Guest Editorial

Special Issue: Exploring languages for expressing medium to massive on-chip parallelism

Gabriele Jost\textsuperscript{a} and Alice Koniges\textsuperscript{b}
\textsuperscript{a} Texas Advanced Computing Center, The University of Texas, Austin, TX, USA
\textsuperscript{b} National Energy Research Scientific Computing Center, Berkeley Lab, Berkeley, CA, USA

1. Introduction

The many-year trend of increasing processor speed has ended, and high-performance computing (HPC) is looking for a path to Exascale computation (i.e., at a rate exceeding 10\(^{18}\) operations per second) that does not involve simply improving the processor speed/clock rate. Indeed, ‘Going to the exascale’ will mean a radical change in computing architecture – basically, vastly increasing the levels of parallelism to the point of millions of processors working in parallel. Newer processors tend to have larger numbers of cores, and there is a hierarchy of shared memory cores and distributed nodes containing those cores. A natural way to deal with this increase in the number of cores as well as the lack of uniform memory access is to re-examine programming languages. This Special Issue addresses the programming model developments that are being studied to deal with the next generation of hardware, including a new language effort and new features in current languages.

2. The papers

The papers presented in this Special Issue of Scientific Programming discuss and illustrate the issues, problems and trends related to hardware developments and their implications on programming models. They cover a broad range of topics starting from conventional programming such as combining MPI and OpenMP (often called “Hybrid Programming”) to new and upcoming models. Some of the studies present an application programmer’s point of view, while other contributions come from compiler developers.

MPI is the most commonly used model for message passing while OpenMP is the currently most common shared memory programming model. Both MPI and OpenMP have been in use for several years, and even the hybrid approach of combining MPI and OpenMP has been well studied. However, newer architectures seem to be a better fit for the hybrid models, and performance gains in hybrid programming are becoming more prevalent. As hardware architectures become increasingly hierarchical displaying shared as well as distributed memory characteristics, it seems natural to combine MPI and OpenMP when trying to exploit parallelism on multi-core node clusters. It is therefore not surprising that this is the path most often taken by the scientific programmer.

Three of the papers in this series address issues resulting from the combination of MPI and OpenMP programming models. The paper “Experiences using hybrid MPI/OpenMP in the real world: Parallelization of a 3D CFD solver for multi-core node clusters” (Jost and Robins) describes how the performance and scalability of an existing MPI code was improved by adding OpenMP directives to time consuming loops. This paper describes a full-scale real world application rather than a benchmark. It also provides detailed performance analysis, which demonstrates challenges and opportunities of the hybrid MPI/OpenMP approach. The paper “Overlapping communication with computation using OpenMP tasks on the GTS magnetic fu-
The future

One thing we know for the future – it will no longer be “business as usual” with performance improvements directly in the processor fueling subsequent increases in application performance. Instead, the switch to hybrid multi-core and even GPGPU architectures fuels a corresponding revolution in programming models. While many computational scientists believe that the new models may be “MPI + X,” where the X denotes a “to be determined” parallel language designed for within the multi-core nodes, and MPI remains across the nodes, the actual prediction of what X will be is unknown. Additionally, this revolution in programming models allows for the possibility that entirely new language constructs and formalisms may be developed that really change the way we attack parallel high-performance programming for increases not only in performance but also in programmer productivity.

Other issues that need to be considered include the existence of legacy applications based on the conventional models like MPI and OpenMP. These will require runtime support for their efficient execution: light-weight message passing and synchronization, efficient collective operations and the overlap of communication and I/O needs to be supported. New, more suitable, programming models depend on the possibility an easy conversion from the old to the new model. Another important aspect is efficient memory access for entities like global arrays. The papers presented in this issue provide a starting point for this new era in computation.