Comparative Overview of HPC Frameworks for CPU/GPU Programming

Ivan Smirnov, Vladislav Veselov, Maryana Smirnova, Markus Rampp

Deggendorf Institute of Technology   Max Planck Computing and Data Facility

DEGGENDORF INSTITUTE OF TECHNOLOGY

MAX PLANCK COMPUTING & DATA FACILITY MPCDF

Introduction

Modern High-Performance Computing (HPC) relies on a wide range of hardware, including CPUs, GPUs, and accelerators. While the Message Passing Interface (MPI) remains the standard for distributed-memory computing, developers must choose from an increasing number of node-level programming models, such as OpenMP, CUDA, HIP, OpenACC, oneAPI (SYCL), Kokkos, ALPAKA and RAJA. These frameworks vary in their ability to deliver portability, performance, and ease of use, making it essential to carefully evaluate their features.

This poster compares these programming models based on key factors such as *functional portability* (the ability to run code across different architectures with a focus on TOP500 List), *performance portability* (maintaining efficiency across platforms), *ecosystem maturity* (tooling, libraries, and community support), and *use cases*. Drawing from published studies, benchmarks, and real-world applications, we classify each framework's strengths, limitations, and trade-offs. This analysis aims to provide scientific code developers technical criteria and references to guide the selection of the best-suited framework.

Method

This evaluation systematically analyzes nine prominent parallel programming frameworks in HPC: OpenMP, OpenACC, CUDA, RAJA, Kokkos, ALPAKA, HIP, SYCL and OpenCL. The analysis is based on literature and published results. This evaluation extends work [22] and considers the following criteria:

- *Primary Model* — Frameworks are classified by their parallelization approach (e.g., shared memory, host-device or abstraction-based).
- *Target Hardware* — Compatibility with CPUs, GPUs, FPGAs and hybrid systems.
- *Functional Portability* — The ability to execute code across vendor platforms with minimal modification.
- *Performance Portability* — The capability to maintain efficient performance across diverse hardware with varying levels of tuning.
- *Ecosystem Maturity* — Tool availability, community activity and quality of documentation.
- *Use Cases* — Applicability to specific HPC domains such as scientific simulations and AI/ML.

Evaluation Approach

Functional Portability: Measures how easily code runs across platforms.
- High (green): Supports 3+ vendors with minimal code changes.
- Medium (yellow): Supports 2 vendors, moderate adaptations required.
- Low (red): Vendor-specific, significant rewrites needed.

Performance Portability: Assesses how consistently frameworks achieve high performance.
- High (green): Strong performance across CPUs and GPUs with little tuning.
- Medium (yellow): Good performance on one platform, acceptable on others with moderate tuning.
- Low (red): Optimized for one platform only, requiring extensive reimplementation.

Ecosystem Maturity: Evaluates tools, community support, and documentation.
- High (green): Comprehensive tools, active community, high-quality documentation.
- Medium (yellow): Adequate tools, moderate community, sufficient documentation.
- Low (red): Limited tools, niche adoption, outdated or minimal documentation.

Conclusion

This work reveals the following key observations regarding GPU-focused programming models and complementary CPU-based paradigms:

- Hardware-Specific Approaches (for example, CUDA for NVIDIA, HIP for AMD, and oneAPI for Intel) typically naturally achieve the best performance on their native architectures but may increase maintenance complexity when porting to alternative hardware.
- Directive-Based Methods (for example, OpenMP and OpenACC) offer convenient multi-vendor support, but performance may lag behind platform-native solutions.
- Abstraction Layers (for example, Kokkos and RAJA) provide single-source development for multiple platforms, helping manage code complexity. Nonetheless, consistent performance across different architectures depends on the maturity of underlying compilers and runtimes.

Selecting the best-suited framework involves balancing immediate performance needs against longer-term sustainability. Although CUDA remains dominant in many NVIDIA-based environments, advanced solutions from AMD, Intel, and high-level abstractions like Kokkos continue to expand the possibilities for portable HPC development. Future studies could further explore the role of emerging frameworks in large-scale applications and evaluate their performance across a wider range of accelerators. This poster supports HPC researchers and developers in navigating this complex landscape by classifying various frameworks based on published results and comparisons.

