Available online at www.sciencedirect.com
SCIENCE DIRECT®
Parallel Computing 31 (2005) 957–959
PARALLEL COMPUTING
www.elsevier.com/locate/parco

Guest editorial

OpenMP

Computer vendors increasingly rely on shared memory parallelism to raise the computational power of their hardware. Most of them market shared memory multiprocessors (SMPs). Machines with a low CPU count may be "symmetric" (the term SMP originally stood for Symmetric Multiprocessor), which means that the cost of a memory access is the same no matter which CPU (or thread) performs this operation; larger shared memory machines typically provide Non-Uniform Memory Access (NUMA) and this may affect a program's performance.

This industrial trend is undergoing further acceleration, as most vendors are now building either Simultaneous Multithreaded CPU cores or chips sporting multiple cores on the same silicon die. In the former case, multiple sets of registers permit several threads to issue instructions for execution simultaneously so that they will be interleaved. In the latter, most features are replicated on the chip, but the CPUs may share some levels of cache. Of course, these features may also be combined. It is conceivable that, in a not so distant future, single CPU computers will be an exception, not the norm.

But in contrast to established technologies such as software pipelining and instruction level parallelism, it is hard for a compiler to automatically detect independent sequences of instructions that are long enough to exploit this kind of architectural parallelism. Some amount of programmer involvement appears to be necessary. But on the other hand, even though some discussion of concurrency is a mandatory part of every computer science and software engineering curriculum, most software developers do not gain the experience needed to tame the subtleties of concurrent programming. Thus it was imperative that an application programming interface (API) be provided that facilitates the use of this hardware for a broad variety of applications. That API is OpenMP. Developed by a consortium of vendors that includes almost all of those that provide such hardware, OpenMP is both easy to use as well as powerful enough for deployment on very large platforms. It is successfully deployed on dual-CPU platforms as well as on large-scale distributed memory machines with over a thousand CPUs.

0167-8191/$ - see front matter
doi:10.1016/j.parco.2005.08.001

OpenMP primarily consists of a set of compiler directives (pragmas) that are inserted into a Fortran, C or C++ program to convey information to an OpenMP compiler. The compiler uses this additional information to create an explicitly parallel version of the program. Fortunately, most standard Fortran and C compilers are able to translate these directives; thus the API is very widely available. As a result of this, and its relative simplicity, many applications now include OpenMP directives and library routines to exploit shared memory parallelism in computers. Considering the sophistication of the underlying software technology, this simplicity and the consequent diffusion of OpenMP constitute an outstanding achievement.

It is a well-known fact that applications significantly outlive computer architectures, as well as most other kinds of software. At the same time, the success and spread of a programming interface triggers interest in fields not originally targeted. More rarely, but inevitably, new applications and algorithms arise. Thus it is imperative that any widely used programming interface is widely available, well maintained and suitable for implementation on a variety of different kinds of hardware. Fortunately, OpenMP is actively maintained by a consortium of hardware vendors and compiler companies that have collectively formed the OpenMP Architecture Review Board (ARB) to ensure that the language continues to meet the needs of a broad swath of user communities. Several major users and a consortium of researchers (cOMPunity) also contribute significantly to the work of the ARB.

Although it is often straightforward to add OpenMP directives to an existing program, it requires much more effort to obtain high levels of performance. A pioneering spirit is needed if it is used to create code for execution on new kinds of hardware.

Advances in compiler technology, and careful language design, are required to increase the expressivity of the language without making it harder to use, to improve its modularity and to increase its ability to support scalable application development.

Feedback from applications experts is needed to enlarge the scope of its applicability, particularly when deployed on new kinds of hardware or to parallelize algorithms that are not usually executed on parallel platforms. Tools are needed to help programmers create OpenMP programs rapidly, to analyze and improve the performance of OpenMP code, and in particular to help programmers find bugs in their code. Work is needed to develop the compiler and system software technology that will enable the use of this API on hardware that does not provide shared memory. In short, much research is needed to keep OpenMP relevant.

We are pleased to be able to present a variety of papers in this Special Issue that represent some of the on-going activities in this field. The contributions include three papers that consider language features and their implementation. Two of these papers discuss the need for, and then the implementation of, nested parallelism. This OpenMP feature was not implemented in most of the early compilers but is important, since it enables modularity and can help create scalable code. The third introduces task queueing features and is discussed further below. Two papers discuss tools: one of these gives an overview of a sophisticated effort to provide support for the generation of efficient OpenMP programs. The other describes work to predict

the performance of code written using both the Message Passing Interface (MPI) and OpenMP. Four papers in this collection describe a variety of experiences using OpenMP to parallelize several distinct kinds of applications: a branch and bound computation, a finite element solver that is deployed on the Japanese Earth Simulator, agent-based models that are applicable to application areas such as biology and finance, and algorithms for image and video coding.

Finally, we have included four papers that represent on-going work to extend the range of applicability of OpenMP. Two of these represent the state of the art in technology for implementing OpenMP on clusters. The remaining two papers discuss a translation of OpenMP, along with additional language and run-time features, to support the execution of master–slave kinds of computations. OpenGR uses OpenMP to help realize remote procedure calls for grid computing; the final paper introduces support for tasks to OpenMP, and extends its run-time library to provide additional information that enables OpenMP code to exploit the structure of a cluster of SMPs.

These papers discuss just some of the on-going work in research laboratories and universities that aims to improve our ability to write and run efficient shared memory parallel programs on a variety of architectures. Some of them, in exploring new ideas, implicitly or explicitly create new challenges. Unfortunately, OpenMP is often seen simply as a finished ''product'' by the research community and it is our opinion that insufficient attention is paid to the problem of improving this API.

OpenMP is the first successful directive-based API for parallel programming that is intended for general-purpose computing. It will not remain a success unless the research community attends to these challenges.

Barbara M. Chapman
Department of Computer Science
University of Houston
Houston, TX 77204-3010
USA
E-mail address: chapman@cs.uh.edu

Federico Massaioli
CASPUR
Via dei Tizii 61b, 00185 Roma
Italy
E-mail address: federico.massaioli@caspur.it

Available online 21 October 2005