

PROGRAMMING MODELS FOR HIGH PERFORMANCE COMPUTING

Barbara Chapman, University of Houston and University of Southampton, chapman@cs.uh.edu

Abstract

High performance computing (HPC), or supercomputing, refers to all aspects of computer technology required to solve the most challenging computational problems of the day. By definition, these problems require innovation in hardware, software and computer system infrastructure. During the brief history of research and development in Information Technology, HPC has therefore served as a testbed for the development and early deployment of new technologies. Although the immediate market for HPC products is not large, it is a critical one, including industry sectors such as aerospace and automobile construction, a variety of other engineering disciplines, weather forecasting and financial analyses. Many of the innovations that affect our everyday use of computers begin in high performance computing, where leading edge users attempt to squeeze maximal performance out of the fastest machines of the day.

In this article, we look at recent developments in supercomputer hardware and the rise of programming standards for such systems. We discuss current challenges in the provision of standards for HPC application development and consider how Europeans participate in the increasingly global HPC arena.

Keywords: supercomputing, high performance computing, parallel programming, MPI, OpenMP

1 INTRODUCTION

As a research and development discipline, High Performance Computing spans a variety of activities from the development of new algorithms and numerical methods for solving particularly challenging computational problems to the design and construction of novel computer hardware. Computer architects build and deploy new and faster machines to execute supercomputing applications, application programmers learn how to write code for these machines and system software designers create tools to help them achieve good performance.

Those whose task it is to develop applications that have exceptional computational requirements are accustomed to the hard work required to create and run programs effectively on emerging architectures. Not only must they gain an understanding of the characteristics of new hardware and learn to apply emerging methodologies, systems and software, they frequently also need to master

new programming models. However, this reprogramming is labor-intensive and inefficient, and HPC applications experts are scarce. Standards for application development that work across different HPC platforms are vital.

But providing programming standards for HPC application creation is a major challenge. Not only are there at any given time a variety of architectures for large-scale computing, the need for performance does not permit the use of standards that cannot be efficiently implemented and may well require exploitation of specific features of an individual hardware system. Moreover, HPC is a relatively small market and vendors can support just a few programming models that target it, so a standard must be usable for a large fraction of HPC codes.

HPC has enjoyed considerable support from the programs of the European Commission. Although there are virtually no European hardware vendors in Europe, many major HPC applications are of European origin and their owners are fast to adopt new programming standards. European researchers have a strong tradition in language design and have made significant contributions to current standards.

In this article, we discuss the hardware technology that is currently used to execute HPC applications and then consider attempts to define programming standards for the creation of such applications, the main focus of our paper. The 1990s was the era of standardization and hardware that was deployed at the time dictated the nature of these activities. They must be adapted to solve programming problems posed by the HPC systems that are beginning to be deployed; we briefly consider some of the ways in which this might occur. We conclude with some remarks on the increasing globalization of research and development activities that has removed many traditional barriers to active collaboration with international partners. Europeans have more opportunities than ever to help shape the future of high performance computing.

2 HPC COMPUTING SYSTEMS

High-end hardware technology is evolving rapidly. In Michael Crichton's Jurassic Park novel, a vector supercomputer from Cray Research controls major operations at the park (in the film it is a system from Thinking Machines); when the film was made, many of its images were generated using a supercomputer made up of a number of Sun workstations. High performance

computing technology had developed faster than had the bioengineering that was central to the novel's plot.

2.1 Development of HPC Systems in the 1990s

At the start of the past decade, vector supercomputing systems such as those marketed by Cray Research, Fujitsu and NEC, were generally used to run large-scale applications. Two to four vector processors were combined to form particularly powerful systems with a single shared memory. Bus-based shared memory parallel systems (SMPs), consisting of small numbers of RISC CPUs that share memory, were also available. Transputers had given Europeans their first taste of inexpensive parallel computing, and some researchers had migrated to the new distributed memory parallel platforms (DMPs), produced by companies such as Intel, Meiko and nCUBE.

