonic mean satisfies criteria 3) and 5) above. It is also robust to large outliers, preventing applications from making $\mathcal{P}$ artificially large by adding more platforms; this ensures that, given measurements for two applications on the same set of platforms, it is clear which application is most likely to achieve the best performance on any selected platform.

There are multiple performance efficiency metrics that could be used to compute $\mathcal{P}$. In this paper, we consider two: 1) *architectural efficiency* (achieved performance as a fraction of peak theoretical hardware performance), which represents the ability of an application to utilize hardware efficiently; and 2) *application efficiency* (achieved performance as a fraction of best observed performance) [29], which represents the ability of an application to use the most appropriate implementation and algorithm for each platform. This distinction is important, because an application that hits 100% of peak performance (*e.g.* DRAM bandwidth or GFLOP/s) is not necessarily well-optimized (*e.g.* a bandwidth-bound application may be moving data between DRAM and cache unnecessarily, or a compute-bound application may be performing redundant computation), and naturally the best known performance of an application is not necessarily the best that is possible. In both cases, performance efficiency can be represented as the ratio of observed and ceiling (*i.e.* peak theoretical or best observed) performance arranged such that it lies in $[0,1]^2$; 1 means that the observed performance matches the best, and lower values show how much worse the observed is than the best.

A. Constructed Examples

Table I(a) contains some example performance data from an imaginary application running on five different imaginary platforms. The example has been constructed to represent the sorts of complications that may arise when comparing performance across different hardware platforms: the implementation used for Platform B trades communication for redundant computation, performing twice as many floating-point operations as the other platforms; Platforms A and C are microarchitecturally very similar but for the introduction of fused multiply-add instructions in Platform C; and Platform D is a new platform that the application doesn't yet support.

---

[2]Or an equivalent percentage in $[0,100]$

Platform	GFLOP/s		Arch Eff.	Time (s)		App. Eff.
	Achieved	Peak		Achieved	Best	
A	40	250	16%	25.0	25.0	100%
B	160	700	23%	12.5	10.0	80%
C	40	500	8%	25.0	12.5	50%
D	-	800	0%	-	6.0	0%
E	200	1250	16%	5.0	1	20%

(a) Application and architectural efficiencies.

Platform Set (H)	$\mathbf{\Phi}(a, p, H)$	
	Arch. Eff.	App. Eff.
{A, B, C, D, E}	0.00%	0.00%
{A, B, C, E}	13.62%	43.23%
{A, B, C}	12.97%	70.59%
{A, C}	10.67%	66.67%
{A}	16.00%	100.00%

(b) Performance portability.

TABLE I: A constructed example with five platforms.

Implementation	$\mathbf{\Phi}(a, p, H)$	
	Arch. Eff.	App. Eff.
McCalpin	73.39%	96.33%
SYCL	55.70%	68.79%
RAJA	64.66%	85.68%
Kokkos	64.69%	85.66%
OpenMP (C++)	61.01%	80.86%
OpenACC	50.02%	65.79%
CUDA	46.41%	64.21%
OpenCL	50.04%	65.27%

TABLE II: Performance portability of GPU-STREAM 2.0, where H is the subset of platforms supported by the implementation. Configurations can be found in [6], § 4.

We apply our performance portability metric to different subsets of our example platforms in Table I(b) to highlight interesting properties of the metric. In the first subset, the inclusion of Platform D causes $\mathbf{\Phi}$ to be zero, regardless of which efficiency metric is used – our metric cannot be used to obscure unsupported platforms, and this encourages application comparisons to consider only platforms that are supported by all applications. This also may encourage comparisons across different meaningful subsets of platforms (*e.g.* an application has a $\mathbf{\Phi}$ of X% across CPUs, Y% across accelerators and Z% across both). The other subsets demonstrate that $\mathbf{\Phi}$ for one application can vary significantly depending on which platforms are being considered; the inclusion of Platform E in particular significantly lowers $\mathbf{\Phi}$ when application efficiency is used, because the harmonic mean tracks the minimum value.

Our performance portability metric is well-defined, albeit trivial, for a set of just one platform (A). We see no reason to require two or more platforms, since such a restriction may be circumvented by different interpretations of "platform": it is entirely fair to consider two different products based on the same microarchitecture (*e.g.* two processors with different core counts or frequencies) to be distinct platforms.

