RAJA Performance Suite: Performance Portability Analysis with Caliper and Thicket

Olga Pearce*‡, Jason Burmark*, Rich Hornung*, Befikir Bogale†, Ian Lumsden†, Michael McKinsey‡, Dewi Yokelson*, David Boehme*, Stephanie Brink*, Michela Taufer†, Tom Scogland*
*Lawrence Livermore National Laboratory, Livermore, CA, USA
†Department of Electrical Engineering and Computer Science, University of Tennessee, Knoxville, TN, USA
‡Department of Computer Science and Engineering, Texas A&M University, College Station, TX, USA

***Abstract*—Maintaining performant code in a world of fast-evolving computer architectures and programming models poses a significant challenge to scientists. Typically, benchmark codes are used to model some aspects of a large application code's performance, and are easier to build and run. Such benchmarks can help assess the effects of code or algorithm changes, system updates, and new hardware. However, most performance benchmarks are not written using a wide range of GPU programming models. The RAJA Performance Suite provides a comprehensive set of computational kernels implemented in a variety of programming models. We integrated the performance measurement and analysis tools Caliper and Thicket into the RAJA Performance Suite to facilitate performance comparison across kernel implementations and architectures. This paper describes the RAJA Performance Suite, performance metrics that can be collected, and experimental analysis with case studies. The RAJA Performance Suite and integration of Caliper and Thicket enabled us to effectively measure the variety of implementations on the variety of hardware, and automatically characterize subsets of kernels which exhibit similar bottlenecks—and therefore perform similarly on new architectures which provide a different balance between resources such as FLOPS and memory bandwidth. We definitively demonstrate that the most memory bound kernels show the most performance gains on architectures with high-bandwidth memory (HBM), and that the kernels that have other bottleneck may, to a lesser extent, benefit from the higher-FLOPS GPUs.**

I. Introduction

The rapid advancement of accelerator hardware technology in HPC has provided substantial performance gains for many applications, but writing code that runs well on a wide range of hardware is not simple and achieving good performance often requires architecture-specific implementations. Thus, scientists may be constrained to run their applications on a single platform, unable to run elsewhere without significant work to develop multiple versions of their application code. As systems age and are replaced by new and different architectures, this problem intensifies. This problem has led to the development of performance portability models (*e.g.,* OpenMP [1], RAJA [2], [3], and Kokkos [4], [5]) that enable scientists to write *single-source* parallel code that can be compiled to run on many different architectures without sacrificing application performance. Such portability layers provide frameworks to compare performance across architectures and to more easily port applications to new systems.

Additionally, as new systems come online, they must be evaluated and vetted for the codes that will run on them. This is usually done using benchmark suites representing key aspects of full applications. There is a long history of benchmark suites that represent computations performed at national laboratories [6], [7], [8] and the HPC field as a whole [9], [10]). The RAJA Performance Suite (RAJAPerf) is an evolving suite of computational kernels of interest to Lawrence Livermore National Laboratory (LLNL). Specifying how to run such suites is a challenge of its own [11], necessitating the additional development of increasingly automated runtime specifications [12], [13].

RAJAPerf enables performance analysis for kernels with a variety of different algorithms and programming models. Analysis of these kernels is helpful for both platform procurement and application porting. Evaluation of new machines requires benchmark analysis to assess the readiness and feasibility of new architectures. In addition, this analysis can evaluate the health of aging hardware, and help determine when it may be time to retire systems. Another benefit is to help decide whether to port code by extrapolating performance for applications with similar algorithmic characteristics to the kernels. RAJAPerf provides useful performance-related metrics, and by combining these with the collection of metrics and analysis from other tools, we gain more insight into the execution behavior of these kernels.

In this work, we leverage Caliper [14], [15] and Nsight Compute [16] to measure time and hardware counters. We also use Thicket [17], a Python tool that enables exploratory data analysis (EDA) of parallel performance data. We employ the Intel Top-Down Analysis [18] on CPUs, and the Roofline analysis [19] on GPUs. We perform similarity analysis to characterize similarities and differences across architectures and kernels. The main contributions of this work are:

1) Comprehensive overview of the RAJA Performance Suite (RAJAPerf), a curated set of kernels implemented in a variety of CPU and GPU programming models.
2) Integration of a portable performance analysis toolchain, including Caliper for recording measurements, and Thicket for Exploratory Data Analysis (EDA).
3) Analysis of performance portability of the RAJA programming model across four architectures.

TABLE I: RAJAPerf kernels organized into seven groups, kernel programming model implementations (B or R for Base or RAJA variants), kernel features, and complexities.

Group	Kernel Name	Programming Models													Features					Complexity
		Sequential		OpenMP		OMPTarget		CUDA		ROCM		SYCL		Kokkos	Sorts	Scans	Reducts	Atomics	Views	Complexity
		B	R	B	R	B	R	B	R	B	R	B	R							O( )
Algorithms	ATOMIC	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓							✓		n
	HISTOGRAM	✓	✓	✓	✓	✓		✓	✓	✓	✓			✓				✓		n
	MEMCPY	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓									n
	MEMSET	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓									n
	REDUCE_SUM	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓				✓			n
	SCAN	✓	✓	✓	✓	✓		✓	✓	✓	✓					✓				n
	SORT	✓	✓		✓				✓		✓				✓					$n \lg n$
	SORTPAIRS	✓	✓		✓				✓		✓				✓					$n \lg n$
Applications	CONVECTION3DPA	✓	✓	✓	✓			✓	✓	✓	✓	✓	✓							n
	DEL_DOT_VEC_2D	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓							n
	DIFFUSION3DPA	✓	✓	✓	✓			✓	✓	✓	✓	✓	✓							n
	EDGE3D	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓							n
	ENERGY	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓							n
	FIR	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓							n
	LTIMES	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓						✓	n
	LTIMES_NOVIEW	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓							n
	MASS3DEA	✓	✓	✓	✓			✓	✓	✓	✓	✓	✓							n
	MASS3DPA	✓	✓	✓	✓			✓	✓	✓	✓	✓	✓							n
	MATVEC_3D_STENCIL	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓							n
	NODAL_ACCUMUL_3D	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓							✓		n
	PRESSURE	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓							n
	VOL3D	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓							n
	ZONAL_ACCUMUL_3D	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓							n
Basic_Patterns	ARRAY_OF_PTRS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓						n
	COPY8	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓							n
	DAXPY	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓						n
	DAXPY_ATOMIC	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓			✓						n
	IF_QUAD	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓						n
	INDEXLIST	✓		✓		✓		✓		✓						✓				n
	INDEXLIST_3LOOP	✓	✓	✓	✓	✓		✓	✓	✓	✓					✓				n
	INIT3	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓						n
	INIT_VIEW1D	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓					✓	n
	INIT_VIEW1D_OFFSET	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓					✓	n
	MAT_MAT_SHARED	✓	✓	✓	✓			✓	✓	✓	✓	✓	✓							$n\sqrt{n}$
	MULADDSUB	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓						n
	MULTI_REDUCE	✓	✓	✓	✓	✓		✓	✓	✓	✓			✓				✓		n
	NESTED_INIT	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓						n
	PI_ATOMIC	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓			✓				✓		n
	PI_REDUCE	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓				✓			n
	REDUCE3_INT	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓			✓			n
	REDUCE_STRUCT	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓						✓			n
	TRAP_INT	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓			✓			n
Comm	HALO_EXCHANGE	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓									$n^{2/3}$
	HALO_EXCH_FUSED	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓									$n^{2/3}$
	HALO_PACKING	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓									$n^{2/3}$
	HALO_PACKING_FUSED	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓									$n^{2/3}$
	HALO_SENDRECV	✓		✓		✓		✓		✓										$n^{2/3}$
LCALS	DIFF_PREDICT	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓						n
	EOS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓						n
	FIRST_DIFF	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓						n
	FIRST_MIN	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓			✓			n
	FIRST_SUM	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓						n
	GEN_LIN_RECUR	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓						n
	HYDRO_1D	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓						n
	HYDRO_2D	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓						n
	INT_PREDICT	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓						n
	PLANCKIAN	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓						n
	TRIDIAG_ELIM	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓						n
Polybench	2MM	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓							$n\sqrt{n}$
	3MM	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓							$n\sqrt{n}$
	ADI	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓							n
	ATAX	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓							n
	FDTD_2D	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓							n
	FLOYD_WARSHALL	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓							$n\sqrt{n}$
	GEMM	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓							$n\sqrt{n}$
	GEMVER	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓							n
	GESUMMV	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓							n
	HEAT_3D	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓							n
	JACOBI_1D	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓							n
	JACOBI_2D	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓							n
	MVT	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓							n
Stream	ADD	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓						n
	COPY	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓						n
	DOT	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓			✓			n
	MUL	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓						n
	TRIAD	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓						n

Section II describes the RAJAPerf suite, its curated kernels, the programming models used, and the integrated portable performance analysis toolchain. Section III details the performance metrics we considered across the architectures. Sections IV and V showcase performance portability analysis enabled by RAJAPerf. Section VI describes related work.

II. RAJA Performance Suite

The RAJA Performance Suite (RAJAPerf) [20] is an established suite offering a comprehensive collection of benchmark and real application kernels. RAJAPerf provides multiple *variants* (or implementations) of each kernel, each performing the same computation. Initially, the suite was created to compare the performance of kernels implemented using the RAJA performance portability layer [2], [3] with the performance of those kernels implemented directly in programming models appropriate for specific architectures. Now, the suite is used to assess new hardware architectures and is an important collaboration tool for interactions between application teams and hardware, compiler, and runtime vendors. Both RAJA and non-RAJA (i.e., *baseline*) variants are available for Sequential (C++ code), OpenMP, OpenMP Target Offload, CUDA, HIP, and SYCL programming models. Variants are also used to identify C++ features, such as lambda expressions. For each programming model, a RAJAPerf kernel includes at least two variants: (1) a baseline variant implemented directly in a particular programming model such as CUDA, and (2) a RAJA variant implemented using the RAJA portability layer, which insulates the kernel source code from the programming model implementation details. Some kernels also appear using the Kokkos portability abstraction layer. These are developed and maintained separately by the Kokkos team. Performance analysis and results associated with those are not discussed in this article. Each kernel contains a reference description in the form of a C-style, sequential for-loop implementation. The suite features over 70 unique kernels, as detailed in Table I, with new kernels regularly added. It is important to note that not all kernels are available in all programming models and variants.