Vector systems were expensive to purchase and operate. As a result, they were usually operated by centers that serviced the needs of a number of paying customers. DMPs could be built in many different sizes, from small to (in principle) arbitrarily large configurations, and so it was possible to build machines to suit the budgets of individual organizations. They appeared to be a suitable basis for the massive parallelism envisioned by leaders of the day. As the technology stabilized, organizations increasingly procured their own DMP supercomputers. Thus the use of distributed memory platforms, as well as the size of systems built, continued to grow steadily throughout the decade and their share of the HPC market rose accordingly [16]. However, vendor supplied systems such as the Meiko CS2, Cray T3D and T3E and IBM's SP architectures, were not the only DMPs in use throughout the '90s. Such systems essentially consist of a number of individual computers, complete with CPU and memory, connected via a custom high-speed network and with system support to ensure that data is rapidly transported between the different memories. Workstations connected by a local area network began to be similarly used together to compute a single job, a trend that was greatly encouraged by the emergence of third-party software to simplify parallel job submission. They became known as Clusters of Workstations (COWs). Although the Ethernet networks used to build COWs were slow in comparison to the custom networks of true DMPs, they were much cheaper to build.

In the latter half of the 1990's, most US hardware vendors began to produce shared memory parallel machines. In contrast to the SMPs built in the late 1980's, these relatively inexpensive systems are intended for wide usage as desktop computers and powerful servers. With the tremendous increase in clock speed and size of memory and disk storage on workstations, many applications that formerly required HPC hardware can

now be executed on a single SMP. Technological advances now enable the construction of SMPs with 16 and more processors.

2.2 Current HPC Systems

Most hardware vendors now combine DMP and SMP technologies to provide the cycles demanded by large-scale computations. IBM's SP series, for example, are essentially a collection of SMPs connected via a fast network, or a DMP with SMP components. As a result, they have two distinct levels of architectural parallelism. Vendors such as Sun Microsystems offer flexible solutions where SMPs of varying sizes may be configured in a DMP platform. A COW, too, may be configured as a cluster of SMPs. Switching technology has produced off-the-shelf interconnects (e.g. Myrinet, Gigabit Ethernet) that reduce the performance penalty incurred by a COW.

Many modern HPC platforms enable code running on a CPU to directly access the entire system's memory, i.e. to directly reference data stored on another SMP in the system. Some have global addressing schemes and others provide support for this in the network. The so-called ccNUMA (cache coherent Non-Uniform Memory Access) systems marketed by Compaq, HP and SGI transparently fetch data from remote memory when needed, storing it in the local cache and invalidating other copies of the same dataset as needed. They can thought of as large virtual SMPs, although code will execute more slowly if it refers to data stored on another SMP (hence the non-uniform cost of access). The cost of such references can be exacerbated by contention for network resources if there are many such references simultaneously. The size of such machines has continued to grow as the technology matures, and they have become an important platform for HPC applications.

The size of HPC configurations is increasing. The number of processors configured on each SMP is growing modestly; the number of components in a cluster is growing rapidly. The Los Alamos National Laboratory, for example, will soon deploy an SMP cluster with 12,000 Compaq Alphas; the components are 4-processor SMPs. It uses a network switch provided by a European hardware vendor, Quadrics Supercomputers World that is based on the ELAN/ELITE technology designed by Meiko in the EC-funded GENESIS project of the early 1990s.

The HPC market does not dominate the agenda of hardware vendors. While it remains an important testbed for new technologies, its market share is declining. Thus most CPUs configured in HPC systems are designed for the mass market. The power of an HPC platform is derived solely from the exploitation of multiple CPUs. In

order to obtain high performance, an application must be suitably rewritten to execute on many CPUs in parallel. The parallel code must use each CPU efficiently; all processors should contribute equally to the computation and their work must be coordinated with the minimum of interference. APIs for parallel application development are needed to realize this programming task. We discuss the rise of APIs for parallel programming next.

3 HPC PROGRAMMING MODELS

Creating a parallel application that utilizes the resources of a parallel system well is a challenging task for the application developer, who must distribute the computation involved to a (possibly large) number of processors in such a way that each performs roughly the same amount of work. Moreover, the parallel program must be carefully designed to ensure that data is available to a processor that requires it with as little delay as possible. Overheads for global operations including start-up times may be significant, especially on machines with many processors, and their impact must be minimized.