B. Real-life Examples

We also retroactively apply our metric to some previous application studies (including two of our own). In order to be able to compute efficiency (either application or architectural), we are limited to studies which include some comparison to achievable performance across multiple platforms. Note that because these studies were performed by different researchers and at different times, each study uses a different set of platforms[3] – the performance portability measurements here are used to demonstrate the utility of the metric, and not to directly compare different research efforts.

1) GPU-STREAM: The GPU-STREAM benchmark [6] is a reimplementation of McCalpin's STREAM benchmark [17] in 7 different programming models, and its authors provide performance numbers for 12 different hardware platforms. Since STREAM is bandwidth-bound (by design), we can easily compute both architectural and application efficiency from these results: the theoretical peak performance corresponds to the peak bandwidth reported in the platform's specification sheet; and the best-known performance is the highest achieved by any GPU-STREAM implementation on the platform.

The results in Table II show the performance portablity for each implementation of GPU-STREAM on the subset of platforms that the implementation supports. From this, we see three clear groupings: 1) McCalpin; 2) RAJA, Kokkos and OpenMP (C++); and 3) SYCL, OpenACC, CUDA and OpenCL. These results demonstrate the danger in comparing performance portability computed with different platform sets – McCalpin has the highest performance portability across the platforms that it supports (and this is useful information), but the other implementations support a greater number of platforms based on a wider variety of microarchitectures.

Table III presents the same performance data relative to three different platform sets: 1) CPUs (five x86-based processors); 2) GPUs (four NVIDIA[4] GPUs); and 3) the union of the sets in 1) and 2). No implementation of GPU-STREAM runs across all 12 of the platforms, but the nine platforms in these subsets are supported by five of the programming models.

The $\mathbf{\Phi}$ measurements across CPUs are notably lower than the equivalent measurements across GPUs, and this is reflected in the measurements across the union of both subsets; the fastest way for the authors of GPU-STREAM to improve the performance portability of their benchmark across *both* CPUs and GPUs is to close the performance gap between GPU-STREAM and STREAM on CPUs. Whether or not it is possible to match STREAM performance using each of these programming models on CPUs (*i.e.* whether the programming model forces the algorithm to be structured in a way that limits performance) remains to be seen, but exposing such imbalance in platform support is important – it allows end-users to make more informed decisions regarding application/platform selection, and encourages application developers to expend more effort in the platforms where efficiency is lowest.

2) Other Benchmarks: Table IV presents $\mathbf{\Phi}$ values computed using a combination of architectural and application efficiencies from a number of additional performance portability

---

[3]Configurations are available in the original papers, as specified by Tables II, III and IV. See also the disclaimers in § VII.

[4]Other names and brands may be claimed as the property of others.

Implementation	$\mathbf{\Psi}(a, p, \textbf{CPUs})$		$\mathbf{\Psi}(a, p, \textbf{GPUs})$		$\mathbf{\Psi}(a, p, \textbf{CPUs} \cup \textbf{GPUs})$	
	Arch. Eff.	App. Eff.	Arch. Eff.	App. Eff.	Arch. Eff.	App. Eff.
McCalpin	75.66%	99.76%	0.00%	0.00%	0.00%	0.00%
SYCL	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
RAJA	58.11%	76.08%	72.92%	99.22%	63.88%	84.88%
Kokkos	57.42%	75.08%	73.23%	99.63%	63.51%	84.32%
OpenMP (C++)	57.49%	75.37%	68.00%	92.42%	61.73%	82.10%
OpenACC	38.21%	50.00%	70.22%	95.64%	47.91%	63.46%
CUDA	0.00%	0.00%	72.37%	98.50%	0.00%	0.00%
OpenCL	35.28%	46.15%	73.22%	99.65%	45.84%	60.61%

TABLE III: Performance portability of GPU-STREAM 2.0 for three platform sets. Configurations can be found in [6], § 4.