A. Curated Kernels

Each kernel in RAJAPerf is a self-contained loop-based computation. As a whole, the suite represents a spectrum of algorithms used in many application codes. A *group* is a subset of kernels in the suite that originate from a single benchmark suite, or represent a related set of computational patterns. Kernels in RAJAPerf fall into the following seven groups:

1) *Algorithms*: Kernels that focus on specific parallel constructs, such as atomic operations, scans, reductions, and sorts, and memory operations like *memcpy* and *memset*.
2) *Applications*: Kernels derived from important application operations in various LLNL mutliphysics applications.
3) *Basic_Patterns*: Kernels that are small and simple, yet often present optimization challenges for compilers.
4) *Comm*: Communication buffer packing/unpacking patterns from distributed memory applications using MPI.
5) *LCALS*: Kernels originating in the Livermore Loops suite [6], designed to explore Fortran compiler vectorization, and subsequently translated into C++ in *LCALS - Livermore Compiler Analysis Loop Suite*. LCALS was an LLNL-internal suite designed to study the ability of C++

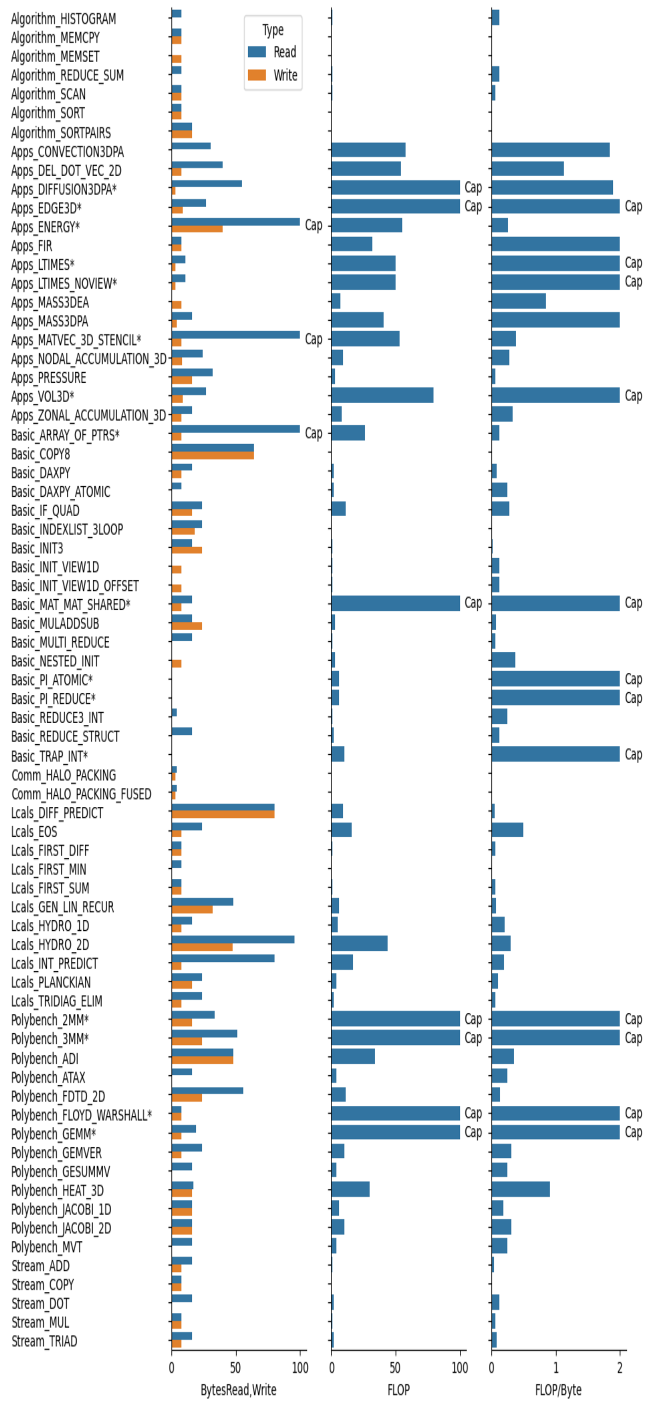


Fig. 1: Analytic metrics per kernel iteration. A "*" next to a kernel name indicates one or more metric values, denoted with "Cap", is shown as a truncated value on the axis despite the value being larger, to enable seeing smaller values for other kernels on the same scale.

compilers to optimize kernels that employ C++ templates and lambda expressions, which are core to RAJA.

6) *Polybench*: A subset of kernels from the Polybench Suite used to study polyhedral optimization in compilers [21].
7) *Stream*: Streaming kernels found in the McCalpin STREAM benchmark [22].

Kernels are annotated based on the RAJA *features* that they exercise such as sort, scan, reduction, atomic operations, and data views. Kernels are also annotated based on their computational complexity. Most kernels run a number of operations of the the same order as their data size, however some kernels do more or less work on that data and some use no stored data at all. For instance, some kernels perform matrix-matrix multiplication which uses $O(N^{\frac{3}{2}})$ operations relative to the size of the matrix storage.

RAJAPerf provides a wide variety of command line options to run subsets of kernels, such as specific groups, variants, or those that exercise specific RAJA features, and to run different kernel working set sizes, algorithm tuning options, etc. As a result, the suite enables a broad range of performance studies. Various text-based files can be generated for each run for processing with common plotting and other tools.

B. Analytic Metrics

Analytic metrics are platform-independent and provide valuable insights into the performance characteristics of kernels. RAJAPerf provides the following metrics for each kernel:

- *Bytes Read*: This metric measures the amount of data read from memory during the execution of a kernel.
- *Bytes Written*: Similar to bytes read, this metric measures the amount of data written to memory.
- *Floating-Point Operations (FLOPs)*: This metric counts the number of floating-point operations a kernel performs. It is a critical measure of computational workload and is used to evaluate the arithmetic intensity of a kernel.
- *FLOPs per Byte of Memory Touched*: This derived metric indicates how many floating-point operations are performed per byte of memory accessed.

These metrics provide a quantitative view of kernels' computational and memory demands. For example, Bytes Read and Bytes Written provide insight into memory access patterns and are crucial for understanding data movement overhead.

A default problem size is defined for each kernel. Problem size can also be chosen at runtime via a command line argument. Examples of problem size include the number of elements in a matrix or the size of a numerical grid in a simulation. Fig. 1 shows analytic metrics for each kernel, normalized by the kernel's problem size to facilitate comparison. At a glance, we see which kernels perform the most FLOPs, and which kernels stress the memory subsystem most.

These static metrics are pivotal at execution time; given the execution time of each kernel, we can use the metrics to derive the Read Memory Bandwidth Rate (*i.e.,* the rate at which data is read from memory), the Write Memory Bandwidth Rate (*i.e.,* the rate at which data is written to memory), and

TABLE II: Supercomputers used for the experiments, with FLOPS and memory bandwidth listed per single compute unit (GPU or CPU) and per node. Achieved FLOPS are measured using the Basic_MAT_MAT_SHARED kernel, achieved memory bandwidth using the Stream_TRIAD. The % exp columns denote the percentage of the theoretical hardware performance achieved by these kernels. Unit and node FLOPS numbers for EPYC-MI250X assume double-precision vector instructions.

Shorthand	System Name	Architecture executing kernels (CPU or GPU)	Units on one node	Rate of floating-point operations (TFLOPS) unit	node	Basic_MAT_MAT	% exp	Memory bandwidth (TB/s) unit	node	Stream_TRIAD	% exp
SPR-DDR	Poodle (DDR)	Intel Sapphire Rapids	2 sockets	2.3	4.7	0.8	18.0	0.3	0.6	0.5	77.7
SPR-HBM	Poodle (HBM)	Intel Sapphire Rapids	2 sockets	2.3	4.7	0.7	15.5	1.6	3.3	1.1	33.7
P9-V100	Sierra	NVIDIA V100	4 GPUs	7.8	31.2	7.0	22.4	0.9	3.6	3.3	92.6
EPYC-MI250X	Tioga	AMD MI250X	8 GPUs	24.0	191.5	13.3	7.0	1.6	12.8	10.2	79.5

the number of Floating-Point Operations per Second (FLOPs) (*i.e.,* the total number of floating-point operations performed divided by the execution time).

C. Focus on Understanding Performance

Performance analysis is the primary goal of RAJAPerf. *Execution time,* or the time it takes to complete the execution of a kernel, is the bottom line. In addition, RAJAPerf enables users to evaluate (1) kernel *scalability* with the increase in computational resources, such as more CPU cores or GPU threads; (2) kernel *computational efficiency*, including the effective use of hardware resources like CPU, GPU, and memory bandwidth; and (3) kernel *overhead* added by using RAJA abstractions compared to using programming models, such as OpenMP or CUDA directly. Users can assess kernel *portability*, or the ability to perform consistently across different architectures and programming environments, and find optimal configurations for specific hardware by *tuning* various execution parameters, such as GPU thread-block sizes. Runtime parameters enable running the entire suite or a subset of specified kernels, variants, and tunings.