In the decade prior to 1990, researchers proposed a variety of new languages for parallel programming, few of which experienced any real use, and the major hardware vendors created programming interfaces for their own products. Vendor extensions were fairly well understood and commonalities emerged among the sets of features provided for architecturally similar machines. Yet although discussions were held with the goal of agreeing on portable APIs for both vector and shared memory programming constructs, there were no standards. As a result, it was common practice for programmers to rewrite HPC applications for each new architecture, a laborious procedure that often required not only prior study of the target machine but also the mastery of a new set of language features for its programming.

3.1 APIs for Distributed Memory Systems

The 1990s was the age of standardization of parallel programming APIs. An established standards committee completed the definition of Fortran 90 with its array operations, broad community efforts produced HPF and MPI and a vendor consortium defined OpenMP for parallel application development. Fortran 90 enables the expression of operations that can be applied to all (or a region of) an array simultaneously. This can support program execution on a vector machine, but does not provide all the features necessary for parallel program execution on a DMP or SMP. We do not discuss it here further.

When a DMP is used to execute an application (we include COWs implicitly in this category from now on),

data stored in memory associated with a processor must be transferred via the system's network to the memory associated with another processor whenever the latter needs it. The most popular solution to programming such machines gives full control over this data transfer to the user. In an analogy to human interaction, it is referred to as message-passing; a "message" containing the required data is sent from the code running on one processor to the remote code that requires it. The individual codes, or processes, are logically numbered for identification. MPI, or Message Passing Interface, was developed by a group of researchers and vendors to codify what was largely already established practice when their deliberations began in 1993 [10]. It consists of a set of library routines that are invoked from within a user code written in Fortran or C to transfer data between the processes involved in a computation. Although it is possible to write an MPI program using few library routines, the API provides rich functionality that enables optimization of many common kinds of communication. It also supports the programming of large systems by, for instance, providing for the creation of groups of processes, each of which may be involved in a distinct subcomputation. MPI was soon available on all parallel platforms of interest, in both public domain and proprietary implementations.

MPI realizes a so-called local model of parallel computing, as the user must write program code for each participating processor, inserting MPI routines to send or receive data as needed. Most MPI programs are created according to the so-called SPMD (Single Program Multiple Data) model, where the same code runs on each processor, parameterized by the MPI-supplied process identification number.

Considerable work is involved in creating an MPI code, and the cost of program maintenance is high. The most appropriate MPI implementation of a program may differ between platforms; it is not uncommon to see multiple MPI versions of an application, each tuned for a target system of interest. However, since the user must specify all details of the program's execution explicitly, its performance can be analyzed with relative ease, and remedies for performance problems sought. Thus it is not ease of programming, but good performance, coupled with wide availability of the API that has led to its continuing popularity. MPI versions of a large number of important applications have been developed and are in wide use. It remains the most widely used API for parallel programming.

An effort to create a programming interface that relies on a compiler to generate the data transfers was also broadly supported. Many researchers and vendor representatives took active part in the definition of High Performance Fortran (HPF)[7]. The HPF Forum began meeting regularly from early 1992 to develop an implicit parallel programming model for high performance

computing; meetings took place in the US, which effectively limited European attendance, although the email discussion lists attracted a huge international audience. The forum kept to a tight schedule, producing its first language definition in just over a year. As the name implies, the features were designed for use with Fortran, and in particular with Fortran 90.

The core of HPF consists of a set of compiler directives with which the user may describe how a program's data is to be distributed among the CPUs of a machine. By default, computation is distributed among the CPUs in such a way that processors are responsible for updating elements distributed to them. However, there are also directives for distributing loop iterations to processors. An HPF application resembles a sequential program; there is one program code. The parallelism is implicit in the sense that it is the compiler that is responsible for creating the explicitly parallel processes, one of which will run on each participating processor. Thus the compiler must allocate storage for local data as well as for copies of non-local data needed by a process, and must assign computations to each processor. Moreover it also has to determine which data is to be communicated between participating processors and insert the necessary communication instructions. This is a complex translation that is easily impaired if sufficient information is not available. For instance, if it is not clear whether or not a statement uses data defined by another statement, as can easily happen if an array is indexed using an unknown variable value, then the compiler must assume that this is the case and insert the appropriate communication. Runtime testing may alleviate the problem, but the compiler-generated code will frequently transfer too much data.

