What Is Performance Portability for CSE Applications?

The IDEAS Scientific Software Productivity Project

ideas-productivity.org/resources/howtos/

Portability: An application code is portable if it can run on a diverse set of platforms without needing significant modifications to the source and can produce predictably similar output.

Performance portability: An application has portable performance if in addition to running on diverse platforms it exhibits similar accuracy, stability, and reliability across these platforms for a given configuration. Moreover, the time to solution should reflect efficient utilization of available computational resources on each platform.

Performance components in CSE applications: The performance of computational science and engineering (CSE) applications depends on the model, discretization, numerical algorithm, input data, and implementation. In roughly homogeneous platforms with similar basic architecture, implementation details such as data structures, synchronization, and other communication patterns dominate performance portability of an application. With heterogeneity and diversity in platform architecture, however, all the factors (the application model, its discretization, numerical algorithm, input set, and implementation) can impact performance portability.

Performance factors: The performance of a code can be measured from many different perspectives. From the application science perspective these include (1) quality of solution, (2) stability of the code, and (3) time to solution. From the system perspective these include (1) quality of machine utilization; (2) percentage of peak on the machine, measured in operations per second (e.g., FLOPS/s), bandwidth, or some other normalized value; and (3) throughput of the machine.

Constraints on performance: Many CSE applications require flexibility and composability because they address different physical regimes either within the same simulation or in different instances of simulations. Composability comes at the cost of raw performance and plays both positive and negative roles in performance portability. On the positive side, composability also allows flexibility and separation of concerns, whereby different experts can focus on performance of different aspects of the code. Also, composability encourages individual components to be limited in scope, thus making it easier for the compiler to analyze and optimize. On the negative side, increasing composability decreases the compiler's ability to localize data use between functions.

Performance portability in the past: *Golden Age of Vectorization* – 1970-1980s – with vector machines (mostly Cray). *Golden Age of MPI* – 1995-2005 on systems with high-performance networks. The dominant abstract machine model that enabled applications to achieve portable performance was that of a powerful CPU with 2-3 levels of cache hierarchy and a generous amount of DRAM. The dominant parallelization mode was the distributed-memory model. The finer details differed, and maximizing performance on any one platform needed specific

This material is based upon work supported by the U.S. Department of Energy Office of Science, Advanced Scientific Computing Research and Biological and Environmental Research programs.

optimization; but reasonable performance could be obtained across platforms by programming to the general abstract machine model.

Current impediments to performance portability: At this time the most common solution for performance portability is to maintain a different version of the performance-critical sections of the application code, for every machine variant. In different applications this may range from a few kernels to most of the application. Some applications may even require completely different organization of the code and data structures. Known machine variants at present are multi- or many-cores with or without accelerators, and whether the data transfer occurs across PCI interface. An additional factor is the degree of available SIMD or vectorization. Most often used programming languages (C, C++, and Fortran) and parallel programming environments (MPI and OpenMP) in CSE cannot provide a solution at present, although MPI with the latest OpenMP standards may provide a reasonable solution in future. One of the major obstacles to performance portability is the diverse and conflicting set of constraints on memory access patterns across devices. Contemporary portable programming models address many-core parallelism (e.g., OpenMP, OpenCL, OpenACC) but fail to address memory access patterns.

No general solutions currently exist for achieving good performance across all the variants. Individual application codes have achieved some degree of performance portability through development of domain-specific languages and custom back-ends for the target platforms, but they are at best boutique solutions.

Future possibilities: Researchers at DOE laboratories and in industry are exploring software abstractions based on standard C++ features to solve HPC performance and portability problems for current and future platforms. The Kokkos and RAJA[5] C++ libraries, developed by Sandia National Laboratories and Lawrence Livermore National Laboratory, respectively, attempt to enable applications and domain libraries to achieve performance portability on diverse many-core architectures by unifying abstractions for both fine-grained data parallelism and memory access patterns. Nvidia is taking a similar approach in the HEMI library. Another approach for achieving performance portability is using domain-specific languages (DSLs) that map code to the different back-end languages such as CUDA and OpenMP, for example PyOP2. These DSLs create a portable code across multicore architectures, but they are limited to one specific problem domain.

FAQ:

Q: I use my code only on a local cluster, so why should I worry about performance portability?
A: In the past, nodes of the cluster were relatively uniform, and the parallel programming model was distributed programming. Now, heterogeneity is coming even at the node level, so performance portability is everyone's problem.

This document was prepared by Anshu Dubey with key contributions from Todd Gamblin, Michael A. Heroux, Irina Demeshko, and Barry Smith.

This material is based upon work supported by the U.S. Department of Energy Office of Science, Advanced Scientific Computing Research and Biological and Environmental Research programs.