Scientific Programming 16 (2008) 101–103
DOI 10.3233/SPR-2008-0258
IOS Press

Foreword

Barbara Chapman [a] and Dieter Kranzlmüller [b,c,d]

[a] *Department of Computer Science, University of Houston, Houston, TX, USA*
[b] *GUP – Institute of Graphics and Parallel Processing, Johannes Kepler University of Linz, Linz, Austria*
[c] *Department of Informatics, Ludwig-Maximilian University of Munich, Munich, Germany*
[d] *LRZ – Leibniz Supercomputing Centre, Garching, Germany*

The current growth in scale and complexity of the computing systems used for High End Computing (HEC) is unprecedented. Not only do modern HEC platforms consist of up to several thousands of interconnected nodes, but the nodes themselves have become highly complex. Those configured today as components of large-scale computers are themselves heterogeneous multicore, potentially multithreading parallel systems on a chip. Further complexity is introduced by the combination of HEC systems into heterogeneous grid infrastructures, which are collaboratively used for solving scientific problems.

The number of concurrently executing threads that can be supported on a single node of an HEC system is projected to increase rapidly. As a result, such platforms pose traditional parallel programming challenges, exacerbated by the scale of the system, and additionally dramatically increase intra-node programming challenges. The sharing of resources among threads may lead to resource contention, which affects the efficiency of the codes on multiple levels: on the node itself, within the surrounding HEC platform, and on the grid containing this resource.

The HEC application developer is thus confronted with computer systems with multiple levels of architectural parallelism, each with its own set of challenges. To address this situation, application developers have begun to experiment with new programming models, e.g. the combination of MPI and OpenMP, where the former supports scalable programming across nodes while the latter provides a widely supported high-level model for shared memory parallel programming within the nodes. Emerging parallel programs may therefore have multiple levels of parallelism, where the layers of parallelism are potentially expressed via several different programming models.

The applications themselves are also increasing in complexity. Current trends in science and engineering indicate multidisciplinary programs, which may be multiscale, regionally irregular, and may need to deal with rapidly growing problem sizes and data sets. They may be written using several different programming languages, in order to exploit the most useful features of each of them where appropriate. In order to fully tune a program for a given architecture, the application developer may thus need to understand in depth the characteristics of a number of different hardware and software components, and their interactions, and may need to tune for multiple languages and multiple parallel programming paradigms. Even power consumption is often an issue that can no longer be neglected. Clearly, the task of developing and deploying high end applications has reached an unprecedented level of complexity.

It is the role of programming tools to provide powerful help to overcome these extraordinary difficulties. Suitable HEC programming tools are expected to aid the programmer in developing, tuning and deploying large, complex applications across a variety of computers from conventional clusters to computational grids. The tools developers themselves must learn to deal not only with the details of emerging architectures, but with the challenge of handling exceptional amounts of information. They have to master the difficulties inherent in gathering, processing and condensing huge amounts of data on a program and its behavior, or in monitoring aspects of a huge, distributed system. They may need to identify the salient facts buried within a very large amount of relatively unimportant information; they will need to help the user to select and interpret information related to a given program or its execution, in a manner that will enable problems of program development and deployment to be resolved. To accomplish these goals, the tools' developers may need to devise new approaches to traditional problems, and to overcome significant implementation challenges. It must be their focus to provide means for reducing the overall complexity of the tasks facing the user. Fortunately, progress is being made on a number of fronts and work is already under way to provide software support to the program developer on new high-end platforms.

In this volume, we are very pleased to present some of the most important on-going efforts to create software and techniques that will enable the next generation of application developers to accomplish their goals. These 8 papers are the result of an open call to the tool developers' community and have been selected after extensive review by a group of international experts over the last few months. Each paper has received at least 3 reviews, ensuring the usual high level of quality.

Five of these papers discuss progress in tools that aid performance analysis and improvement:

- In their contribution, Schulz and colleagues describe the Open|SpeedShop project, an extensible set of tools that interoperate with the goal of making sophisticated performance analyses more accessible to large-scale application developers. Key components of their system provide a variety of common program analysis tasks that can be exploited by multiple tools in order to facilitate the integration of new tools.
- Huck, Malony, Shende and Morris describe a performance analysis framework that is designed to support the analysis of very large sets of performance-related data and compare results from multiple experiments. They describe enhancements to PerfExplorer, a framework for parallel performance data mining and knowledge discovery, giving several examples of its use. PerfExplorer is part of the TAU performance analysis framework which has been widely deployed to support the tuning of HEC codes.
- Hernandez et al. describe the benefits of integrating compiler technology with performance tools, showing how this may potentially provide higher levels of automation to the end user. The compiler can supply context information to help interpret performance data, help to make decisions for selective instrumentation, and support the task of bottleneck analysis. Moreover, a methodology for performing bottleneck analysis is described and illustrated. Both MPI and OpenMP performance problems are addressed.
- Chan, Gropp and Lusk explain that one of the most important requirements for performance visualization tools such as their own Jumpshot is the ability to allow the user to examine small time intervals in considerable detail. They have developed a hierarchical trace file format, SLOG2, which enables the display of an arbitrary time window with time roughly proportional to the number of events within the window. This enables them to extract the desired information from very large trace files without loss of detail.
- Some kinds of behavior only manifest themselves at extreme scales. Wylie and co-authors demonstrate the use of the newly-developed SCALASCA toolset to quantify and isolate a range of significant performance issues in real world high-end applications on 3 different systems with thousands of processors. This toolset is based upon a redesign and re-engineering of the KOJAK toolkit to address highest levels of scalability in performance measurement, analysis and investigation.

The remaining three papers provide the latest research results in the area of tools for grid computing:

- In their contribution, Baker and Boakes report on a system called Slogger, which supports the user in gathering and analyzing heterogeneous log files generated by various layers within a distributed system. Their tool utilizes emerging Semantic Web technology to obtain and store the data, and for querying and visualizing potential problems within the user's application. The goal of the tool is to support the understanding of why an application in a distributed system is not behaving as expected.
- The observation of program behavior on grids is also the focus of the paper by Balis et al., who describe a tool for the monitoring of scientific grid workflows. This is exceptionally challenging for the user, as grid workflows are possibly highly distributed, loosely coupled in space and time, heterogeneous, and probably using legacy codes. The GEMINI monitoring system proposed by the authors is described together with the data correlation problem and an algorithm for on-line distributed collection of monitoring data. With examples from real-world workflows, the authors demonstrate the usability of their prototype implementation.
- Finally, Peachey et al. presents the latest addition to the Nimrod framework, where fractional factorial design and associated analysis tools can now be used for performing parameter sweep experiments on the grid. The resulting environment pro-

vides a convenient way to automate the design of an experiment, to execute the jobs on the grid, to return the results to the user, and to assist the user in interpreting the results. Nimrod itself has been extensively used for exploration of scientific data. The new extension and its associated tools should enable the execution of carefully designed experiments with the power of distributed systems.

The contributions to a volume such as this one can only provide insight into the progress made in certain areas of support for the application developer, and the tools and technologies described here represent just a subset of possibilities in this domain. However, we believe that it is important to raise the application developers' awareness of the tools that are being created to enable them to perform their work with higher levels of productivity, while we also hope that the tools' providers may also get new ideas for future improvements.