D. Integrating Caliper and Thicket into RAJAPerf

We integrated two performance analytics tools into RAJAPerf to augment metric gathering and analysis: Caliper [14], [23] and Thicket [17], [24]. *Caliper* is a performance profiling library that provides performance study capabilities in HPC software stacks. Caliper is integrated into RAJAPerf similarly to how it is integrated in large-scale simulations. Kernels within RAJAPerf are annotated as Caliper regions, and the analytical kernel metrics calculated in RAJAPerf are collected as Caliper metrics associated with these regions. The Adiak [25] library annotates per-run metadata, such as the programming model used and the variant run. A single RAJAPerf run generates a Caliper profile containing one variant and one tuning. Caliper collects all requested performance data at runtime and generates a .cali file for each run. These files are then read into Thicket for analysis and visualization.

Thicket is an open-source Python toolkit for exploratory data analysis of multi-run performance experiments. It provides an interface for interacting with performance data, enabling an understanding of performance configurations for large-scale application codes. Thicket has a modular structure composed of three components: a multi-dimensional table of performance metrics per node matching a call tree profile for each application run; a metadata table including the application's build settings; and aggregated statistics summarizing performance metrics collected across runs. Thicket handles multiple runs by generating one profile per run, recording critical information about build settings and execution contexts in the metadata, and leveraging Thicket components to avoid redundant data representation. Using Thicket to simplify data composition involves reading multiple Caliper files (performance profiles) into Thicket, grouping the data by variants and tunings in the metadata table, and visualizing and analyzing the dataset using Thicket's capabilities. Such capabilities include performance composition from different runs across different configurations and parameters; performance data manipulation, organizing the data based on specific aspects; performance visualization to identify patterns and insights; and analysis and modeling of collected performance.

Integrating Caliper and Thicket into RAJAPerf enhances the ability to measure, analyze, and visualize performance metrics. Caliper and Thicket standardize the performance profiling format in RAJAPerf, thus facilitating knowledge extraction and enhancing the understanding of performance behavior across different programming models and hardware architectures.

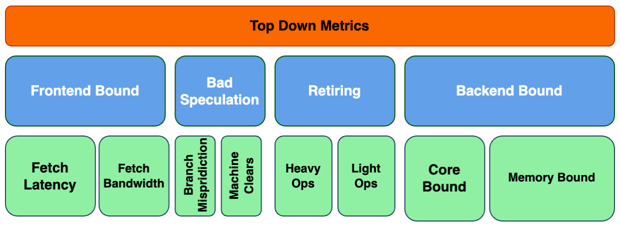


Fig. 2: Top-down structured hierarchical method to identify dominant bottleneck in out-of-order processors.

III. Performance Metrics

We conducted experiments on four different supercomputer platforms, detailed in Table II. These include two Intel CPU-only systems on the Poodle platform at LLNL: one is equipped with Sapphire Rapids and Double Data Rate Memory (SPR-DDR), and the other with Sapphire Rapids and High Bandwidth Memory (SPR-HBM). SPR-DDR operates with Double Data Rate (DDR) memory across two sockets, achieving 4.7 TFLOPS with a 0.6 TB/s memory bandwidth, reaching

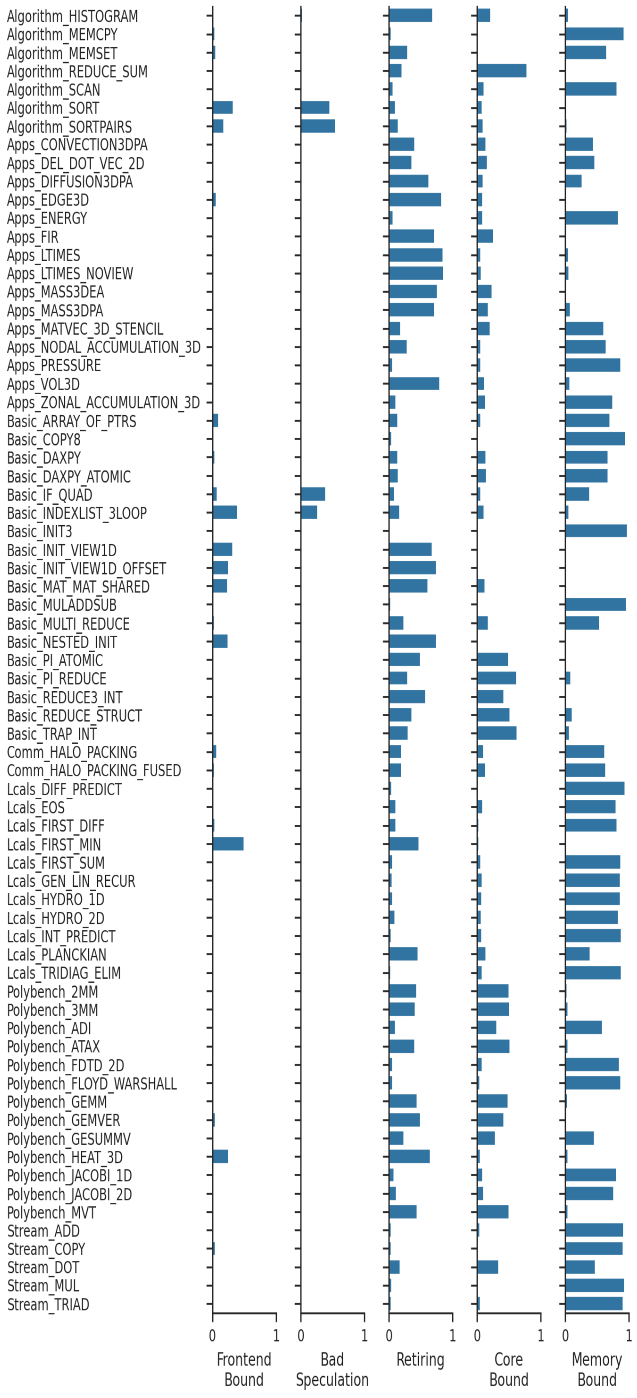


Fig. 3: SPR-DDR top-down metrics

77.7% of its expected memory bandwidth. In contrast, SPR-HBM uses High Bandwidth Memory (HBM) to deliver 4.7 TFLOPS at 3.3 TB/s, with 33.7% of its expected memory bandwidth. Additionally, we use two CPU+GPU integrated systems. One is the P9-V100 Sierra system at LLNL, which integrates IBM Power9 CPUs and NVIDIA V100 GPUs, providing 31.2 TFLOPS and a 3.6 TB/s memory bandwidth, achieving 92.6% of theoretical memory bandwidth. The other is the Tioga system at LLNL, which integrates AMD EPYC CPUs and AMD MI250X GPUs, producing an exceptional 192 TFLOPS and 12.8 TB/s, achieving 79.5% of its theoretical memory bandwidth. These metrics enable us to compare these system's substantial computational capabilities and efficiency in performing complex operations.

TABLE III: RAJAPerf parameters used in this paper. We kept the problem size per node constant across all systems at 32M.

System	Variant	Tuning	Num. MPI Processes	Problem Size Per Process
SPR-DDR	RAJA_Seq	default, funcptr	112	300K
SPR-HBM	RAJA_Seq	default, funcptr	112	300K
P9-V100	RAJA_CUDA	block_256, blkatm_occgs_256, block_64 block_25, funcptr_256, cub, default, atomic_occgs_256, blkdev_occgs_256	4	8M
EPYC-MI250X	RAJA_HIP	block_256, blkatm_occgs_256, block_64 block_25, funcptr_256, rocprim, default, atomic_occgs_256, blkdev_occgs_256	8	4M

For each of these systems, we run RAJAPerf with specific variants, tunings, number of MPI processes, and problem sizes per process to ensure a fair comparison for each kernel for a total problem size of 32,000,000 for the entire node. These parameters are shown in Table III. On the CPU systems (SPR-DDR and SPR-HBM), we use MPI for parallelism to maximize performance across the two sockets on each node. Additionally, we use 112 processes with the RAJA_Seq variant to fully load each of the 112 cores per node. On P9-V100, we use four processes with the RAJA_CUDA variant to make use of the four NVIDIA V100 GPUs on each node. On EPYC-MI250X, we use eight processes with the RAJA_HIP variant to utilize the eight GCDs available across the four AMD MI250X GPUs per node.

A. *Hardware Metrics on CPUs*

On CPUs, we use the industry-standard PAPI counters [26] to measure performance. To begin our analysis, we use the Top-down Microarchitecture Analysis (TMA) [18] method for out-of-order processors designed by Intel. The top-down analysis uses designated hardware performance counters in a hierarchical structure to identify dominant bottlenecks in the collected performance data. The analysis shown in Fig. 2 breaks down the observed CPU pipeline utilization into four broad categories: *Frontend Bound*, *Bad Speculation*, *Retiring*, and *Backend Bound*. Frontend Bound measures the initial stages of instruction processing, such as instruction fetch latency and bandwidth; Bad Speculation captures the costs associated with the CPU's predictive mechanisms; Retiring