The task of modifying a program to perform well under HPF was poorly understood, requiring a great deal of insight into the HPF translation process and a commitment that few were willing to bring for a new and relatively untested technology. The stringent time constraints adhered to by the HPF Forum did not permit the first version of the language extensions to adequately address the needs of unstructured computations. Moreover, as a result of the complexity of the compilation process, and the need to develop Fortran 90 compiler technology concurrently, HPF products were slow to emerge. Finally, HPF did not address the needs of the growing fraction of HPC codes that were written in C. Thus HPF did not enjoy the same level of success as MPI. Nevertheless, work on this API did lead to an upsurge of research into a variety of compiler optimizations and it continues to influence compiler technology.

3.2 An API for Shared Memory Systems

Both HPF and MPI were developed to support application development on DMPs. Yet during the latter

part of the last decade, SMPs were being sold in large quantities. OpenMP was introduced to fulfill the need for an API for this class of platforms [12]. It is also suitable for ccNUMA machines. First introduced with an interface for Fortran late 1997 by an influential group of computer vendors, it was extended by C and C++ versions just a year later. The programming model realized by OpenMP is based upon earlier attempts to provide a standard API for shared memory parallel programming [20]; it primarily supports the parallelization of loops, as needed for many scientific and engineering computations, and its features were both familiar and appropriate for many such applications. It is much easier to parallelize unstructured computations under OpenMP than under either of the above-mentioned APIs, since there is no need for a prior partitioning of their data to processors.

Like HPF, OpenMP is an implicit parallel programming model, consisting of a set of compiler directives and several library routines. However, it is much easier to compile OpenMP than HPF, and compilers for all three language APIs rapidly appeared. Tools are now available for OpenMP application development, debugging and performance analysis, e.g. [8]. In consequence, OpenMP was quickly accepted as the de facto standard for shared memory parallel programming.

OpenMP realizes the fork-join execution model, in which a single master thread begins execution and spawns worker threads, typically one per available processor, to perform computations in parallel regions. It is the task of the user to specify such parallel regions and to state how the work is to be shared among the team of threads that will execute them. Threads may have their own private copies of some data; other data is shared between them. A conditional compilation feature facilitates the maintenance of a single source code for both sequential and parallel execution (this is especially important if OpenMP library routines are called).

OpenMP can be used in several different ways to develop a parallel program. The straightforward approach does not require the user to analyze and possibly modify an entire application, as is the case with both HPF and MPI. Instead, each loop is parallelized in turn. Performance gains are reasonable and except for the need to understand and handle data dependences in a program, it is even appropriate for use by non-expert application developers [3]. However, the performance requirements of HPC applications generally necessitate a different strategy for using OpenMP.

Problems arise with this kind of parallelization strategy when data being updated is stored in more than one cache of an SMP or ccNUMA platform. This can happen because data is transferred to cache in lines, or consecutive sets of storage locations. A line with updated data may need to be transferred from one cache to

another, leading to significant delays. The delay is more costly on ccNUMA platforms, where memory is physically distributed. Furthermore, we have already seen that frequent or heavy accessing of data on other SMPs of a ccNUMA machine can lead to network contention and considerable degradation of performance. But as a shared memory API, OpenMP does not provide features for mapping data to SMPs or CPUs. An alternative approach to using OpenMP relies on the user to deal with these performance problems by rewriting the program in such a way that data is distributed among the threads that will execute it. To do so, thread-private data structures are created and shared data structures are declared to hold data needed by more than one thread. Since OpenMP provides thread identification numbers, work can be explicitly allocated to threads as desired. The strategy minimizes interactions between threads and can yield high performance and scalability [18]. Although fewer details need be specified than with MPI, this so-called SPMD-style OpenMP presupposes a more extensive program analysis and modification than loop-level parallelization. But it is easier to maintain than a corresponding MPI code, since fewer lines are added.

In recognition of the effort required to create SPMD style code, vendors have also provided a variety of mechanisms to help a user achieve good data locality on ccNUMA platforms. One of these is a default strategy for allocation of data in memory local to the first thread that accesses it; if the application developer carefully distributes the initial data references across threads, this simple mechanism may suffice. An alternative is to automatically migrate pages of data so that they are near to threads that reference them. This is appealing, as it is transparent to the user. However, pages of memory are fairly large, and without application insight it can be hard for the system to determine when it is appropriate to move data. Thus more research into this area is needed [11]. Finally, both Compaq and SGI have provided custom extensions to OpenMP to permit the user to express the data locality requirements of an application, e.g. [4]. They have used the basic approach embodied by HPF, although it is not easy to provide these features in an API that permits incremental program development. Despite considerable overlap of functionality, the syntax and a number of details differ substantially. The features also suffer from the deficiency that they do not explicitly consider the needs of unstructured computations.