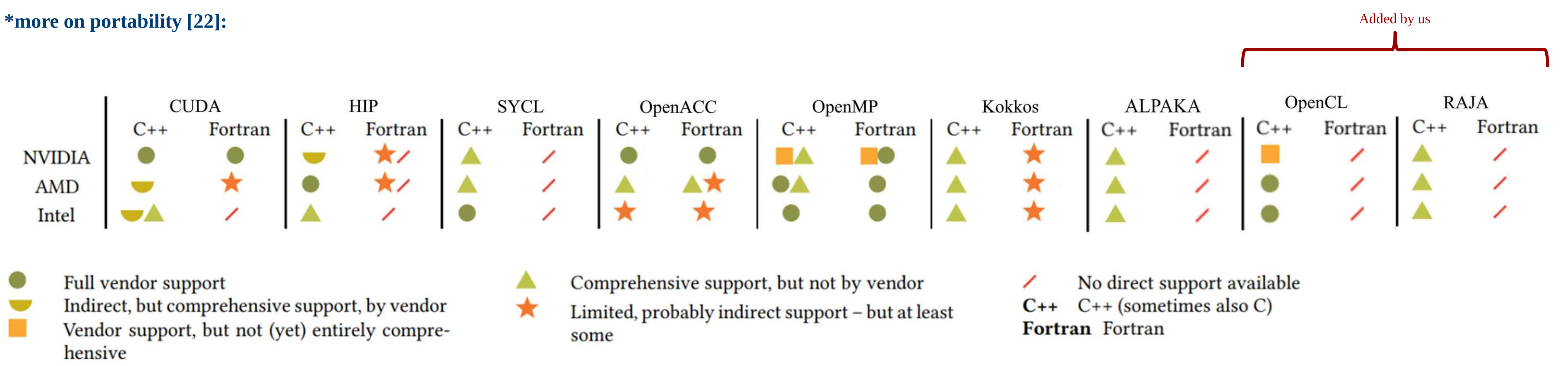
Results & Discussion

Frame-work	Primary Model	Target Hardware	Functional portability*	Performance Portability	Ecosystem Maturity	Use Cases
OpenCL	Cross-platform, kernel-based, host-device(with OpenCL C use)	CPU, GPU, FPGA, DSP	(⏳) Was created as cross-vendor [1]	Varies significantly across platforms [3], and even in convenient single-node cases is 1.3 times slower than CUDA [2]	Never gained much traction in the HPC-GPU space, mostly due to the lukewarm support by NVIDIA [22]	Cross-vendor HPC, embedded systems, AI/ML and scientific computing
SYCL	Cross-platform, single-source C++	CPU, GPU	implementations are available from an increasing number of vendors, including adding support for diverse acceleration API back-ends in addition to OpenCL: Intel oneAPI, AdaptiveCpp, triSYCL, neoSYCL, SimSYCL [4]	It is high on NVIDIA and Intel GPU, but limited on CPU [5]	(❓) Growing tooling and libraries through Intel oneAPI; still developing maturity compared to CUDA [6]	HPC, scientific computing, AI/ML and data-parallel tasks
RAJA	Abstraction layer, loop-level parallelism (multi-backend)	CPU, GPU	Vendor interactions to support new hardware from IBM, NVIDIA, AMD, Intel, and Cray [7]	It is high on NVIDIA GPU, but limited on AMD GPU [8]	(❓) Well-supported within DOE but slightly less comprehensive than Kokkos [9]	Scientific simulations, multi-backend HPC and loop management, also performance-portable HPC applications at LLNL
Kokkos	Abstraction layer, parallel execution and memory management (multi-backend)	CPU, GPU (NVIDIA, AMD, Intel)	(⏳) Provides backend switching between OpenMP, CUDA, and HIP for portability across vendors [12]	Achieves close-to-native performance with tuning [12] [13]	Strong DOE backing, integrated with major HPC libraries like Trilinos [14]	HPC simulations, computational science, fine-grained parallelism and performance-portable C++ applications [13]