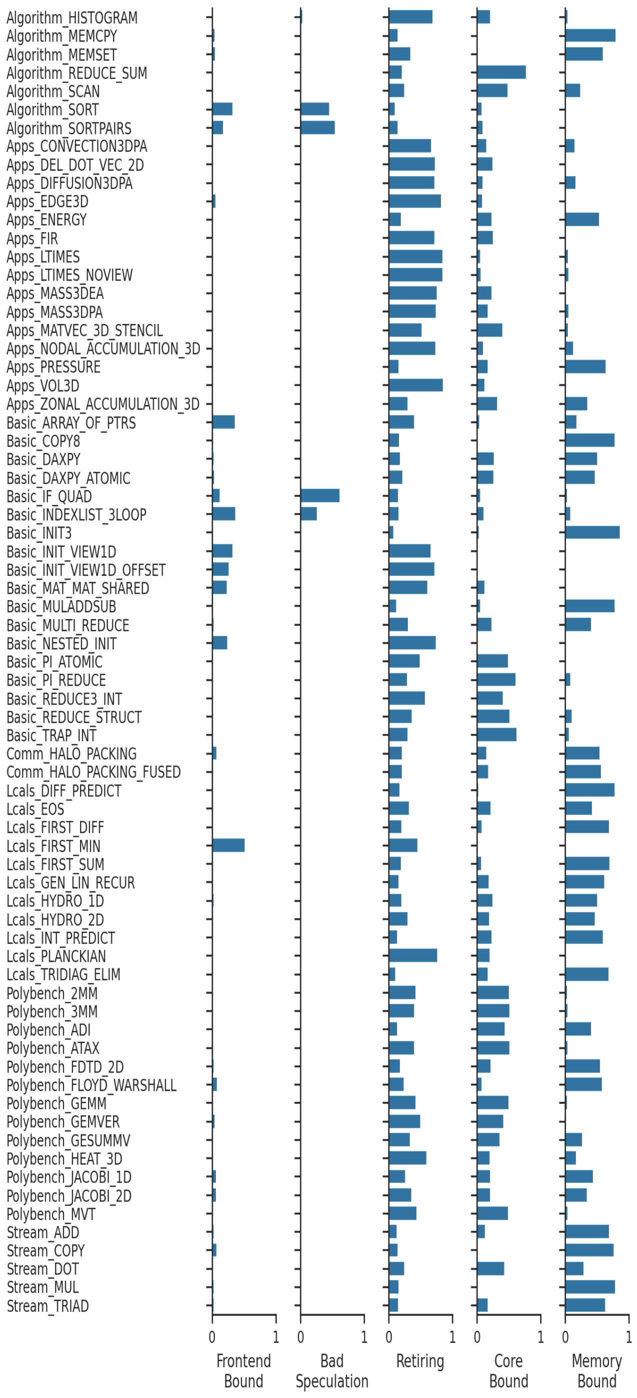


Fig. 4: SPR-HBM top-down metrics

computes the rate at which the CPU completes and retires instructions; and Backend Bound highlights delays due to data or resource availability needed to execute instructions. Each category is hierarchically divided into more detailed sub-categories to narrow down specific performance bottlenecks. Backend bound is comprised of *Core Bound* (saturation within the CPU core) and *Memory Bound* (saturation within the memory), which is further broken down by specific memory levels (L1, L2, L3) and external memory interactions, describing issues in data retrieval from various cache levels or RAM.

In this work, we focus only on the top two levels of the TMA hierarchy. Fig. 3 shows the top-down metrics across the suite when running on SPR-DDR. Fig. 4 shows the top-down metrics across the suite when running on SPR-HBM. The figures demonstrate how specific kernels are more memory bound on SPR-DDR than SPR-HBM, since the higher-bandwidth HBM (partially) alleviates the memory bandwidth bottleneck. Data-intensive kernels such as the Stream kernels, where the need to quickly read and write large volumes of data is a primary performance requirement, particularly benefit from HBM. Additionally, the higher throughput of HBM ensures that data-dependent kernels can operate more continuously without stalling for memory access, enhancing overall computational efficiency and performance. For instance, with Algorithm_SCAN, higher memory bound metric on SPR-DDR indicates that this kernel—often used for performing prefix sums—is limited by memory bandwidth constraints. The significantly lower memory bound metric on SPR-HBM suggests that the higher bandwidth of HBM effectively accommodates the high data throughput required by this algorithm. HBM also offers lower latency compared to DDR. For kernels involving frequent memory access, such as those performing multiple matrix operations, this reduced latency can significantly improve overall execution times. For instance, Polybench_GESUMMV displays substantial memory bound constraints with DDR. In this case, the kernel needs rapid memory access for subsequent multiplications, a task that DDR struggles to manage efficiently. In contrast, the reduced memory bounding in the HBM configuration highlights the efficient use of its superior bandwidth to streamline data access and processing speeds.

On the other hand, kernels can be more significantly limited by the compute resources available for performing operations than by memory speed. For instance, the bars representing the memory bound metric for Algorithm_REDUCE_SUM for both DDR and HBM are relatively low, suggesting that the bottleneck is not primarily in memory bandwidth. Similarly, Polybench_2MM and Polybench_ATAX also exhibit low memory bound metric values on both systems, indicating that their performance is likely more dependent on the efficiency of the compute operations rather than the speed of data retrieval from memory. The same patterns are observed in various application-specific kernels, such as Apps_MATVEC_3D_STENCIL.

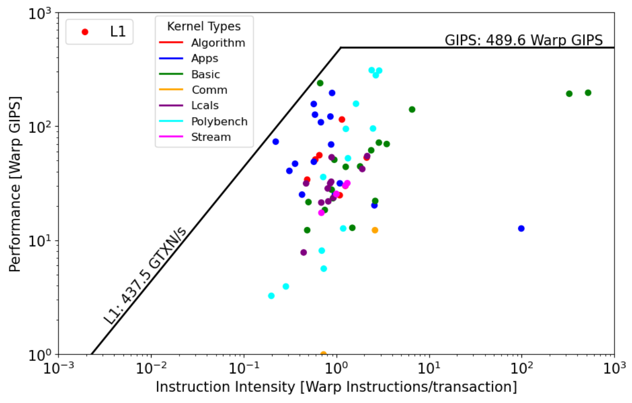
B. Hardware Metrics on GPUs

For NVIDIA GPUs, we utilize the Instruction Roofline Analysis model [19] to interpret performance measurements from NVIDIA's Nsight Compute CLI (NCU), including metrics like memory throughput, compute utilization, and latency. The Instruction Roofline Model is a performance analysis tool that visualizes performance (y-axis, measured in Warp Giga-Instructions Per Second, Warp GIPS) versus instruction intensity (x-axis, measured in Warp Instructions per Transaction) for various kernels, providing a comprehensive view of the GPU's performance limitations. Given the hardware constraints of the GPU, the ceiling line (*i.e.,* roofline) represents the maximum possible performance. The theoretical maximum bandwidth is shown on the diagonal line and is measured in 10^9 transactions per second (GTXN/s). The theoretical maximum instruction rate is shown on the horizontal line and is measured in 10^9 instructions per second (GIPS). Each point in the figure indicates a kernel's instruction intensity and performance per cache layer. When a kernel (*i.e.,* point) is farther from the horizontal roof of a model it indicates that performance is limited by the kernel not fully utilizing the GPU's processing power. On the other hand, when a kernel is farther from the diagonal roof it indicates that performance is limited by the kernel not efficiently accessing the GPU's memory resources. The metrics and equations for calculating kernel performance in our work were specified by Ding and Williams [27] and are shown in Table IV.

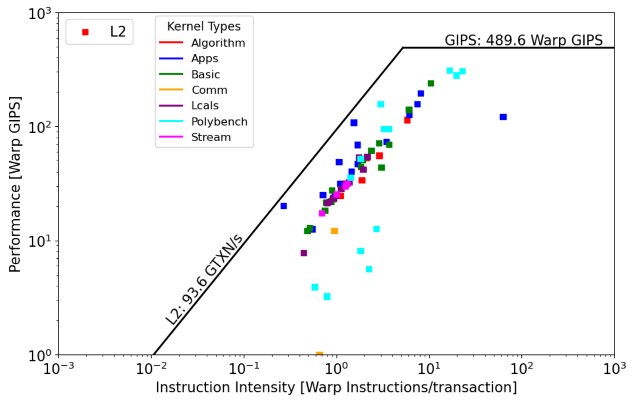
We used NCU and Caliper to collect the performance counters for the Instruction Roofline Analysis, used Thicket to read in the data, and applied the analysis to all RAJAPerf kernels. Fig. 5 shows the roofline models for L1, L2, and HBM cache layers on the NVIDIA V100 GPUs on the P9-V100 system. The plots help identify whether each kernel is compute bound or memory bound. Each point on a plot represents a specific kernel or computational task. For readability, we color the kernels by RAJAPerf *groups*—Algorithms, Apps, Basic, Communication, Lcals, Polybench, and Stream—and graphed them separately per cache layer. Points close to the horizontal segment use the compute resources efficiently but are limited by the maximum computational capability of the GPU (compute-bound). Points close to or along the diagonal segment are limited by memory bandwidth. Moving horizontally towards higher instruction intensities without significantly increasing performance suggests that improving the computation-to-memory access ratio can deliver better performance (memory bound). Points below the roofline suggest inefficiencies that could be due to several factors like poor

TABLE IV: Metrics for instruction roofline analysis for NVIDIA GPUs, collected using NVIDIA Nsight Compute.

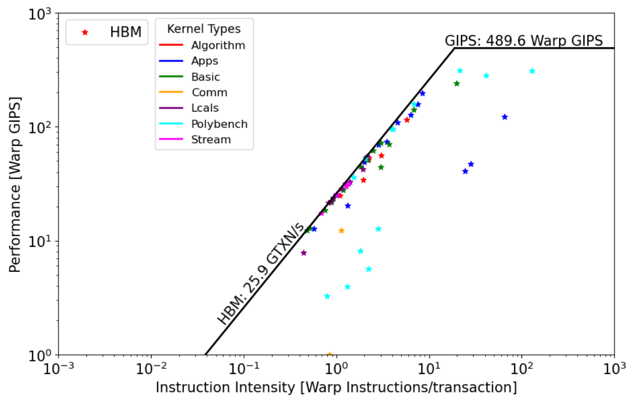
Category	Metric	Description
thread-based	sm__sass_thread_inst_executed.sum	non-predicated
warp-based	l1tex__t_sectors_pipe_lsu_mem_global_op_ld.sum l1tex__t_sectors_pipe_lsu_mem_global_op_st.sum l1tex__t_sectors_pipe_lsu_mem_local_op_ld.sum l1tex__t_requests_pipe_lsu_mem_local_op_st.sum	L1 cache transactions
	lts__t_sectors_op_read.sum lts__t_sectors_op_write.sum lts__t_sectors_op_atom.sum lts__t_sectors_op_red.sum	L2 cache
	dram__sectors_read.sum dram__sectors_write.sum	HBM memory
kernel-based	time (gpu)	execution time



(a) L1 cache instruction



(b) L2 cache instruction



(c) HBM cache instruction

Fig. 5: Instruction roofline model for the P9-V100 system, split into three figures for the different cache layers: L1, L2, and HBM. The theoretical maximum bandwidth is on the diagonal line in GTXN/s and the theoretical maximum instruction rate is shwon on the horizontal line in Warp GIPS.

cache utilization (where the cache is not effectively storing and retrieving data), thread divergence (where threads in a warp take different execution paths, reducing parallelism), or unoptimized data access patterns (where the data is not accessed in a way that maximizes cache efficiency).