3.3 Is A Standard API in Sight?

The HPC community is small and its ability to sustain even a single specific programming standard has been questioned. Yet if an API is to be broadly used in High Performance Computing, it must be expressive enough to allow application developers to specify the parallelism in their applications; it must be capable of efficient

implementation on most HPC platforms and it must enable users to achieve performance on these machines.

We have seen that modern HPC platforms are essentially large clusters of SMPs. With the exception of ccNUMA machines, they are chiefly programmed using MPI or a combination of MPI and OpenMP. In the latter hybrid approach, MPI is used to program the DMP machine, performing synchronization and data transfer between the different SMPs whereas OpenMP is used to program within each SMP, usually with a straightforward loop parallelism. Thus both levels of parallelism in a machine are explicitly programmed for. Where more than one kind of parallelism is naturally present in an application, it is a strategy that can work well, and it is felt by many that this hybrid approach is key to the deployment of some codes on the large SMP clusters now available. Unfortunately, the easiest way to develop such a program, particularly if development begins with an MPI code, is to create OpenMP parallel regions between MPI library calls. It is not an efficient way to use OpenMP and performance results can be disappointing. Several hybrid strategies are discussed in [15].

MPI requires substantial programming and maintenance effort; the hybrid model may make it easier to program very large systems, but neither is a particularly comfortable approach to HPC application development. OpenMP is easier to use and is a more appealing API for high performance computing. Moreover it potentially has a larger market since it may satisfy the needs of a range of applications that do not necessarily belong to HPC, but benefit from exploitation of multiple processors in an SMP. Wider deployment of SMPs, and a broader user community mean greater vendor support.

But there are a number of challenges that must be met if OpenMP is to become a widely used API in the HPC community. First among them is the need to extend its range of applicability. Unlike MPI, which can be implemented on SMPs with reasonable efficiency, OpenMP is hard to realize efficiently on DMPs. The current approach is to implement OpenMP on top of software that provides virtual shared memory on a DMP, a so-called software distributed shared memory system, such as Omni [13] or TreadMarks [1]. Although this technology does not yet provide the performance levels required for HPC applications, it is the subject of a good deal of on-going work and a production version of TreadMarks is expected to appear soon. Further, a promising proposal has been made for a strategy to implement OpenMP on DMPs that enable direct access to memory across all SMPs [9].

OpenMP is not without its difficulties. One of these is the relative ease with which erroneous codes can be produced. It is up to the application developer to insert all

necessary synchronization to coordinate the work of the independently executing threads. In particular, this requires the recognition of data dependences and the correct ordering of accesses, tedious and error-prone work. The very power of OpenMP lies in the ease with which large parallel regions may be created and these may span many procedures, complicating this task. Although the solution may lie in tool support rather than any language modification, it is an issue that requires attention.

Other challenges include the need to facilitate the expression of multiple levels of parallelism in OpenMP codes, and to provide additional features that support the creation of codes for very large platforms. Starting up threads across an entire machine, for example, can become very costly, as can the allocation of shared data and global operations such as reductions. Unstructured computations will require more language and compiler support for efficient execution across large numbers of SMPs. Some applications require synchronization that is hard to express in the current OpenMP language [14].

OpenMP currently supports only the simplest expression of hierarchical parallelism. There is no facility for the distributed execution of multiple levels of a loop nest in parallel within a single parallel region, let alone the ability to assign work to threads within the same SMP. Yet being able to exploit this hierarchy could be critical to achieving scalable performance. This might be realized by the provision of means to indicate how resources should be allocated to nested parallel regions, possibly with some architectural description. Features to allow for the grouping of threads would also help, and could permit several independent subcomputations to take place concurrently. It is to be hoped that at least some of the above can be added without loss of the comparative ease of programming. A number of proposals exist for extending OpenMP [2,5].