ALPAKA	Abstraction layer, fine-grained parallelism (multi-backend)	CPU, GPU (NVIDIA, AMD, Intel), FPGA	Multi-backend support with backend switching between frameworks like CUDA, OpenMP, and HIP [26]	Achieves close-to-native performance but requires tuning; evaluated as performance portable across HPC architectures [24][25]	Smaller ecosystem compared to Kokkos; tools and libraries still maturing but actively used in research [25]	HPC simulations, cross-platform performance-portable applications, and fine-grained parallelism tasks [26]
Open ACC	Directive-based, host-device (focused on GPU offloading)	NVIDIA and AMD GPUs	(❓) Supports multi-vendor systems but favors NVIDIA GPUs due to more mature implementations	(⏳) Performance depends heavily on compiler quality and vendor support [10], [11]	(❓) Limited tools and libraries, mostly focused on legacy projects [11]	Climate modeling, GPSU-accelerated legacy applications [10]
OpenMP	Directive-based, shared memory (with GPU offloading support)	CPU, GPU	Vendor-neutral [17], [18]	Tuning required for GPUs [19], [8]	Robust tools, broad adoption, and active vendor/community support [17]	Shared-memory HPC, engineering simulations, hybrid AI/ML [20]
CUDA	Hardware-specific, kernel-based, host-device	NVIDIA GPUs	Only NVIDIA hardware [15]	Performance portability across vendors is non-existent, but high across different NVIDIA GPU models and generations[15], [16]	Extensive libraries (cuBLAS, cuDNN), industry-standard tools (Nsight), strong NVIDIA support [16]	GPU-accelerated AI/ML, scientific simulations, rendering[16]
HIP	Hardware-specific, kernel-based, host-device (CUDA-like)	AMD GPUs	Portable for AMD and convertible CUDA applications with HIPIFY [21]	Optimized for AMD, tuning required for other vendors [8], [21]	AMD-focused tools and libraries, still maturing	AMD-targeted HPC, AI/ML and engineering simulations

Table. Confidence indicators:

- Question mark (❓) — uncertainty due to limited data or conflicting studies
- Hourglass (⏳) — information older than 10 years, potential obsolescence