Application	Problem	Model	Approach	$\mathbf{\Psi}(a, p, H)$		Configuration
				Arch. Eff.	App. Eff	
miniMD	256k	OpenCL	Specialization	1.60%	54.03%	[23], § III-C
NAS-LU	Class C	OpenCL	Specialization	-	76.32%	[22], §§ 5.1 and 6.1
D3Q19-BGK	128^3	OpenCL	Agnostic	31.21%	91.70%	[18], §§ 3 and 5.1
ROTORSIM	Cylinder	OpenCL	Agnostic	29.67%	100.00%	[18], §§ 3 and 5.2
CloverLeaf	1920×3840	OpenCL	Agnostic	37.31%	90.97%	[18], §§ 3 and 5.3
CloverLeaf	3840^2	OPS	DSL	95.96%	-	[26], § V
LUD	2048	OpenACC	Auto-tuning	-	46.75%	[28], Table 4 and § 4.1
HOTSPOT	4096^2	OpenACC	Auto-tuning	-	93.06%	[28], Table 4 and § 4.1
SpMV	2.7m	OpenCL	Agnostic	5.93%	-	[33], Table 1 and § 2.1

TABLE IV: Performance portability results for some previous application studies.

studies. For each study, we note not only the programming model used but also the design philosophy of the developers: performing some *specialization* of the source code for each platform; employing only general optimizations that are *agnostic* to platform selection; using a *DSL* to generate efficient binaries for each platform; and *auto-tuning* a parameterized implementation to find the best fit for a platform.

There is not enough data here to draw conclusions about the effectiveness of different development approaches and programming models, but variation in the metric across studies and approaches demonstrates the importance of an objective and comparable measure of performance portability.

V. DISCUSSION

$\mathbf{\Psi}$ is a useful aggregate measure of performance and portability, but it is only meaningful when published alongside the set of platforms and the problem used to compute it. When reading a $\mathbf{\Psi}$ measurement, careful attention should be paid to the platforms and problem the authors have chosen, and generalizations beyond what was recorded should be made only with great caution; as with any metric [3], it is possible to hide and misrepresent information.

The simplest (and most meaningful) way to use our metric is to compare performance portability when only one variable is changed (*e.g.* one application executing one problem across multiple platform sets, multiple applications executing one problem across one platform set, etc). Comparing the performance portability of different applications executing different problems on different platform sets *may* be useful to draw broad conclusions regarding programming model and design philosophy choices, but any such comparisons should be made attentively (and with many more datapoints).

Although both types of performance efficiency considered in this paper are compatible with the calculation of $\mathbf{\Psi}$, it

is clear from our examples that they have different strengths. Given multiple implementations of an application and an in-depth knowledge of the problem, application efficiency is more representative – it conveys not only the application's performance, but also any performance penalty arising from the abstraction that it relies upon to provide portability. However, we acknowledge that it may be difficult to compute in practice (since it requires a pre-existing best-known performance result or an accurate performance model). Given a single implementation of an application and no pre-existing knowledge of expected performance, architectural efficiency is more representative (since the application efficiency is 100% on every platform!), but it obscures cases where better algorithms would have provided better performance on some platforms. The efficiencies are complementary, and presenting both is the easiest way to address their shortcomings.

VI. SUMMARY

We have proposed a novel, quantitative metric for performance portability. This metric is simple to compute, is representative of an application's performance *and* portability, and can be used to meaningfully compare the performance of different applications and high-level frameworks across the same base set of hardware platforms.

Adoption of our metric for performance portability is an important step towards productive debate, discussion and collaboration on the topic. We anticipate further demonstrations of its utility by other studies, and intend to investigate the degree to which existing programming models and design philosophies permit or prevent high levels of performance portability in future work.

VII. Disclaimers

Intel, the Intel logo, Intel Xeon, Intel Xeon Phi and Intel VTune are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit www.intel.com/benchmarks.

Intel does not control or audit third-party benchmark data or the other papers referenced in this document. You should visit the referenced documents and confirm whether referenced data are accurate.