A slight relaxation of the semantics of OpenMP might also improve its performance for some applications. The current standard requires that the user state that there are no dependences between a pair of consecutive parallel loops. Otherwise, the compiler will ensure that one loop has completed in its entirety via an expensive barrier before the next loop can be computed. But in reality, even where there is dependence between a pair of loops, it is often possible to do better than that. My group is working on a compiler that translates OpenMP to an equivalent code that makes calls to the SMARTS runtime system from Los Alamos National Laboratory [17]. The translation determines the dependences between chunks of parallel loops; chunks are sets of iterations that will be executed by a single thread. It creates a graph of the dependences between chunks. At run time, a chunk may be executed as soon as those it depends on have completed. This removes many of the barriers in a user

code and in general also improves concurrency. The translation process may also assign chunks to threads so that data in cache is reused across loops [19].

4 GLOBALIZATION OF HPC

High Performance Computing is a critical technology that is deployed in the design of many engineering products, from packaging for electronics components to aeroplanes and their component parts. It is also needed in the design of computers themselves, and in safety testing of automobiles and reactors. It is used in modern drug design and in the creation of synthetic materials. It has helped us investigate global climate change and understand how to preserve rare books. Many of the commercial packages that are used to solve such problems are European products. It is of great concern to their vendors that these applications efficiently exploit the platforms installed at their customer sites. For this to be the case, they must keep abreast of all developments in programming standards. Strong support for this has been forthcoming from the European Commission. There have been many HPC application development projects, but few that have had the industrial impact of EUROPORT. With 40+ partners from all over Europe, including many significant application providers, it is the most concerted effort to date to parallelize industrial applications. It led to the availability of MPI versions of STAR-CD, LS-DYNA, PAM-CRASH and many other significant codes.

Although the amount of money spent on HPC products is relatively static, its share of the computer market has declined significantly. As a result, high performance computing is now a truly global market in which hardware and software vendors alike are locked in an aggressive competition. American and Japanese companies have long dominated the hardware sector, yet they have increasingly become engaged in European research and development activities as they target this market. In addition to building fabrication plants, many of them now have research centers in Europe. As a result, it is not hard for Europeans to gain early access to technology, and they have paths for providing input to new developments. European researchers have always been strong in language design. They were involved in the definition of both MPI and HPF from the start and have had considerable influence on these APIs, e.g. [5]. However, it was hard for many of those interested to attend standards meetings held in the US and despite a few individual standardization meetings and workshops held in Europe, broad participation was restricted to input via (admittedly extensive) email discussions.

OpenMP is organized differently. The vendor organization that controls development and maintenance of the OpenMP standard, the OpenMP ARB, has

members located in the US, Europe as well as Japan. Business is conducted without travel and thus international participation is no longer limited by the location of meetings. Other than vendors, organizations with a strong interest in OpenMP may join this not-for-profit corporation and participate in the maintenance and further development of this standard.

A group of interested researchers has founded an organization that not only aims to promote interaction between those interested in OpenMP, its use and its further development, but also to discuss many of the issues raised in the previous section. Called Compunity, this community of OpenMP researchers, developers and users is an open forum, led by the author of this article that organizes a series of workshops on topics of interest to OpenMP. These are held in Europe, Asia and North America in an attempt to ensure that those interested have an opportunity to attend, no matter where they are located. Indeed, the very first of these meetings was organized in Europe under the auspices of the EC-funded EUROTOOLS project.

It is not surprising that research activities are also increasingly global. The US National Science Foundation and the European Commission have begun to request proposals for research and development activities that specifically call for international collaboration, including the exchange of students and course curricula between the two continents. Europeans have much to offer their American colleagues and it is to be hoped that the many existing informal collaborations will be augmented by formal cooperations of this kind. The focussed approach and vertical structure of typical European projects has supported the development of pragmatic and realistic solutions to many of the problems faced when deploying high performance computing technologies in practice.

5 SUMMARY AND OUTLOOK

Although the market for HPC products is no longer a major one, it is recognized to be the laboratory for the development of new methods and systems. Moreover, it is a pre-requisite for industrial competitiveness in key areas of industry, and thus knowledge and rapid deployment of innovations in this area is essential for companies that compete at the high end of science and technology.

