Guest Editorial

Emerging Programming Paradigms for Large-Scale Scientific Computing

High performance computing (HPC) systems are experiencing a rapid evolution of node architectures as power and cooling constraints limit increases in microprocessor clock speeds. In response to this power wall, computer architects are dramatically increasing on-chip parallelism to improve overall performance. The traditional doubling of clock speeds every 18 to 24 months is being replaced by increasing the number of cores for explicit parallelism. During the next decade, the level of parallelism on a single microprocessor will rival the number of nodes in the most massively parallel supercomputers of the 1980s. By 2020, extreme scale HPC systems are anticipated to have on the order of 100,000 to 1,000,000 sockets, with each socket containing between 100 and 1000 potentially heterogeneous cores. These enormous levels of concurrency must be exploited efficiently to reap the benefits of such exascale systems.

This explosion of parallelism has two significant challenges from a programming perspective. The first is how to best manage all the available resources to ensure the most efficient use of the peak performance provided by the hardware designs. The other, and equally important, question is how the enormous potential of these systems can be effectively utilized by a wide spectrum of scientists and engineers who are not necessarily parallel programming experts. The problem for application programmers is further compounded by the diversity of multicore architectures that are now emerging, ranging from complex out-of-order CPUs with deep cache hierarchies, to relatively simple cores that support hardware multithreading, to chips that require explicit use of software-controlled memory. Algorithms must therefore expose parallelism at multiple levels to effectively exploit these diverse architectures.

These changes are forcing the computational community to re-examine fundamental approaches in the design of parallel languages and runtime systems, as they have a profound effect on the productivity and efficiency of application design and execution. This special issue examines several key research topics geared towards productively enabling high utilization of next-generation supercomputing systems. Broadly speaking, researchers have taken two paths for leveraging the parallelism provide by modern platforms. The first focuses on optimizing parallel programs as aggressively as possible by leveraging the knowledge of the underlying architecture; thus quantifying a methodology for maximizing system performance. The second path provides the tools, libraries, and runtime systems to simplify the complexities of parallel programming, without sacrificing performance, thereby allowing domain experts to leverage the potential of high-end systems.

The first four papers in this special issue pursue the strategy of optimizing applications to take advantage of the system architecture. The paper by Madduri et al. presents an analysis of four modern multi-core architectures in the context

of gyrokinetic particle-in-cell methods. This paper showcases different optimization techniques that can be applied to various computing platforms to realize best performance. This case study uses three “traditional” multicore CPU designs and one General Purpose GPU (GPGPU) to analyze the performance characteristics of a fusion simulation. Thus the difficulty of parallel performance optimization is further compounded by the subtle but non-trivial architectural differences as well as the irregular memory access patterns and fine-grain synchronization requirements of the underlying algorithms.

Modern architectural designs are also witnessing a paradigm shift towards heterogeneous HPC platforms, which now constitute some of the world’s largest supercomputers. These systems prove to be extremely challenging optimization targets as they contain numerous levels of parallelism (and sometimes different programming models at each level) and the need to efficiently orchestrate communication and computation among all the available resources. This special issue presents three studies that explore detailed code optimization on heterogeneous platforms. Two of the papers, one by Xian and Takayuki, and the other by Feichtinger et al., analyze the Lattice Boltzmann Method (LBM) as an alternative for solving the Navier-Stokes equations for computational fluid dynamics (CFD) problems. Applications in the Xian and Takayuki paper were run on a multi-node nVIDIA GPGPU cluster using CUDA with MPI via a host server. The research group of Feichtinger et al., on the other hand, used a heterogeneous cluster consisting of CPUs and GPGPUs with varying node configurations. The third study by Kerbyson et al. shows how wave-front algorithms can be optimized on the petascale Roadrunner platform that is based on the Sony Toshiba IBM (STI) Cell processor. All these algorithms are important not only for their wide applicability in computational science but also as case studies for other applications with similar memory access characteristics.

The remaining five papers in this special issue focus more on hiding the afore-mentioned difficulties and nuances of parallel computing, while allowing application domain experts to leverage performance benefits of parallelization without resorting to hand optimizing their codes. Two of these articles examine the motivations for different parallel programming models and their associated runtime environments. They focus on the implementation of efficient communication runtimes to deliver high network performance while maintaining simple interfaces to their associated libraries. The first work by Jin et al. shows how a popular parallel programming paradigm, MPI+OpenMP, can be modified to further leverage locality and exploit the hierarchical memory layout of multicore architectures. Results demonstrate the viability of this hybrid approach to improve load balance and numerical convergence. The authors claim that although modern parallel languages are maturing, the traditional MPI+OpenMP strategy is still widely used on SMP clusters. The paper by Nishtala et al. investigates how to build collective operations, i.e. data movement functions that require participation amongst all the nodes involved, across a wide variety of modern platforms. The paper points out the need for automatically selecting the best communication algorithm, schedule, and topology in order to maximize

parallel performance. It also analyzes the tradeoffs associated with one- and two-sided communication mechanisms for the data transfers.

Lastly, we present three papers that propose higher-level language semantics to further abstract the problems of parallel computing. Gay et al. show the benefits of a new programming language called Yada. The key feature of Yada is that programming is parallel by default with constructs for expressing serialized execution. Evaluations of a prototype on a dozen algorithms and applications demonstrate good parallel speedup with only a few minor modifications being necessary. A study by Plimpton and Devine shows how the popular cloud computing paradigm, MapReduce, can be applied to graph algorithms, even those that need access to out-of-core data. Their MPI implementation allows efficient processing of terabyte-scale datasets on distributed-memory clusters. The final investigation by Wilde et al. focuses on improving parallel programming productivity by simplifying the process of composing numerous parallel programs together. Their parallel scripting language called Swift reduces the complexities of data management and application execution in two ways: first by making file system structures accessible via language constructs, and second by allowing domain-specific codes to be decomposed into parallel scripts.

Overall, the articles in this special issue provide an excellent overview of the different research areas in programming paradigms engaged by the parallel computing community. Numerous teams have made significant impact; however, the dream of allowing domain experts to quickly and simply express an efficient computational program at scale is yet to be realized. We hope that the variety of topics presented in these studies will help provide the insights needed by future researchers to build and deliver the languages, runtimes, and libraries that attain the goal of productive, efficient, and portable programming for tomorrow's exascale systems and beyond.

Leonid Oliker
Computational Research Division / NERSC,
Lawrence Berkeley National Laboratory,
Berkeley, CA 94720, USA

Rajesh Nishtala
Department of Electrical Engineering and Computer Science
University of California,
Berkeley, CA 94720, USA

Rupak Biswas
NAS Division,
NASA Ames Research Center,
Moffett Field, CA 94035, USA
Tel: +1 650 604 4411; fax: +1 650 604 3957
E-mail address: rupak.biswas@nasa.gov