***more on portability [22]:**



References

1. Stone, J. E., Gohara, D., & Shi, G. (2010). OpenCL: A parallel programming standard for heterogeneous computing systems. *Computational Science and Engineering, 12*(3), 66–72.
2. Pennycook, S. J., Hammond, S. D., Wright, S. A., Herdman, J. A., Miller, I., & Jarvis, S. A. (2013). An investigation of the performance portability of OpenCL. *Journal of Parallel and Distributed Computing*, *73*(11), 1439–1450. https://doi.org/10.1016/j.jpdc.2012.07.005
3. Bertoni, C., Kwack, J., Applencourt, T., Ghadar, Y., Homerding, B., Knight, C., Videau, B., Zheng, H., Morozov, V., & Parker, S. (2020). Performance portability evaluation of OpenCL benchmarks across Intel and NVIDIA platforms. *Argonne National Laboratory.* Retrieved from https://www.anl.gov
4. The Khronos Group. (n.d.). SYCL. Retrieved from https://www.khronos.org/sycl/
5. Reguly, I. Z. (2023). Evaluating the performance portability of SYCL across CPUs and GPUs on bandwidth-bound applications. *Workshops of the International Conference on High Performance Computing, Network, Storage, and Analysis (SC-W 2023), Denver, CO, USA, November 12–17, 2023.*
6. HPCwire. (2023, February 28). State of SYCL: ECP BoF showcases progress and performance. Retrieved from https://www.hpcwire.com/2023/02/28/state-of-sycl-ecp-bof-showcases-progress-and-performance/
7. Lawrence Livermore National Laboratory. (n.d.). RAJA Portability Suite: Enabling performance portable CPU and GPU HPC applications. Retrieved from https://computing.llnl.gov/projects/raja-managing-application-portability-next-generation-platforms
8. Davis, J. H., Sivaraman, P., Minn, I., Parasyris, K., Menon, H., Georgakoudis, G., & Bhatele, A. (2023). An evaluative comparison of performance portability across GPU programming models. *Department of Computer Science, University of Maryland, & Lawrence Livermore National Laboratory.*
9. RAJA Documentation. (n.d.). Retrieved from https://raja.readthedocs.io/en/develop/
10. Sabne, A., Sakdhnagool, P., Lee, S., & Vetter, J. S. (2014). Evaluating performance portability of OpenACC. In *Languages and Compilers for Parallel Computing* (pp. 63–77). Springer.
11. Deakin, T., et al. (2019). Performance portability across diverse computer architectures. *2019 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC), Denver, CO, USA, 2019*, pp. 1–13. https://doi.org/10.1109/P3HPC49587.2019.00006
12. Edwards, H. C., Trott, C. R., & Sunderland, D. (2014). Kokkos: Enabling manycore performance portability through polymorphic memory access patterns. *Journal of Parallel and Distributed Computing*, *74*(12), 3202–3216.
13. Kokkos Documentation. (n.d.). Available at https://kokkos.org/kokkos-core-wiki/
14. Kokkos Abstract. (2023). Available at https://kokkos.org/about/abstract/
15. NVIDIA Developer. (n.d.). CUDA Toolkit Documentation. Retrieved from https://docs.nvidia.com/cuda/
16. NVIDIA Developer. (n.d.). CUDA Zone. Retrieved from https://developer.nvidia.com/cuda-zone
17. OpenMP. (n.d.). *OpenMP (Open Multi-Processing)*. Retrieved from https://en.wikipedia.org/wiki/OpenMP
18. OpenMP Architecture Review Board. (n.d.). *OpenMP Compilers & Tools*. Retrieved from https://www.openmp.org/resources/openmp-compilers-tools/
19. Malik, D. (2022). *Performance Portability of OpenMP*. Technical University of Munich. Retrieved from https://events.gwdg.de/event/243/contributions/503/attachments/139/174/OpenMP.Pd
20. van Waveren, M. (2020). *OpenMP Use Cases*. OpenMP ARB & CS GROUP. Retrieved from https://www.openmp.org/wp-content/uploads/OpenMP-Use-Cases-vanWaveren.pdf
21. AMD. (n.d.). *HIP Documentation: Performance Portability for Heterogeneous Systems*. Retrieved from https://rocm.docs.amd.com/projects/HIP/en/latest/index.html
22. Herten, A. (2023). Many cores, many models: GPU programming model vs. vendor compatibility overview. Proceedings of the P3HPC Workshop, hosted at SC23 (International Conference for High Performance Computing, Networking, Storage, and Analysis). Retrieved from https://doi.org/10.48550/arXiv.2309.05445
23. Zenker, E., Worpitz, B., Widera, R., Huebl, A., Juckeland, G., Knupfer, A., Nagel, W. E., & Bussmann, M. (2016). Alpaka: An Abstraction Library for Parallel Kernel Acceleration. arXiv:1602.08477. Available at https://arxiv.org/abs/1602.08477
24. Stephan, J., Bastrakov, S., Di Pilato, A., Ehrig, S., Gruber, B. M., Vyskočil, J., Widera, R., & Bussmann, M. (n.d.). Performance portability with alpaka. Helmholtz-Zentrum Dresden-Rossendorf (HZDR).
25. Atif, M., Bhattacharya, M., Calafiura, P., Childers, T., Dewing, M., Dong, Z., Gutsche, O., Habib, S., Knoepfel, K., Kortelainen, M., Leggett, C., Lin, M., Pascuzzi, V., Tsulaia, V., Wang, T., & Yu, H. (2023). Evaluating Portable Parallelization Strategies for Heterogeneous Architectures in High Energy Physics. Available at https://arxiv.org/abs/2306.15869
26. ALPAKA Documentation. (n.d.). Introduction to ALPAKA. Available at https://alpaka.readthedocs.io/en/stable/basic/intro.html