Acknowledgements

References

[1] DOE Centres of Excellence Performance Portability Meeting: Post-meeting Report. Technical Report LLNL-TR-700962, Lawrence Livermore National Laboratory, 2016. 1, 2

[2] J. Ansel, C. Chan, Y. L. Wong, M. Olszewski, Q. Zhao, A. Edelman, and S. Amarasinghe. PetaBricks: A Language and Compiler for Algorithmic Choice. In *Proceedings of the 30th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '09, pages 38–49, New York, NY, USA, 2009. ACM. 1

[3] D. H. Bailey. Twelve Ways to Fool the Masses When Giving Performance Results on Parallel Computers. Technical Report RNR-91-020, NASA Ames Research Center, 1991. 5

[4] J.-S. Camier. Improving performance portability and exascale software productivity with the numerical programming language. In *Proceedings of the 3rd International Conference on Exascale Applications and Software*, pages 126–131. University of Edinburgh, 2015. 1

[5] M. Christen, O. Schenk, and H. Burkhart. PATUS: A Code Generation and Autotuning Framework for Parallel Iterative Stencil Computations on Modern Microarchitectures. In *Parallel Distributed Processing Symposium (IPDPS), 2011 IEEE International*, pages 676–687, May 2011. 1

[6] T. Deakin, J. Price, M. Martineau, and S. McIntosh-Smith. *GPU-STREAM v2.0: Benchmarking the Achievable Memory Bandwidth of Many-Core Processors Across Diverse Parallel Programming Models*. Lecture Notes in Computer Science. Springer, 5 2016. 1, 4, 5

[7] Z. DeVito, N. Joubert, F. Palacios, S. Oakley, M. Medina, M. Barrientos, E. Elsen, F. Ham, A. Aiken, K. Duraisamy, et al. Liszt: A Domain Specific Language for Building Portable Mesh-based PDE Solvers. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, page 9. ACM, 2011. 1

[8] H. C. Edwards, C. R. Trott, and D. Sunderland. Kokkos: Enabling Manycore Performance Portability Through Polymorphic Memory Access Patterns. *Journal of Parallel and Distributed Computing*, 74(12):3202 – 3216, 2014. Domain-Specific Languages and High-Level Frameworks for High-Performance Computing. 1, 2

[9] K. Fatahalian, D. R. Horn, T. J. Knight, L. Leem, M. Houston, J. Y. Park, M. Erez, M. Ren, A. Aiken, W. J. Dally, and P. Hanrahan. Sequoia: Programming the Memory Hierarchy. In *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, SC '06, New York, NY, USA, 2006. ACM. 1

[10] T. Gysi, C. Osuna, O. Fuhrer, M. Bianco, and T. C. Schulthess. STELLA: A Domain-specific Tool for Structured Grid Methods in Weather and Climate Models. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '15, pages 41:1–41:12, New York, NY, USA, 2015. ACM. 1

[11] J. A. Herdman, W. P. Gaudin, S. McIntosh-Smith, M. Boulton, D. A. Beckingsale, A. C. Mallinson, and S. A. Jarvis. Accelerating Hydrocodes with OpenACC, OpenCL and CUDA. In *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:*, pages 465–471, Nov 2012. 1, 2

[12] R. D. Hornung and J. A. Keasler. The RAJA Portability Layer: Overview and Status. Technical Report LLNL-TR-661403, Lawrence Livermore National Laboratory, September 2014. 1

[13] Khronos OpenCL Working Group. *The OpenCL Specification*, 2.2 edition, March 2016. 1

[14] K. Komatsu, K. Sato, Y. Arai, K. Koyama, H. Takizawa, and H. Kobayashi. Evaluating Performance and Portability of OpenCL Programs. In *The Fifth International Workshop on Automatic Performance Tuning*, volume 66, 2010. 1

[15] J. Larkin. Performance Portability Through Descriptive Parallelism. DOE CoE Performance Portability Workshop 2016, April 2016. 1, 2

[16] M. Martineau, S. McIntosh-Smith, and W. Gaudin. Assessing the Performance Portability of Modern Parallel Programming Models using TeaLeaf. *Concurrency and Computation: Practice and Experience*, 4 2016. 2

[17] J. D. McCalpin. Memory Bandwidth and Machine Balance in Current High Performance Computers. *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter*, pages 19–25, December 1995. 4

[18] S. McIntosh-Smith, M. Boulton, D. Curran, and J. Price. *On the Performance Portability of Structured Grid Codes on Many-Core Computer Architectures*, pages 53–75. Springer International Publishing, Cham, 2014. 1, 2, 5

