Int J Parallel Prog (2007) 35:437–439
DOI 10.1007/s10766-007-0055-0

Introduction

Eduard Ayguadé · Matthias S. Mueller

Received: 6 November 2006 / Accepted: 26 January 2007 / Published online: 20 July 2007

OpenMP is an Application Programming Interface (API) widely accepted as a standard for high-level shared-memory parallel programming. OpenMP is a portable, scalable programming model that provides a simple and flexible interface for developing shared-memory parallel applications in Fortran, C, and C++. Since its introduction in 1997, OpenMP has gained support from the majority of high-performance compiler and hardware vendors. There is also active research in OpenMP compilers, runtime systems, tools, and environments. Under the direction of the OpenMP Architecture Review Board (ARB), the OpenMP standard continues to evolve.

The community of OpenMP researchers and developers in academia and industry is organized under cOMPunity, a forum for the dissemination and exchange of information about and experiences with OpenMP. IWOMP is the consolidation of three OpenMP workshops: the European Workshop on OpenMP (EWOMP), the Workshop on OpenMP Applications and Tools (WOMPAT), and the Workshop on OpenMP Experiences and Implementation (WOMPEI) that took place on several occasions starting in 1999.

Two consecutive issues of the International Journal of Parallel Programming have been devoted to extend the best papers that were presented at the second International Workshop on OpenMP (IWOMP), celebrated in Reims (France), 12–15 June 2006.

In this second issue, we have included those papers showing the experience on using OpenMP to parallelize applications, the performance benefits as well as possible

E. Ayguadé (✉)
Barcelona Supercomputing Center, Barcelona, Spain
e-mail: eduard.ayguade@bsc.es

E. Ayguadé
Technical University of Catalunya, Barcelona, Spain

M. S. Mueller
Technical University of Dresden, Dresden, Germany

language extensions to widen the applicability and performance of OpenMP. The first paper, "High-Scalability Parallelization of a Molecular Modeling Application—Performance and Productivity Comparison Between OpenMP and MPI Implementations" by Russell Brown and Ilya Sharapov, describes the parallelization of the Born calculation. This application solves the interactions between a protein and the surrounding water molecules, for both the OpenMP and MPI environments. This is a force-field calculation which is very amenable to parallel execution. The scalability on a symmetric multiprocessor is shown for up to 144 processors, concluding that in all cases OpenMP performs better than MPI, with less programming effort.

The second paper, "Nested Parallelization with OpenMP" by Dieter an Mey, Samuel Sarholz, and Christian Terboven, reports experiences on the use of nested parallelism in three production codes: a C++ code for content-based image retrieval, a C++ code for the computation of critical points in multi-block CFD datasets, and a multi-block Navier-Stokes solver written in Fortran90. The paper also discusses the opportunities, as well as the deficiencies, of the current nested parallelization support in OpenMP.

The third paper, "Dynamic Data Migration for Structured AMR Solvers" by Markus Nordén, Henrik Löf, Jarmo Rantakokko and Sverker Holmgren, reports on the performance of a parallel PDE solver written in OpenMP with adaptive mesh refinement, which makes dynamic load balancing necessary. Due to the dynamically changing memory access pattern caused by the runtime adaption, the paper addresses the challenging task of achieving a high degree of co-location of threads and data. The solution proposed is based on the use of dynamic page migrations of misplaced data and the use of a migrate-on-next-touch directive whose implementation has low overhead.

Finally, "OpenMP Implementation of SPICE3 Circuit Simulator" by Tien-Hsiung Weng, Ruey-Kuen Perng and Barbara Chapman describes the experience of creating an OpenMP implementation of the SPICE3 circuit simulator program, which has needed to address the irregular patterns of access to dynamic data structures in the SPICE code. Authors present two implementations, one with minimal code modification and one without modification to the original SPICE3 program using Intel taskq construct. Based on their experience, authors also discuss what future compiler tools need to include helping OpenMP application developers to successfully address similar porting goals.

In the previous issue of the Journal, we included papers with proposals to the current definition of the programming model. The first paper, "Complete Formal Specification of the OpenMP Memory Model" by Greg Bronevetsky and Bronis R. de Supinski, focused on the formal verification of OpenMP programs through a formal memory model. The paper provides a two-step process to verify whether an observed OpenMP execution is conformant. The model covers the entire specification and discusses ambiguities in the description of the current memory model.

The second paper, "A Proposal for Error Handling in OpenMP" by Alejandro Duran et al., addressed one important lack in the current OpenMP language specification: reliability and recovery mechanisms. This is important when OpenMP is used to target non-numerical applications, like Web servers. In order to add error handling to OpenMP the authors propose a directive and a clause to define a scope for the error handling (where the error can occur) and specifying a behavior for handling the specific errors.

The last paper in the previous issue, "Supporting Nested OpenMP Parallelism in the TAU Performance System" by Alan Morris, Allen D. Malony and Sameer S. Shende, described the problem that is found in performance measurement tools that must determine thread context to properly maintain per-thread performance data, and a novel solution for identifying threads uniquely in the context of OpenMP nested parallelism. The approach proposed by the authors is implemented in the TAU performance system and has been successfully used in profiling and tracing OpenMP applications with nested parallelism. Authors propose extensions to the OpenMP standard that can help tool developers to uniquely identify threads.

There is still much to learn about OpenMP, its implementation on a variety of platforms, ranging from multi- and many-core chips to very large shared-memory platforms, and its use in the emerging applications that will run on these systems. We hope that this collection of papers will prove to be interesting and useful to readers, and that the issues raised will stimulate further research in this area.