Exploring the Performance Portability of Modern Programming Languages

Brett Eiffert, Mohammad Alaul Haque Monil, Narasinga Rao Miniskar,
William Godoy, Keita Teranishi, Jeffrey S. Vetter
Oak Ridge National Laboratory, Oak Ridge, TN, USA (eiffertbc@ornl.gov)

I. Abstract

This poster evaluates the performance portability of four modern programming languages on heterogeneous systems. Julia, Mojo, Python, and Rust were designed for different purposes, but each supports portable kernel development for high-performance computing (HPC) workloads. Julia was designed for scientific workflows, Python as a productive general-purpose language, Rust as a systems-level and memory-safe language, and Mojo for performance portability. With the exception of Mojo, the communities of the other three languages developed performance portable additions for the languages as they matured: KernelAbstractions.jl for Julia, Triton for Python, and CubeCL for Rust. Using four scientific benchmarks that stress different aspects of performance, this study compares these approaches on modern NVIDIA and AMD hardware.

Starting from NVIDIA CUDA and AMD HIP implementations of the seven-point stencil [1], BabelStream [2], Hartree–Fock [3], and miniBUDE [4] benchmarks, each benchmark was ported to the four target languages. Seven-point stencil updates grid points using neighboring points, BabelStream measures attainable memory bandwidth, Hartree–Fock approximates molecular electronic structure using quantum chemistry calculations, and miniBUDE computationally scores molecular docking poses relevant to drug discovery. All four benchmarks were validated for correctness using error metrics within defined bounds. Each benchmark was evaluated across different configurations to provide a fuller picture of where each target excels or struggles. Experiments were run on the NVIDIA H100 NVL and AMD MI300A in both single and double precision.

Porting native implementations to these targets was not straightforward. Each language differed in its strengths, weaknesses, compiler design, and programming models. In this experiment, an idiomatic approach was taken. To produce comparable benchmarks, tradeoffs were necessary to preserve this intent. Attributes such as the size and shape of compute resources were kept consistent across ports, while other parameters were allowed to freely reflect language and programming model design choices. No language was forced to behave like another, and some decisions were left to the compiler. Compilation methods also differed, with some systems using ahead-of-time compilation, others just-in-time compilation, and some combining both approaches. For example, Triton uses a compiler-controlled tile-centric model, whereas CUDA exposes threads, warps, and blocks more directly. Also, some compilers provide built-in fusion of mathematical operations while others do not. These idiomatic differences appeared across all ports, yet the results remained coherent.

Many distinct findings emerged across the four benchmarks. The seven-point stencil showed KernelAbstractions.jl performed poorly on memory-bound benchmarks, but performed well on compute-bound AMD implementations. BabelStream showed that Triton's tile-centric model optimized particularly well for single-precision memory operations. Hartree–Fock indicated that Mojo is immature in its atomic-operation capabilities. MiniBUDE demonstrated Mojo performing well on supported precisions, but also a lack of support for 64-bit trigonometric functions on both NVIDIA and AMD hardware. CubeCL performed consistently acceptable across all benchmarks.

This study compares languages at varying levels of maturity against native implementations. CUDA and ROCm are mature vendor platforms, while the portable alternatives vary in development status. Given the limited capabilities and experimental nature of these HPC approaches, the effort required to run all implementations in comparable configurations was substantial. Their long-term futures are also uncertain. Some may improve significantly, while others may not be maintained over time.

Performance portability across heterogeneous HPC systems remains a complex balance of usability, abstraction, and hardware-specific optimization. The results for KernelAbstractions.jl, Triton, Mojo, and CubeCL show that higher-level abstractions can enable productive cross-platform development, but may also introduce constraints that limit performance relative to native implementations. Differences in language and compiler design prevent direct equivalence between ports, making idiomatic implementations a viable approach for a fair comparison. Although these languages are constantly changing and improving, their ability to run scientific workloads on both NVIDIA and AMD hardware is promising. Continued development and long-term maintenance will determine whether they can become and stay practical alternatives for performance portable high-performance computing.

***Index Terms*—Performance Portability, Productivity, Julia, Mojo, Triton, Rust**

ACKNOWLEDGMENT

This work was supported by the UT-Battelle LLC through U.S. Department of Energy (DOE) under Contract DE-AC05-00OR22725. This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research Program, through "Competitive Portfolios" DE-FOA-0003264, under award number DE-SC0025645.

This research used the same benchmarks as a 2025 paper [5] on the Mojo programming language.

This research used resources of the Experimental Computing Laboratory (ExCL) at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

The authors used generative AI and AI-assisted tools, including frontier large language models, to support background research, literature discovery, text editing, data analysis, code development, and the preparation of graphs and figures. All AI-assisted outputs were reviewed, verified, edited, and integrated by the authors, who take full responsibility for the accuracy, originality, interpretation, and integrity of the final work.

REFERENCES

[1] J. Chang, R. Arora, T. H. Gibson, S. Miller, and O. O'Reilly, "Finite difference method—Laplacian part 1," AMD Lab Notes, GPUOpen, 2022, accessed: Sep. 13, 2025. [Online]. Available: https://gpuopen.com/learn/amd-lab-notes/amd-lab-notes-finite-difference-docs-laplacian_part1/

[2] T. Deakin, J. Price, M. Martineau, and S. McIntosh-Smith, "Evaluating attainable memory bandwidth of parallel programming models via BabelStream," *Int. J. Comput. Sci. Eng.*, vol. 17, no. 3, pp. 247–262, 2018.

[3] Argonne National Laboratory, "Basic Hartree-Fock proxy application," ECP Proxy Applications, 2025, accessed: Sep. 28, 2025. [Online]. Available: https://proxyapps.exascaleproject.org/app/basic-hartree-fock-proxy-application/

[4] A. Poenaru, W.-C. Lin, and S. McIntosh-Smith, "A performance analysis of modern parallel programming models using a compute-bound application," in *Proc. 36th Int. Conf. High Perform. Comput. (ISC High Performance)*, ser. Lecture Notes in Computer Science, vol. 12728. Cham, Switzerland: Springer, 2021, pp. 332–350.

[5] W. Godoy, T. Melnichenko, P. Valero-Lara, W. Elwasif, P. Fackler, R. Ferreira Da Silva, K. Teranishi, and J. Vetter, "Mojo: Mlir-based performance-portable hpc science kernels on gpus for the python ecosystem," in *Proceedings of the SC '25 Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC Workshops '25. ACM, Nov. 2025, p. 2114–2128. [Online]. Available: http://dx.doi.org/10.1145/3731599.3767573