The performance for different kernel types in Fig. 5a (L1 cache level) is quite varied, suggesting different behaviors or requirements in terms of data fetching and processing efficiency. The kernels near the theoretical maximum performance (roof) are compute bound and can efficiently utilize the GPU's computational resources. The kernels below the roof are either memory bound or not optimally utilizing the L1 cache. The spread of kernels in Fig. 5b (L2 cache level) is narrower than L1, indicating a more consistent use of the L2 cache across different kernels. Fewer kernels reach closer to the roof than for L1, indicating that the L2 cache could be more effectively used or that these kernels are more memory bound with respect to L2 than L1. The dispersion of kernels in Fig. 5c (HBM) suggests a clear division between higher-performing kernels and those that lag. In other words, some kernels can efficiently leverage high-bandwidth memory to achieve near-optimal performance. However, others, like most Polybench kernels, are limited by memory access patterns or other factors preventing them from fully exploiting HBM.

IV. Kernel Similarity Analysis

We employ clustering techniques to understand which kernels perform similarly across different architectures. For comparison purposes, we use a problem size of 32,000,000 per node on each of the four architectures, as described in Table III. Because this problem is decomposed to run on 112 cores on the CPU systems, four GPUs on P9-V100, and eight GPUs on EPYC-MI250X, the decomposition introduces incomparable amounts of work for the kernels in the suite with complexity other than $O(N)$, such as our Halo communication kernels. We thus exclude 12 out of 75 kernels (16%) from our subsequent analysis.

Each kernel is represented by a tuple of top-down metrics (*i.e.,* frontend bound, bad speculation, retiring, core bound, and memory bound). We use agglomerative, bottom-up, hierarchical clustering to group the kernel tuples, and we use Euclidean distance to measure the distance between tuples. We apply the Ward merge strategy [28] to minimize variance within each cluster. In the Ward strategy, we set the distance threshold to 1.4, identifying four distinct clusters with similar kernels. Fig. 6 visualizes the clustering results using a dendrogram.

Fig. 7 outlines how the four clusters are characterized by the distinct patterns defined by the top-down metrics. Fig. 7 also shows how the RAJAPerf kernel groups are distributed across the four clusters (Fig. 6). Cluster 2 is the most memory bound, and includes nearly all Stream and LCALS kernels. It achieves the highest speedup on all three HBM platforms, with an impressive 22.6x on EPYC-MI250X over SPR-DDR. Cluster 0 is the second most memory bound, and includes 40% of the applications kernels. It achieves the second highest speedup on all three HBM platforms, including 14x on EPYC-MI250X over SPR-DDR. Cluster 3 is the most core bound, and includes roughly a quarter of Basic and Polybench kernels. Because it is not memory bound, it achieves a modest speedup on the GPUs, with 6.3x on EPYC-MI250X over SPR-DDR. Cluster 1 is the most frontend bound, and includes 40% of the applications kernels. It achieves a modest speedup on the GPUs, with 7.1x on EPYC-MI250X over SPR-DDR.

Fig. 8 shows a parallel coordinate plot that links the average TMA values with the average speedup over the three different architectures for each cluster (color). The pattern visualized here demonstrates the commmonalities in the autogenerated clusters: Cluster 2 is the most memory bound and had the highest speedup across all three systems since they all have higher memory bandwidth. Cluster 0 is the second highest memory bound and exhibited the second highest speedup across all three systems. Cluster 1 and Cluster 3 are not memory bound and had the least speedup on the systems that predominantly improve the memory bandwidth over the SPR-DDR system we used as the baseline.

V. Memory Speed and FLOPS Trade-Offs

Identifying and measuring the bottlenecks of each RAJAPerf kernel using TMA provides valuable insights for predicting potential speedups on different architectures. For instance, if a kernel is constrained by memory bandwidth on the CPU, transitioning to high-bandwidth memory can alleviate this bottleneck and enhance performance. Once the memory bottleneck is addressed, if the following constraint is FLOPS (core-bound), further speedup might be achievable on processors with higher FLOP rates, such as GPUs.

Our analysis employs the TMA methods on SPR-DDR to assess the memory speed each kernel can reach when running on high bandwidth memory systems like SPR-HBM and GPUs (*e.g.,* NVIDIA V100 and AMD MI250X). Fig. 9 presents a comparative analysis of the performance of RAJAPerf kernels exhibiting memory bound behavior across different architectures.

The figure is divided into four panels. The leftmost panel illustrates the Memory Bound TMA metrics for each RAJAPerf kernel on the SPR-DDR system. The higher the bar, the more constrained the kernel's memory bandwidth. The second panel from the left compares the performance of the same kernels when using the SPR-HBM architecture, which offers higher memory bandwidth. Specifically, the blue bar indicates the speedup achieved by each kernel on SPR-HBM compared to SPR-DDR. The vertical red line marks the 1x performance level, serving as a baseline for comparison. Kernels with blue bars exceeding this red line exhibit speedup, indicating that the transition to HBM alleviates memory bottlenecks for those kernels. Due to the scale of the second panel, we label the kernels with a speedup greater than 1x. The vertical yellow line indicates the value of the Stream_TRIAD kernel for that architecture as another basis for comparison. We chose to highlight Stream_TRIAD because it is significantly memory bound and it contains a higher proportion of bytes read to bytes written, which is characteristic of many scientific applications (see Fig. 1). The two rightmost panels illustrate the kernels' speedup when running on the NVIDIA V100 and AMD EPYC-MI250X GPU architectures compared to the SPR-DDR baseline, respectively. Like the second panel, the blue bars represent the speedup, with the red line

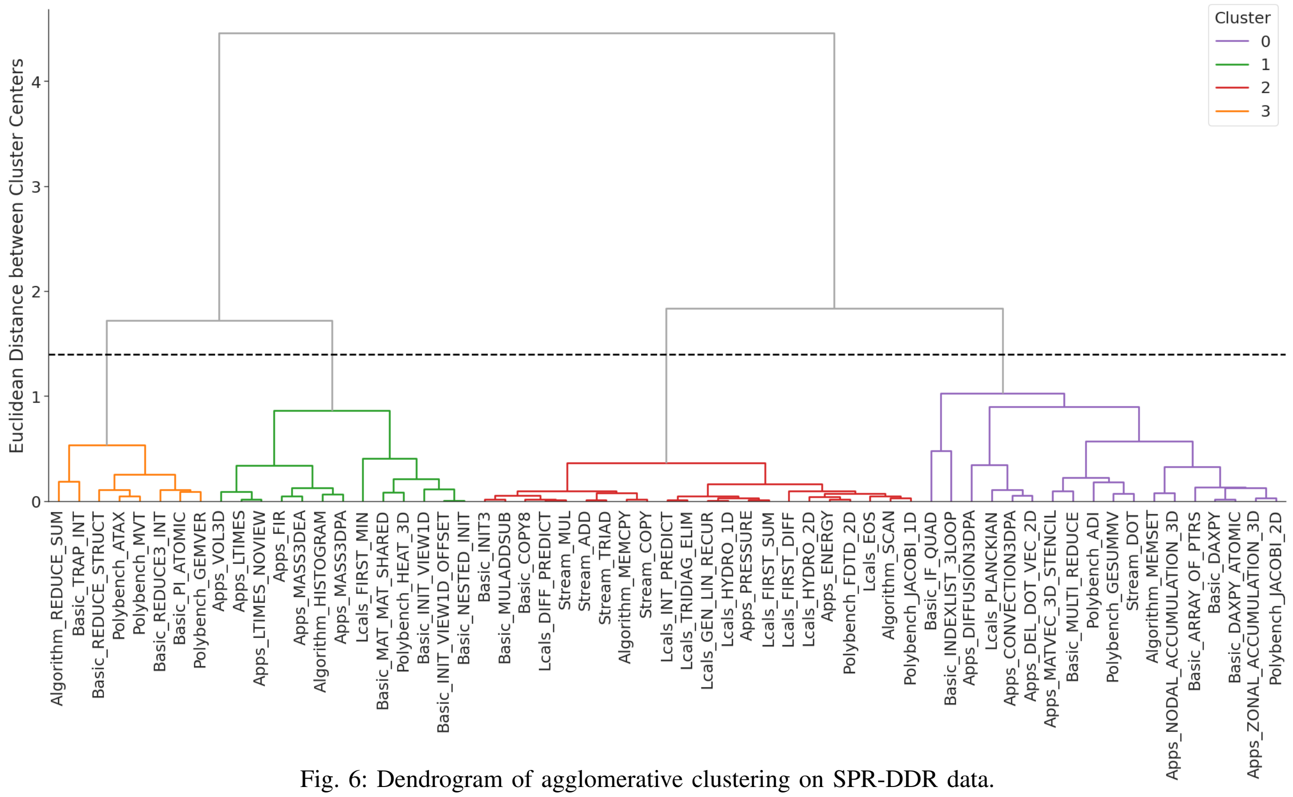
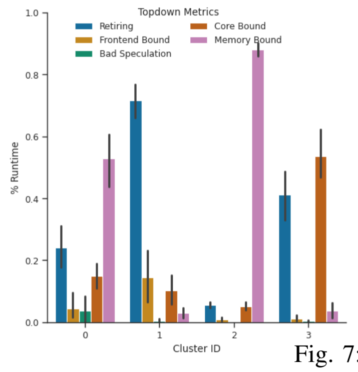


Fig. 6: Dendrogram of agglomerative clustering on SPR-DDR data.



