

Towards High-Level Programming Support for Scientific Computing on Clusters

Extended Abstract – Keynote Address Cluster 2001

Hans P. Zima

*Institute for Software Science, University of Vienna
Liechtensteinstr. 22, A-1090 Vienna, Austria
E-Mail: zima@par.univie.ac.at*

Programming languages define the level of abstraction at which a user interacts with a machine. If target code performance is essential (which is the case for almost all scientific applications) then a compromise must be found between the advantages of high-level programming such as enhanced expressivity, verifiability and portability, and the performance achievable by the explicit exploitation of architectural properties offered by a low-level notation. Scientific applications for clusters have been traditionally parallelized using a combination of a traditional sequential programming language such as Fortran or C/C++ with low-level languages or libraries (such as PVM or MPI) allowing virtually complete control over the structure and degree of parallelism as well as the distribution and alignment of data across the architecture. Program development for performance-critical applications on clusters of SMPs has often gone a step further by adopting two paradigms (such as OpenMP and MPI) in order to fully exploit the specific characteristics of parallelism at each level of the hybrid architecture.

In this talk we will discuss recent developments oriented towards providing the programmer of scientific applications for clusters with higher level of abstraction while maintaining the goal of exploiting performance to the best possible degree. We will particularly focus on the requirements of irregular, dynamic and adaptive applications, and discuss issues related to an integrated program development supporting adaptive restructuring of programs and data.

Many advanced scientific problems are of an irregular nature and characterized by a large amount of inherent parallelism. Examples include sparse matrix computations, sweeps over unstructured grids, tree searches, and particle-in-cell codes; moreover, many important problems display an adaptive behavior. In order to execute such codes efficiently on a cluster architecture, the distribution of work and data must be managed dynamically, and a viable tradeoff between the exploitation of locality and balancing of the workload must be determined depending on parameters of the application, its input set, and the architecture. Providing high-level programming support for running such applications requires an integrated approach combining elements of parallel programming languages, new compilation techniques, and tools supporting performance prediction and analysis.

Programming languages (or their directive-based extensions) must offer linguistic elements for the specification of shared-memory as well as distributed-memory control parallelism and provide features for aligning threads with data. Data distributions as offered by High Performance Fortran are too restrictive and pose too many performance problems to be useful in this context. A promising generalization interprets a data distribution as an object specifying an arbitrary mapping from a data structure index domain to a “pro-

cessor" index domain, which at the time of its instantiation may be parameterized with properties of the architecture and/or the application and its execution status. Such objects can be dynamically bound to one or more data structures which, in addition to Fortran or C arrays may include trees, lists, or sparse matrices. Depending on a distribution and the related class of data structures, specialized run time procedures for instantiation, access, and redistribution may be generated.

The role of compilers in a programming environment supporting dynamic and adaptive applications in a potentially changing hardware environment is undergoing a radical transformation. In particular, the traditional separation of compile time and runtime can no longer be maintained. More specifically, the concept of a compiler is no longer restricted to the static analysis and translation of a program, but compilers may be activated at runtime (just-in-time) to dynamically react to changing conditions in the hardware environment or to performance problems recognized during the execution of the program. Such an enhancement in the role of compilers must be supported by a sophisticated set of software tools. In particular, performance tools must be able to predict program performance, instrument critical segments of the program, recognize bottlenecks, and provide high-level analyses of performance measurements which summarize key aspects of program behavior in terms related to high-level program semantics. In the long-term, AI technology may significantly reduce the role of the user and provide largely automatic support for the tuning and restructuring of programs based on static as well as dynamic analysis.

In the context of this talk we will also give a short overview of a number of key installations of clusters in Europe and discuss related application domains and research activities.