JACC: Easy CPU/GPU Performance Portability for Scientific Applications in Julia

William F. Godoy, Pedro Valero-Lara, Philip W. Fackler, Keita Teranishi, Jeffrey S. Vetter
Oak Ridge National Laboratory
Oak Ridge, TN, USA
{godoywf,valerolarap,facklerpw,teranishik,vetter}@ornl.gov

Jhonny Gonzalez, Jose Gonzalez
The University of Texas at El Paso
El Paso, TN, USA
{jgonzalez183,jvgonzalez6}@miners.utep.edu

Alexis Huante
The University of Texas at Austin
Austin, TX, USA
alexishuante@utexas.edu

ABSTRACT

Significant investments have been made to close the performance gap in high-productivity languages such as Python, Julia, R, and Matlab. However, the challenge of writing code once that performs efficiently across CPUs and GPUs is made even greater by architectural heterogeneity. We present the latest version of JACC [2][1] (Julia for ACCelerators), a performance portable CPU/GPU library implementation for the scientific LLVM-based scientific Julia programming language. JACC provides a unified and lightweight front end across different backends available in the Julia ecosystem[2], enabling the same Julia code to run efficiently on many CPU and GPU targets. Thus JACC leverages existing community investments while ensuring interoperability with the Julia ecosystem. JACC's goal to boost users' productivity, since the same performant code can run on multiple architectures allows for more time to be dedicated to science rather than code development and maintenance.

Our poster for SC25 presents a completely updated version of the JACC best poster finalist at SC24, showcasing the latest added features in the JACC library for productive scientific computing. First, we describe the new and stable JACC API main components: (i) portable memory allocation for multidimensional arrays and shared memory, (ii) kernel launching via the `parallel_for` and `parallel_reduce` functions using the basic and advanced APIs allowing for defaults and launch specification control, respectively, and (iii) back end selection without code changes using Julia's widely adopted Preferences.jl package. JACC's architecture is then described showing how it leverages five back ends (i) Julia's Threads on CPUs; and (ii) CUDA.jl on NVIDIA, (iii) AMDGPU.jl on AMD, (iv) OneAPI.jl on Intel and (v) Metal.jl on Apple M-series GPUs. Second, we present two newly added features: (i) JACC's shared, for exploiting cached shared memory among threads, and (ii) JACC's Multi module, for programming multiple GPUs in a vendor agnostic way. These options allow the portable programmability of future supercomputing nodes with an increasing number of GPUs. We show the performance gained in the common AI-foundational convolution operation exploiting GPU's block-shared memory on Perlmutter's NVIDIA A100 GPUs and Frontier's AMD MI250X. Third, we present preliminary results comparing JACC's performance with scientific kernels with available vendor and performance portable models.

JACC application workloads ports are provided for several representative science application workloads[3]: XSBench, miniBUDE, LULESH, BabelStream, and Hartree–Fock. We showcase preliminary results comparing the portable JACC implementations on XSBench (memory-bound), miniBUDE (compute-bound) and LULESH (memory and compute-bound) on NVIDIA's A100 and H100, and AMD's MI100 and MI250X (Frontier's) GPUs against available programming model implementations on C++. JACC's XSBench performance is competitive with other programming models on AMD, while gaps still exist on NVIDIA GPUs. Results on miniBUDE reveal that JACC is competitive on NVIDIA and GPUs with existing gaps on Intel Arc A770 GPUs - only supporting single precision, providing a glimpse into JACC's portability. Results on LULESH suggest that there is considerable overhead when launching small load kernels, as opposed to Kokkos' C++codes. Hence, we showed that JACC is competitive on a wide variety of workloads and can achieve performance similar to that of OpenMP, Kokkos, OpenCL, SYCL, and vendor-specific CUDA and HIP baseline implementations on C++, C, and Fortran for several memory-bound workloads, though gaps still need to be understood for specific cases.

Our comprehensive work shows that JACC is a strong paradigm for the Julia for HPC [1] ecosystem. JACC and Julia allow developing and running complex performance-portable codes for scientific applications and exploratory work at a fraction of the cost. We still recognize that due to its novelty, JACC must still be evaluated across multidisciplinary science domains and heterogeneous hardware architectures.

KEYWORDS

JACC, Julia, HPC, scientific computing, performance portability, high-productivity languages

[1] https://github.com/JuliaORNL/JACC.jl
[2] https://juliagpu.org

[3] https://github.com/JuliaORNL/JACC-applications

SC 2025, November 17–22, 2025, St. Louis, MO, USA

ACM ISBN XXXXX…$15.00
https://doi.org/10.XXXX/XXXX.XXXX

ACM Reference Format:
William F. Godoy, Pedro Valero-Lara, Philip W. Fackler, Keita Teranishi, Jeffrey S. Vetter, Jhonny Gonzalez,, Jose Gonzalez, and Alexis Huante. 2025. JACC: Easy CPU/GPU Performance Portability for Scientific Applications in Julia. In *The International Conference on High Performance Computing, Network, Storage, and Analysis (SC 2025), November 17–22, 2025, St.Louis, MO, USA.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.XXXX/XXXX.XXXX

ACKNOWLEDGMENTS

This work was supported in part by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research's Computer Science Competitive Portfolios, MAGMA/-Fairbanks project, and the Next Generation of Scientific Software Technologies, PESO and S4PST projects, programs. This research used resources of the Oak Ridge Leadership Computing Facility and the Experimental Computing Laboratory (ExCL) at the Oak Ridge National Laboratory, which is supported by the Office of Science of the US Department of Energy under Contract No. DE-AC05-00OR22725. The authors would like to thank the CERFACS Computational Fluid Dynamics (CFD) team led by Jean-François Boussuge for their early feedback on JACC.

REFERENCES

[1] Valentin Churavy et al. 2022. Bridging HPC Communities through the Julia Programming Language. arXiv:2211.02740 [cs.DC]
[2] Pedro Valero-Lara et al. 2024. JACC: Leveraging HPC Meta-Programming and Performance Portability with the Just-in-Time and LLVM-based Julia Language. In *SC24-W: Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1955–1966. https://doi.org/10.1109/SCW63240.2024.00245