Cluster ID	Algorithms		Applications		Basic		Comm (not		LCALS		Polybench		Stream	
Total Kernels	5	62.5%	14	93%	17	89%	considered)		11	100%	9	69%	5	100%
0	1	12.5%	6	40%	6	32%	0	0%	1	9%	3	23%	1	20%
1	1	12.5%	6	40%	4	21%	0	0%	1	9%	1	7%	0	0%
2	2	25.0%	2	13%	3	16%	0	0%	9	82%	2	15%	4	80%
3	1	12.5%	0	0%	4	21%	0	0%	0	0%	3	23%	0	0%

Cluster ID	Frontend Bound	Bad Speculation	Retiring	Core Bound	Memory Bound	Speedup on SPR-HBM	Speedup on P9-V100	Speedup on EPYC-MI250X
0	0.0452	0.0380	0.2402	0.1488	0.5279	1.4286	4.7197	13.9824
1	0.1460	0.0050	0.7169	0.1021	0.0300	0.9559	4.5510	7.0543
2	0.0103	0.0001	0.0562	0.0522	0.8812	2.5972	7.3578	22.6483
3	0.0118	0.0037	0.4117	0.5358	0.0370	0.8651	3.3596	6.2609

Fig. 7: Per-cluster top-down metrics and speedup over SPR-DDR.

indicating 1x performance, and the yellow line indicating Stream_TRIAD performance. We annotate the speedup for Apps_EDGE3D (118.6) on EPYC-MI250X as it exceeds 40x.

A. Comparing CPU with DDR versus HBM

Our findings support the hypothesis that increased memory bandwidth in memory-constrained kernels enhances their performance. Specifically, the findings confirm that higher memory bandwidth can alleviate bottlenecks, as evidenced by the performance improvement of 40 out of 67 memory bound kernels when transitioning from SPR-DDR to SPR-HBM. The 26 kernels that did not demonstrate substantial performance increases are *less* memory bound than the others. Notable exceptions include Comm_HALO kernels, which are dominated by the MPI communication time.

B. Comparing CPU with DDR versus NVIDIA V100 GPU

When comparing performance from CPUs with DDR memory to the NVIDIA V100 GPU architecture, eight of the kernels did not exhibit speedup on the P9-V100. Contrary to SPR-HBM, where all kernels that demonstrate speedup are at least somewhat memory bound, four kernels—Basic_INIT_VIEW1D, Basic_INIT_VIEW1D_OFFSET, Basic_NESTED_INIT, and Lcals_FIRST_MIN—perform better on the P9-V100 even though they do not exhibit memory constraints. The retiring metric from TMA is the primary bottleneck for the three Basic kernels, indicating they

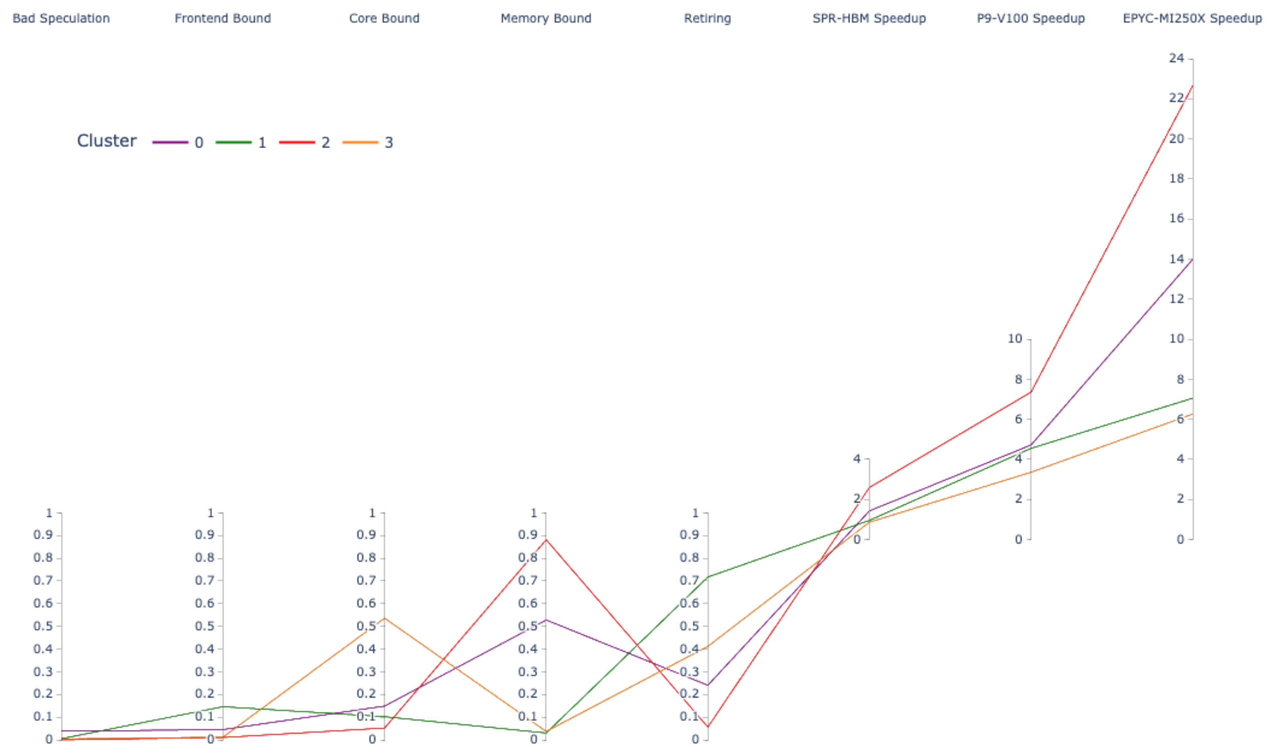


Fig. 8: Parallel coordinate plot showing average top-down metrics and speedup over SPR-DDR per cluster. The average of each of the top-down metrics per cluster is shown in the first five axes and then average speedup per cluster in the last three axes. Cluster 2 (red line), made up of primarily memory bound kernels, exhibits the highest speedup on all architectures.

do not have a specific bottleneck restricting computation. The Lcals kernel is split approximately half and half between retiring and frontend bound. As a result, these kernels can efficiently use the additional parallelism available on the P9-V100. Similarly to SPR-HBM, not all memory bound kernels show speedup on the P9-V100. Besides the Comm_HALO kernels, six other kernels—Basic_PI_ATOMIC, Polybench_ADI, Polybench_ATAX, Polybench_GEMVER, Polybench_GESUMMV, and Polybench_MVT—do not exhibit speedup on the P9-V100. Some of these kernels may not be optimized for GPU execution, which could account for the lack of performance improvement. Other kernels make extensive use of atomics, which are slower than other GPU instructions and introduce synchronization, minimizing the benefit of the additional parallelism available on the P9-V100. Of the memory bound kernels, 11 demonstrate speedup on the P9-V100 but not on SPR-HBM. These include Algorithm_MEMSET, Apps_FIR, Apps_LTIMES, Apps_LTIMES_NOVIEW, Apps_VOL3D, Basic_INIT_VIEW_1D, Basic_INIT_VIEW_1D_OFFSET, Basic_MAT_MAT_SHARED, Polybench_2MM, Polybench_3MM, and Polybench_GEMM. On both SPR-DDR and SPR-HBM, these kernels encounter significant retiring bottlenecks, and the Polybench kernels are also core bound. Thus, these kernels cannot leverage the extra memory bandwidth provided by SPR-HBM alone and instead can benefit from the parallel processing capabilities of the P9-V100. The Polybench kernels here also are $O(N^{3/2})$, so the GPU versions have more work to perform.

C. *Comparing CPU with DDR versus EPYC-MI250X GPU*

On the EPYC-MI250X architecture, almost all of the RAJAPerf kernels demonstrate speedup. The only exceptions are Basic_PI_ATOMIC, Comm_HALO_PACKING, Polybench_ADI, Polybench_ATAX, Polybench_GEMVER, Polybench_GESUMMV, and Polybench_MVT. These kernels all have a non-zero memory bound percentage but also have bottlenecks in both core bound and retiring. They also do not exhibit any speedup on the P9-V100. Two of them, however, show a slight speedup on SPR-HBM: Polybench_ADI and Polybench_GESUMMV. Consequently, the two happen to have the kernels' highest and most significant memory bottlenecks that did not speedup on the EPYC-MI250X (excluding the Comm_HALO kernel, which is an outlier due to the MPI communication and launching many kernels, therefore being kernel launch overhead bound).