The need for standards to support the programming of HPC applications has never been greater. Expertise in parallel application development is scarce and program codes cannot be constantly rewritten for new platforms. Recent developments in the way that HPC systems are deployed have increased the need for portable code: the emerging trend towards combining HPC resources across several sites into so-called computational grids provides

users with a multiplicity of available execution targets. Although such grids are seldom in operational use today, this will soon be the case. The NSF supercomputer centers in San Diego and Illinois, for example, have begun building such a grid to better serve their user base. It will not be practicable for a user to create custom versions of HPC codes for each potential target machine – programs must be portable and easily adaptable to each of the systems.

OpenMP is the simplest parallel programming API that is widely available and if it could be used by HPC application developers for all of their programming needs, the productivity of these experts would grow in both the short and the long term. It is to be hoped that researchers and industry will collaborate globally to extend the current OpenMP API so that it is able to assume the role of a standard API for High Performance Computing.

6 REFERENCES

- [1] C. Amza, A. Cox et al., "TreadMarks: Shared memory computing on networks of workstations", IEEE Computer 29(2), 18-28 (1996)
- [2] E. Ayguade, X. Martorell, J. Labarta, M. Gonzalez and N. Navarro, "Exploiting Multiple Levels of Parallelism in OpenMP: A Case Study", Proc. 1999 Int. Conf. on Parallel Processing, Aizu (1999)
- [3] R. Baxter, P. Graham and M. Bowers, "Rapid parallelization of the industrial modelling code PZFlex", Proc. EWOMP 2000, Edinburgh (2000)
- [4] J. Bircsak, P. Craig, R. Crowel, J. Harris, C.A. Nelson and C.D. Offner, "Extending OpenMP for NUMA Machines", Proc. Supercomputing 2000, Dallas (2000)
- [5] B. Chapman, P. Mehrotra and H. Zima, "Programming in Vienna Fortran", Scientific Programming 1(1):31-50 (1992)
- [6] B. Chapman, P. Mehrotra and H. Zima, "Enhancing OpenMP with Features for Locality Control", Proc. ECMWF Workshop Towards Teracomputing, Reading (1998)
- [7] HPF Forum, "High Performance Fortran language specification, Version 2.0", Rice University (1997)
- [8] Kuck and Associates. KAP/Pro toolset for OpenMP. See www.kai.com/kpts/
- [9] M. Leair, J. Merlin, S. Nakamoto, V. Schuster and M. Wolfe, "Distributed OpenMP – a programming model for SMP clusters", Proc. CPC2000, Compilers for Parallel Computers, Aussois, France (2000)
- [10] Message Passing Interface Forum, "MPI: A message-passing interface standard. Version 1.1", at www.anl.gov/mcs/ (1995)
- [11] D.S. Nikolopoulos, T.S. Papatheodorou, C.D. Polychronopoulos, J. Labarta and E. Ayguade, "Is Data Distribution Necessary in OpenMP?", Proc. Supercomputing 2000, Dallas (2000)
- [12] OpenMP Architecture Review Board, "OpenMP Fortran application program interface. Version 2.0", at www.openmp.org (2000)
- [13] M. Sato, S. Satoh, K. Kusano and Y. Tanaka, "Design of OpenMP compiler for an SMP cluster", Proc. EWOMP 99, Lund (1999)
- [14] S. Shah, G. Haab, P. Petersen and J. Throop, "Flexible Control Structures for Parallelism in OpenMP", Proc. EWOMP '99, Lund (1999)
- [15] L.A. Smith and J.M. Bull, "Development of mixed-mode MPI/OpenMP applications", Proc. WOMPAT 2000, San Diego (2000)
- [16] Top500 Organization, "The Top500 Supercomputers" and historical lists, at www.top500.org
- [17] S. Vajracharya, S. Karmesin, P. Beckman, J. Crotinger, A. Maloney, S. Shende, R. Oldehoeft and S. Smith, "Exploiting Temporal Locality and parallelism through vertical execution", Proc. ICS (1999)
- [18] A.J. Wallcraft, "SPMD OpenMP vs. MPI for ocean models", Proc. EWOMP, Lund (1999)

-
- [19] T-H. Weng and B. Chapman, "Implementing OpenMP using dataflow execution model for data locality and efficient parallel execution", Proc. HIPS02, to appear (2002)
- [20] X3H5 Committee, "Parallel Extensions for Fortran", TR X3H5/93-SD1 Revision M, ANSI accr. Stds. Ctte X3 (1994)