The Portability Challenge in HPC:

Why Code Should Run Everywhere

Authors: Gianmarco Accordi, Davide Gadioli, Gianluca Palermo

Institution: Politecnico di Milano – Dept. of Electronics, Information and Bioengineering

High-Performance Computing (HPC) has always been about pushing the boundaries of performance. From simulating climate change and modelling drug interactions to training large-scale AI systems, scientists and engineers depend on supercomputers to answer more complex questions than ever before. However, as hardware evolves, the portability challenge has emerged. Portability refers to the ability of code to run efficiently across different computing architectures. Historically, most HPC applications were designed for CPUs, requiring only minor adjustments when moving between different processors. Today, the dominance of GPUs and other accelerators has completely changed the scenario. Applications that want to achieve top performance often need to be rewritten in vendor-specific languages, such as CUDA for NVIDIA GPUs. This raises a key question: How can we ensure that HPC codes remain usable and performant across diverse and evolving architectures without the need for constant rewriting?

Portability: What and Why

Portability in computing has two related but different meanings. The term code portability refers to the ability to compile and run the same source code on different hardware platforms with minimal modifications. On the other hand, performance portability refers to the ability of that code to achieve comparable levels of efficiency across platforms. In practice, it is not enough for a program to run on a new machine, it must also run well and efficiently. This is because scientists and engineers would otherwise risk spending valuable time and resources rewriting and optimizing applications for every new generation of hardware.

ACHIEVE project is developed and delivered under European Union's Digital Europe Programme Project no. 101190015

Co-funded by the European Union

Several trends have made portability a central point in HPC. The rise of heterogeneous architectures is one of them. In the last decade, we've seen a major shift in the design of high-end servers. Supercomputers are no longer made up of rows of identical CPUs. Instead, they have become hybrid systems that combine CPUs, GPUs, FPGAs, and even early quantum accelerators. At the same time, the vendor landscape has diversified. For years, NVIDIA GPUs dominated accelerator-based HPC, but now AMD, Intel, and other players are entering the market with their own hardware and programming models. This change is reflected in the Top500 list of the world's fastest supercomputers. An increasing proportion of the leading machines now use GPUs or other accelerators to deliver their maximum performance, demonstrating just how important heterogeneous computing has become in driving scientific discovery and engineering innovation.

To take advantage of the significant computational power offered by such accelerators, we must offload some of the computation to them. However, this often requires the use of hardware- or vendor-specific programming languages, leading to the well-known problem of vendor lock-in. Applications written exclusively in CUDA, for instance, are effectively tied to NVIDIA GPUs, while codes developed using AMD's HIP or Intel's low-level frameworks are restricted to those ecosystems. These constraints significantly reduce portability and pose challenges to long-term sustainability, particularly in publicly funded research, where neutrality and broad usability are essential requirements. Furthermore, codes optimized for one type of hardware may underperform, or even fail to run, when ported to a different architecture. Finally, the rapid evolution of HPC systems highlights the importance of future-proofing. A code that runs only on today's hardware may require a complete revision to run on tomorrow's hardware. In practice, portability plays a critical role in reducing maintenance costs, extending the lifespan of scientific applications, and ensuring they can cope with successive generations of computing architectures.

The Road Ahead: Balancing Portability and Performance

Given the heterogeneity of hardware and vendors, and the rapid evolution of system architectures, the holy grail for HPC is achieving performance portability in some way. To address these challenges, the HPC community has started to adopt new programming models and languages that are designed to abstract hardware differences while still enabling high performance. Examples in this directions include SYCL, an open standard based on C++ from the Khronos Group that provides a single-source programming model for the host and accelerator; OpenMP, a directive-based approach mainly used for shared-

memory programming on CPUs that has now evolved into a standard for heterogeneous computing; and Chapel, a high-level parallel programming language developed by Cray (now part of HPE) that allows parallelism and locality to be expressed abstractly. Other notable frameworks include Kokkos, OpenACC, OpenCL, RAJA, and Alpaka. These frameworks, which represent different approaches to the problem, all rely on compiler-supported actions to remap generic code to hardware-specific optimizations. However, this is easier said than done. Hardware vendors will always continue to optimize their proprietary frameworks, while abstractions inevitably incur overheads.

Nevertheless, the trend is clear: the HPC community is moving towards programming models that prioritize sustainability and portability. To balance portability with performance, researchers are increasingly writing application codes in portable, high-level, parallel frameworks, sometimes alongside CUDA or HIP kernels for the most performance-critical parts. In practice, rather than one universal language, the solution may be a layered strategy of using portable frameworks for most of the code while tuning specific kernels for target architectures. It should be noted that, even in highly optimized embedded and HPC codes, it is still possible to find kernels written in assembly language for efficient execution.

However, measuring performance portability is far from straightforward. This involves balancing raw performance against architectural coverage and capturing development-side concerns such as programmer productivity, code maintainability, and the degree of divergence across implementations. These aspects are difficult to quantify consistently, which makes measurement another central challenge in HPC research.

Why Students Should Learn About Portability

For students entering the world of HPC, understanding portability is not just a technical detail, but it is a fundamental skill. Modern HPC systems are built on heterogeneous architectures, and the ability to adapt code for use with CPUs, GPUs, and emerging accelerators is crucial for long-term employability. Students who understand concepts such as performance portability are better prepared to contribute to large scientific projects, where codes may need to run on various supercomputers and in industrial clusters. Learning portability also fosters adaptability and critical thinking. It trains students to design solutions that are not tied to a single vendor, but instead are flexible enough to survive future hardware changes. In today's fast-moving technological landscape, this mindset is as important as raw coding ability.

Conclusions

In today's HPC landscape, portability is no longer a luxury, but it is a necessity. With systems becoming more heterogeneous and accelerators more diverse in terms of architectures and vendors, the ability to write once and run everywhere will be one of the keys to the sustainability of scientific software. Fortunately, the aforementioned portability frameworks and tools are paving the way, making it possible to combine portability with high performance. These tools are no longer just for research purposes, but they can be used to train students in graduate courses, as the ones planned for students on the double degree Master's in Cloud, Networking Infrastructure and HPC, supported by the ACHIEVE project. The current challenge lies in shaping a new generation of HPC developers who consider portability as a core design principle rather than an additional consideration.

ACHIEVE project is developed and delivered under European Union's Digital Europe Programme Project no. 101190015

Co-funded by the European Union