[19] G. Mudalige, M. Giles, I. Reguly, C. Bertolli, and P. J. Kelly. OP2: An Active Library Framework for Solving Unstructured Mesh-based Applications on Multi-core and Many-core Architectures. In *Proceedings of Innovative Parallel Computing (InPar), 2012*, pages 1–12. IEEE, 2012. 1

[20] OpenACC-Standard.org. *The OpenACC Application Programming Interface*, 2.5 edition, October 2015. 1

[21] OpenMP Architecture Review Board. *OpenMP Application Programming Interface*, 4.5 edition, November 2015. 1

[22] S. J. Pennycook, S. D. Hammond, S. A. Wright, J. A. Herdman, I. Miller, and S. A. Jarvis. An Investigation of the Performance Portability of OpenCL. *J. Parallel Distrib. Comput.*, 73(11):1439–1450, Nov. 2013. 1, 5

[23] S. J. Pennycook and S. A. Jarvis. Developing Performance-Portable Molecular Dynamics Kernels in OpenCL. In *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:*, pages 386–395, Nov 2012. 1, 5

[24] J. Ragan-Kelley, C. Barnes, A. Adams, S. Paris, F. Durand, and S. Amarasinghe. Halide: A Language and Compiler for Optimizing Parallelism, Locality, and Recomputation in Image Processing Pipelines. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '13, pages 519–530, New York, NY, USA, 2013. ACM. 1

[25] F. Rathgeber, G. R. Markall, L. Mitchell, N. Loriant, D. A. Ham, C. Bertolli, and P. H. J. Kelly. Pyop2: A high-level framework for performance-portable simulations on unstructured meshes. In *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:*, pages 1116–1123, Nov 2012. 1

[26] I. Z. Reguly, G. R. Mudalige, M. B. Giles, D. Curran, and S. McIntosh-Smith. The OPS domain specific abstraction for multi-block structured grid computations. In *Domain-Specific Languages and High-Level Frameworks for High Performance Computing (WOLFHPC), 2014 Fourth International Workshop on*, pages 58–67. IEEE, 2014. 1, 5

[27] S. Rul, H. Vandierendonck, J. D'Haene, and K. De Bosschere. An Experimental Study on Performance Portability of OpenCL Kernels. In *2010 Symposium on Application Accelerators in High Performance Computing (SAAHPC'10)*, 2010. 1

[28] A. Sabne, P. Sakdhnagool, S. Lee, and J. S. Vetter. *Evaluating Performance Portability of OpenACC*, pages 51–66. Springer International Publishing, Cham, 2015. 1, 5

[29] N. Satish, C. Kim, J. Chhugani, H. Saito, R. Krishnaiyer, M. Smelyanskiy, M. Girkar, and P. Dubey. Can traditional programming bridge the ninja performance gap for parallel computing applications? In *Proceedings of the 39th Annual International Symposium on Computer*

Architecture, ISCA '12, pages 440–451, Washington, DC, USA, 2012. IEEE Computer Society. 3

[30] J. E. Smith. Characterizing computer performance with a single number. *Commun. ACM*, 31(10):1202–1206, Oct. 1988. 3

[31] Y. Tang, R. A. Chowdhury, B. C. Kuszmaul, C.-K. Luk, and C. E. Leiserson. The Pochoir Stencil Compiler. In *Proceedings of the Twenty-third Annual ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '11, pages 117–128, New York, NY, USA, 2011. ACM. 1

[32] C. R. Trott, S. D. Hammond, D. Dinge, P. T. Lin, C. T. Vaughan, J. Cook, H. C. Edwards, M. Rajan, and R. Hoekstra. ASC Trilab L2 Codesign Milestone 2015 – Sandia. Technical Report SAND2015-7886, Sandia National Laboratories, 2015. 2

[33] Y. Zhang, M. Sinclair, and A. A. Chien. *Improving Performance Portability in OpenCL Programs*, pages 136–150. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. 1, 5

[34] W. Zhu, Y. Niu, and G. Gao. Performance Portability on EARTH: A Case Study Across Several Parallel Architectures. *Cluster Computing*, 10(2):115–126, 2007. 2