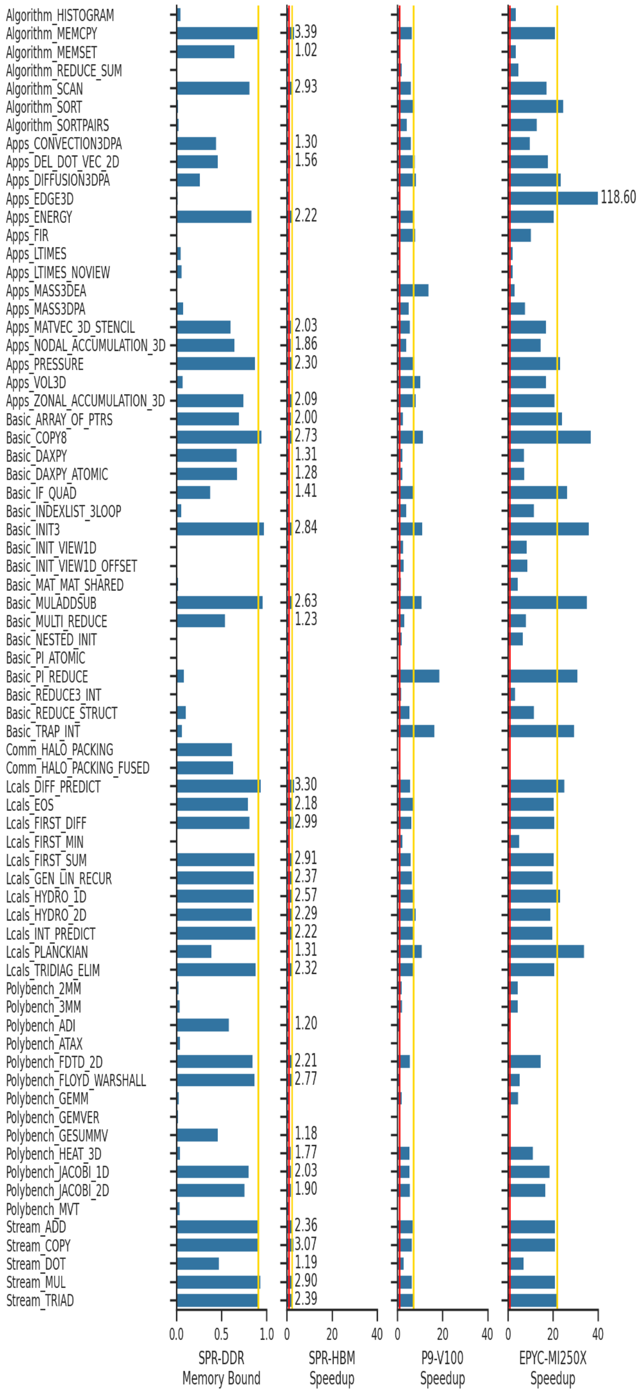


Fig. 9: SPR-DDR memory bound and speedup on SPR-HBM, P9-V100, and EPYC-MI250X relative to SPR-DDR. Red line indicates a speedup of 1; yellow line is the Stream_TRIAD value. All SPR-HBM speedup values greater than 1 are annotated (panel 2). For Apps_EDGE3D on EPYC-MI250X, speedup value is annotated as it exceeds 40x (panel 4).

D. *Comparing CPU with HBM versus GPUs*

When it comes to achievable speedup with GPUs, it helps to take a look at the primarily core (FLOP) bound kernels. Fig. 10 shows whether the RAJAPerf kernels have a higher memory bandwidth (below the diagonal dashed line) or a higher flop rate (above the diagonal dashed line). The 17 FLOP-heavy kernels on SPR-DDR are:

- Apps_CONVECTION3DPA
- Apps_DEL_DOT_VEC_2D
- Apps_DIFFUSION3DPA
- Apps_EDGE3D
- Apps_FIR
- Apps_LTIMES
- Apps_LTIMES_NOVIEW
- Apps_MASS3DPA
- Apps_VOL3D
- Basic_MAT_MAT_SHARED
- Basic_PI_ATOMIC
- Basic_PI_REDUCE
- Basic_TRAP_INT
- Polybench_2MM
- Polybench_3MM
- Polybench_FLOYD_WARSHALL
- Polybench_GEMM

As expected, 15 of these 17 kernels show greater speedup on both the P9-V100 and EPYC-MI250X than on SPR-HBM, as they can take advantage of the additional memory bandwidth. However, the two that do not are Basic_PI_ATOMIC, and Polybench_FLOYD_WARSHALL. Basic_PI_ATOMIC has an extremely high retiring bound, whereas Polybench_FLOYD_WARSHALL is primarily memory bound. Polybench_FLOYD_WARSHALL does exhibit better performance on the EPYC-MI250X than SPR-HBM, but the P9-V100 does not perform as well as SPR-HBM. Polybench_FLOYD_WARSHALL is $O(N^{3/2})$ complexity so the GPU has to do more work due to the decomposition and is not directly comparable to the work performed by the CPU.

Fig. 10 shows that the achievable memory bandwidth increases from SPR-DDR to SPR-HBM, but the FLOP rate stays consistent. This observation contrasts with the P9-V100 in that we see a significant increase in achieved memory bandwidth and a significant boost in achieved FLOPs, especially for some kernels within the Apps, Basic, and Polybench groups. Finally, on the EPYC-MI250X, the memory bandwidth trends towards around 3x of the P9-V100 for many kernels, with an increased FLOP rate as well, though it is not as remarkable a difference.

VI. Related Work

When evaluating new architectures, benchmarks are an important tool to characterize achievable performance. The widely-known LINPACK benchmark [29] is used to categorize systems in the TOP500 list [30]. Similarly, the HPCG benchmark [31] provides additional computation and data access patterns, meant to augment results from LINPACK. While LINPACK and HPCG are used to evaluate performance regardless of architecture, the SHOC [32], NUPAR [33], and Rodinia [34] benchmark suites are focused on evaluating systems that contain GPUs. SHOC and NUPAR contain implementations in OpenCL and CUDA, whereas Rodinia is parallelized with CUDA and OpenMP. The SPEChpc [35] suite contains a variety of MPI, OpenMP, OpenACC, and OpenMP target offload HPC application benchmarks.

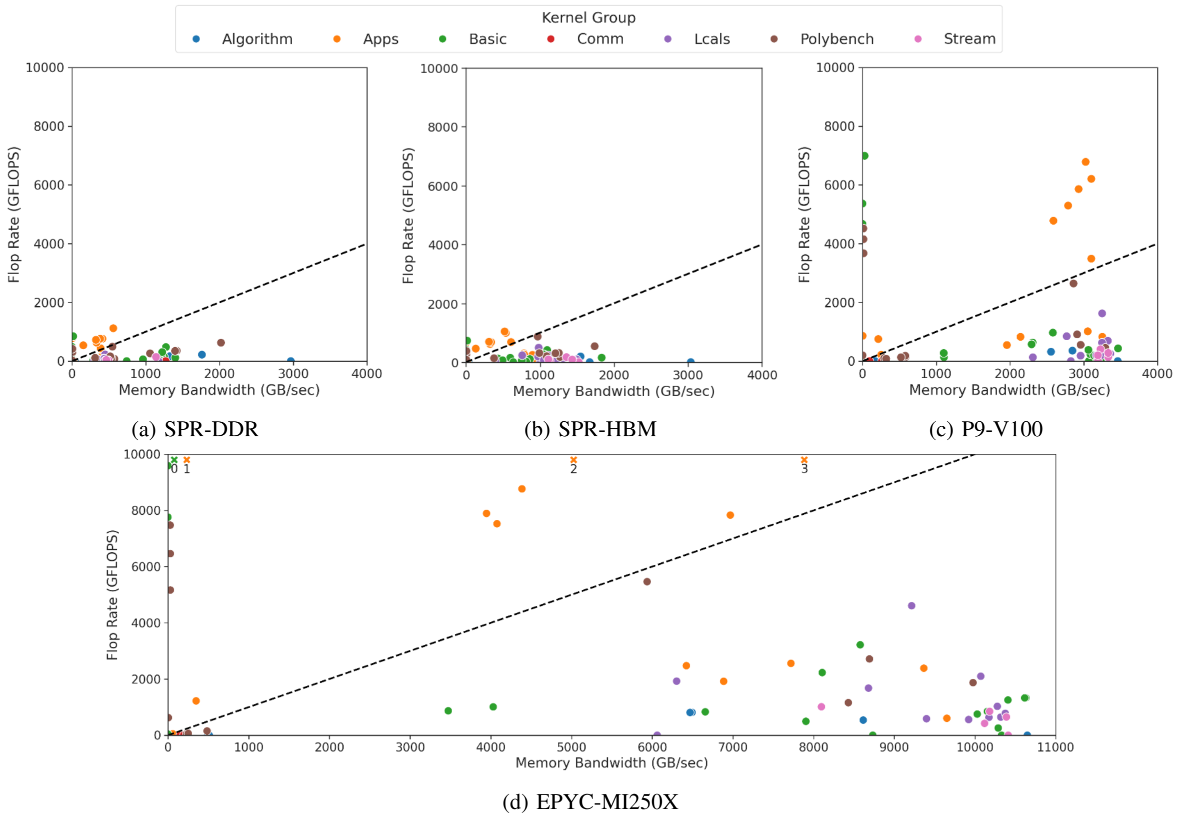


Fig. 10: Memory bandwidth vs. FLOPS on four architectures. Kernels that perform more memory operations than FLOPS fall below the dashed line, kernels that perform more FLOPS are above the dashed line. The four points marked by labeled X symbols at the top of Fig. 10d denote the four kernels with FLOPS greater than 10,000. These kernels are: (0) Basic_MAT_MAT_SHARED (13,326.4 GFLOPS), (1) Apps_EDGE3D (84,113.3 GFLOPS), (2) Apps_VOL3D (11,259.0 GFLOPS), and (3) Apps_DIFFUSION3DPA (14,974.5 GFLOPS).

Kokkos Kernels [36] is a library of portable sparse linear algebra, dense linear algebra, and graph algorithm kernels implemented in the Kokkos programming model, with some changes to the algorithms to increase portability. The primary goal of Kokkos Kernels is to serve as a portable layer to be used within an application as a library, removing the need to actually learn the Kokkos programming model, whereas the RAJA Performance Suite is intended as a standalone benchmark for detailed performance analysis.

Several works conduct benchmarking studies or analyze performance portability across different platforms. In [37], Siefert et al. provide benchmark results for several TOP500 DOE machines using OSU Micro-Benchmarks, BabelStream and Comm|Scope, primarily focusing on memory bandwidth and launch latency. Afzal et al. [38] assess performance and energy characteristics of the Intel Ice Lake and Sapphire Rapids platforms using the SPEChpc suite. In [39], Kwack et al. use roofline models to study the performance portability of RAJA, SYCL, Kokkos, AMReX, and OpenMP codes using three ECP apps and three mini-apps. The results suggest good portability but wide variation in performance portability across the different programming models. In [40], Antepara et al. also use roofline models to evaluate performance portability of block stencil computations across different GPU architectures (NVIDIA, AMD, and Intel) and programming models (CUDA, HIP, and SYCL). In contrast, our study provides detailed comparisons and explanations of performance portability characteristics of the 70+ RAJAPerf kernels across both CPU and GPU platforms.

VII. Conclusions

The RAJA Performance Suite is a performance portable, curated suite of kernels. It enables performance analysis of key computational patterns across modern architectures. This paper presents the analysis of RAJAPerf kernels across four architectures, providing an ability to reason about expected performance on CPUs as well as GPUs. The integration of Caliper into RAJAPerf and analysis with Thicket allowed us to automatically cluster kernels by their execution characteristics and identify their bottlenecks. We further show that the performance of kernels changes as expected when run on an architecture that alleviates the most pressing bottleneck.

Acknowledgements

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 and was supported by the LLNL-LDRD Program under Project No. 24-SI-005 (LLNL-CONF-868342). IL and MT acknowledge the support of NSF #2138811 and #2331152.

References

[1] L. Dagum and R. Menon, "OpenMP: an industry standard API for shared-memory programming," 1998.

[2] D. A. Beckingsale, T. R. Scogland, J. Burmark, R. Hornung, H. Jones, W. Killian, A. J. Kunen, O. Pearce, P. Robinson, and B. S. Ryujin, "RAJA: Portable Performance for Large-Scale Scientific Applications," in *2019 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*. Denver, CO, USA: IEEE, Nov. 2019, pp. 71–81.

[3] LLNL, "RAJA Performance Portability Layer (C++)," http://github.com/LLNL/raja, 2016.

[4] H. C. Edwards, C. R. Trott, and D. Sunderland, "Kokkos: Enabling manycore performance portability through polymorphic memory access patterns," *J. Parallel Distributed Comput.*, vol. 74, pp. 3202–3216, 2014.

[5] C. R. Trott, D. Lebrun-Grandié, D. Arndt, J. Ciesko, V. Q. Dang, N. D. Ellingwood, R. Gayatri, E. Harvey, D. S. Hollman, D. Ibanez, N. Liber, J. Madsen, J. Miles, D. Poliakoff, A. Powell, S. Rajamanickam, M. Simberg, D. Sunderland, B. Turcksin, and J. J. Wilke, "Kokkos 3: Programming model extensions for the exascale era," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, pp. 805–817, 2022.

[6] F. H. McMahon, "Livermore Fortran kernels: A computer test of numerical performance range," UCRL-53724, Tech. Rep., December 1986.

[7] ——, "The Livermore Fortran kernels test of the numerical performance range," *Martin JL (ed) Performance evaluation of supercomputers. Elsevier Science*, p. 143–186, 1988.

[8] LLNL, "CORAL-2 Benchmarks." [Online]. Available: https://asc.llnl.gov/coral-2-benchmarks

[9] K. Asanović, R. Bodík, B. Catanzaro, J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. Plishker, J. Shalf, S. Williams, and K. A. Yelick, "The Landscape of Parallel Computing Research: A View from Berkeley," 2006.

[10] K. Krommydas, W. chun Feng, C. D. Antonopoulos, and N. Bellas, "OpenDwarfs: Characterization of Dwarf-Based Benchmarks on Fixed and Reconfigurable Architectures," *Journal of Signal Processing Systems*, vol. 85, pp. 373 – 392, 2015.

[11] T. Hoefler and R. Belli, "Scientific benchmarking of parallel computing systems: twelve ways to tell the masses when reporting performance results," in *Supercomputing*, 2015.

[12] D. Jacobsen and B. Bird, "Ramble: A flexible, extensible, and composable experimentation framework," in *HPC Tests Workshop at the ACM/IEEE International Conference on High Performance Computing, Network, Storage, and Analysis*. Denver, CO, USA: ACM, Nov. 2023.

[13] O. Pearce, A. Scott, G. Becker, R. Haque, N. Hanford, S. Brink, D. Jacobsen, H. Poxon, J. Domke, and T. Gamblin, "Towards Collaborative Continuous Benchmarking for HPC." New York, NY, USA: Association for Computing Machinery, 2023.

[14] D. Boehme, T. Gamblin, D. Beckingsale, P.-T. Bremer, A. Gimenez, M. LeGendre, O. Pearce, and M. Schulz, "Caliper: Performance Introspection for HPC Software Stacks," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '16. IEEE Press, 2016.

[15] D. Boehme, P. Aschwanden, O. Pearce, K. Weiss, and M. LeGendre, "Ubiquitous Performance Analysis," in *High Performance Computing*, B. L. Chamberlain, A.-L. Varbanescu, H. Ltaief, and P. Luszczek, Eds. Cham: Springer International Publishing, 2021, pp. 431–449.

[16] "NVIDIA Nsight Compute Profiling Tool," https://docs.nvidia.com/nsight-compute/NsightCompute/index.html.

[17] S. Brink, M. McKinsey, D. Boehme, C. Scully-Allison, I. Lumsden, D. Hawkins, J. Lüttgau, K. Isaacs, M. Taufer, and O. Pearce, "Thicket: Seeing the Performance Experiment Forest for the Individual Run Trees," in *32nd Intl Symp on High-Performance Parallel and Distr Computing*, June 2023.

[18] A. Yasin, "A Top-Down Method for Performance Analysis and Counters Architecture," in *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. CA, USA: IEEE, Mar. 2014, pp. 35–44.

[19] N. Ding and S. Williams, "An instruction roofline model for gpus," in *2019 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, 2019, pp. 7–18.

[20] LLNL, "RAJA Performance Suite," http://github.com/LLNL/RAJAPerf, 2017.

[21] L.-N. Pouchet, "The Polyhedral Benchmark Suite," https://web.cs.ucla.edu/~pouchet/software/polybench/, 2010.

[22] J. D. McCalpin, "Stream: Sustainable memory bandwidth in high performance computers," University of Virginia, Charlottesville, Virginia, Tech. Rep., 1991-2007, a continually updated technical report. http://www.cs.virginia.edu/stream/. [Online]. Available: http://www.cs.virginia.edu/stream/

[23] LLNL, "Caliper," https://github.com/llnl/caliper, 2017.

[24] ——, "Thicket," https://github.com/llnl/thicket, 2023.

[25] ——, "Adiak," https://github.com/LLNL/Adiak, 2019.

[26] S. Browne, J. Dongarra, N. Garner, G. Ho, and P. Mucci, "A Portable Programming Interface for Performance Evaluation on Modern Processors," vol. 14, no. 3, 2000. [Online]. Available: https://doi.org/10.1177/109434200001400303

[27] N. Ding, M. Awan, and S. Williams, "Instruction roofline: An insightful visual performance model for gpus," *Concurrency and Computation: Practice and Experience*, vol. 34, no. 20, p. e6591.

[28] J. H. Ward, "Hierarchical grouping to optimize an objective function," *Journal of the American Statistical Association*, vol. 58, no. 301, pp. 236–244, 1963. [Online]. Available: http://www.jstor.org/stable/2282967

[29] J. J. Dongarra, P. Luszczek, and A. Petitet, "The linpack benchmark: past, present and future," *Concurrency and Computation: Practice and Experience*, vol. 15, 2003.

[30] "Top 500 List," https://top500.org.

[31] J. J. Dongarra, M. Heroux, and P. Luszczek, "High-performance conjugate-gradient benchmark: A new metric for ranking high-performance computing systems," *The International Journal of High Performance Computing Applications*, vol. 30, pp. 10 – 3, 2016.

[32] A. Danalis, G. Marin, C. McCurdy, J. S. Meredith, P. C. Roth, K. Spafford, V. Tipparaju, and J. S. Vetter, "The Scalable Heterogeneous Computing (SHOC) benchmark suite," in *GPGPU-3*, 2010.

[33] Y. Ukidave, F. N. Paravecino, L. Yu, C. Kalra, A. Momeni, Z. Chen, N. Materise, B. Daley, P. Mistry, and D. R. Kaeli, "Nupar: A benchmark suite for modern gpu architectures," *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*, 2015.

[34] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," *2009 IEEE International Symposium on Workload Characterization (IISWC)*, pp. 44–54, 2009.

[35] S. P. E. Corporation, "SPEChpc 2021 Benchmark Suite," https://www.spec.org/hpc2021/, 2021.

[36] S. Rajamanickam, S. Acer, L. Berger-Vergiat, V. Q. Dang, N. D. Ellingwood, E. Harvey, B. Kelley, C. R. Trott, J. J. Wilke, and I. Yamazaki, "Kokkos Kernels: Performance Portable Sparse/Dense Linear Algebra and Graph Kernels," *ArXiv*, vol. abs/2103.11991, 2021.

[37] C. M. Siefert, C. Pearson, S. L. Olivier, A. Prokopenko, J. Hu, and T. J. Fuller, "Latency and Bandwidth Microbenchmarks of US Department of Energy Systems in the June 2023 Top 500 List." New York, NY, USA: Association for Computing Machinery, 2023.

[38] A. Afzal, G. Hager, and G. Wellein, "SPEChpc 2021 Benchmarks on Ice Lake and Sapphire Rapids Infiniband Clusters: A Performance and Energy Case Study." New York, NY, USA: Association for Computing Machinery, 2023.

[39] J. Kwack, J. R. Tramm, C. Bertoni, Y. Ghadar, B. Homerding, E. Rangel, C. Knight, and S. Parker, "Evaluation of performance portability of applications and mini-apps across amd, intel and nvidia gpus," *2021 International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*, pp. 45–56, 2021.

[40] O. Antepara, S. Williams, H. Johansen, T. Zhao, S. Hirsch, P. Goyal, and M. Hall, "Performance portability evaluation of blocked stencil computations on gpus." New York, NY, USA: Association for Computing Machinery, 2023.