Scanning the Issue

Special Issue on Program Generation, Optimization, and Platform Adaptation

This special issue of the PROCEEDINGS OF THE IEEE offers an overview of ongoing efforts to facilitate the development of high-performance libraries and applications for engineers and scientists.

The fast evolution, increasing complexity, and increasing diversity of today's computing platforms pose a major challenge to developers of high-performance applications: software development has become an interdisciplinary task that requires the programmer to have specialized knowledge in algorithms, programming languages, and computer architectures. Furthermore, tuning even simple programs tends to be expensive because it requires an intense and sustained effort—which can stretch over a period of weeks or months, if not years—from the technically best expert programmers. But the manual tuning or adaptation of software implementations to a particular platform also leads to a vicious cycle: the code developer invests tremendous energy tinkering with the implementation to exploit in the best way the computing resources available, simply to realize that the hardware infrastructure has become obsolete in the interim as an effect of the relentless technological advances reflecting Moore's Law. This has led to large repositories of applications that were once well adapted to the existing computers of the time, but continue to persist because the cost involved in updating them to the current technology is simply too large to warrant the effort. To break this cycle successfully, it is necessary to rethink the process of designing and optimizing software in order to deliver to the user the full power of optimized implementations on the available hardware resources.

This special issue of the PROCEEDINGS presents an overview of recent research on new methodologies for the design, development, and optimization of high-performance software libraries and applications.

The special issue contains 13 invited papers grouped into four main areas:

- program generators;
- parallel library design;
- domain-specific compiler optimizations;
- dynamic (runtime) software adaptation.

Digital Object Identifier 10.1109/JPROC.2004.840488

We now provide a brief overview of each of these areas and of the papers in each group.

Program generators: The first group of four papers adopts as basic strategy to automatic performance tuning and optimization the generation of routines or complete applications starting with either mathematical descriptions or program templates. The formulas representing the space of possible algorithms can be derived from an initial set of formulas by applying algebraic rules and mathematical theorems. Alternatively, the space of possible implementations can be derived from a program template by applying well-known code transformations such as loop unrolling, loop tiling, and statement reordering.

To obtain an optimal or near-optimal algorithm implementation, the space of possible algorithms and/or implementations is searched. The search can be exhaustive, based on heuristics, or guided by intelligent mechanisms. The function to be optimized can be the actual execution time measured on a real machine or the predicted execution time based on: 1) a functional description of the target machine or 2) analytical models of the target architecture.

The first paper in this area, "The Design and Implementation of FFTW3," by Frigo and Johnson, describes the latest version of FFTW, a widely used library for computing the discrete Fourier transform (DFT) in one and multiple dimensions. FFTW is highly optimized and, furthermore, adapts to the underlying hardware to maximize performance. This is achieved through FFTW's architecture, which implements a flexible compositional framework that decomposes large DFT problems into smaller problems. FFTW supports a relevant set of alternative decompositions, which are searched to find the best match or adapt to the platforms memory hierarchy. Small DFT problems are computed through so-called codelets consisting of highly optimized straightline code (code without loops and control structures) automatically generated by a special purpose compiler. The paper shows that this approach yields a performance that matches libraries that are hand optimized for a particular machine. Beyond the DFT, FFTW serves as a good case study on how to build a self-adaptable numerical library.

The second paper in this area, "SPIRAL: Code Generation for DSP Transforms," by Püschel *et al.*, describes a code generator for digital signal processing (DSP) transforms. These

include the DFT, various discrete trigonometric transforms, filters, and the discrete wavelet transform (DWT). SPIRAL generates its code from scratch. It is highly optimized and tuned to the given computing platform. SPIRAL formulates the implementation tuning as an optimization problem and exploits the domain-specific mathematical structure of discrete signal processing transform algorithms to implement a feedback-driven solver. For a specified transform, SPIRAL autonomously generates and explores alternative algorithms and their implementations and, by measuring their performance, searches among the many alternatives or learns directly how to create the "best" implementation. At the heart of SPIRAL is the signal processing language (SPL), which uses a small number of primitive symbols and operators (e.g., tensor product or direct sum) to compactly describe a large class of algorithms. Further, this compact description is used in SPIRAL to optimize algorithms at the high, algorithmic level, thus overcoming many of the limitations of current compilers. The code generated by SPIRAL is competitive with the best available hand-tuned libraries.

The third paper in this area, "Synthesis of High-Performance Parallel Programs for a Class of *Ab Initio* Quantum Chemistry Models," by Baumgartner *et al.*, presents a program generator for a class of quantum chemistry problems that are described as tensor contractions. Starting from a high-level mathematical description of the computation, the generator produces high-performance code that is tuned to the characteristics of the computing platform. The optimization of the code starts at the mathematical level, where the input expression is formally manipulated to minimize computational cost. Then, memory requirements are minimized and the data is partitioned for the parallel system. Finally, the actual code is generated; various code types are supported. The system described in the paper provides a solution in a domain where manual optimization is not only difficult, but often practically infeasible.

The fourth paper in this area, "Self-Adapting Linear Algebra Algorithms and Software," by Demmel *et al.*, discusses automatic performance tuning for dense and sparse linear algebra computations. The authors describe general ideas for the design of adaptable numerical software. Then, they explain how these ideas are instantiated in LAPACK, a widely used library for dense linear algebra problems. LAPACK is implemented on top of the computationally most expensive kernel routines, which are provided by a different library or application programming interface (API) called BLAS (an acronym for "basic linear algebra subroutines"). The BLAS, in turn, is generated and tuned for every platform by a code generator called ATLAS, which performs an empirical search over different coding options such as block sizes and the degree of instruction interleaving. Further, the authors discuss automatic performance tuning for sparse linear algebra problems. In contrast to dense linear algebra, adaptation in this domain has to take partially place at runtime. The reason is the structure of sparse matrices, which is not available at compiler time, but crucially determines the performance of different storage schemes.

Parallel library design: The second group of two papers targets the problem of high performance library design for parallel architectures. Well-designed interfaces, supported by object-oriented features of the base language, facilitate the use of these libraries.

The first paper in this area, "Parallel VSIPL++: An Open Standard Software Library for High-Performance Parallel Signal Processing," by Lebak *et al.*, describes the Parallel Vector, Signal, and Image Processing Library (Parallel VSIPL++). The goal of Parallel VSIPL++ is to enable the user to write programs for parallel real-time embedded signal processing while still maintaining both portability and high performance across platforms. This is achieved through an object-oriented framework that enables the mapping of data and functions onto parallel hardware. To achieve high performance, Parallel VSIPL++ supports various adaptation mechanisms—for example, at compile time, hardware specialization of functions by the compiler, or, at runtime, through explicit run stages to accelerate communication operations. Parallel VSIPL++ is developed to comply with community-defined interfaces and is compatible with many proprietary high-performance embedded computing platforms.

The second paper in this area, "Parallel MATLAB: Doing it Right," by Choy and Edelman, discusses an extension of MATLAB for parallel computing. The main idea is conceptually simple and elegant. A single operator, $*p$, is used to specify which arrays are to be distributed. The MATLAB functions that manipulate arrays are overloaded so that parallel versions are invoked when the parameters are distributed objects. This approach enables the development of parallel programs that are easy to read and debug because they look very much like conventional MATLAB programs and all the parallelism is encapsulated within MATLAB functions.

Domain-specific compiler optimizations: The third group of four papers adopts a different strategy to code optimization and tuning. They develop new compiler frameworks to support optimizations that work effectively for certain classes of computations and then apply them to conventional languages, such as Fortran and C. The constructs to be optimized are identified via pattern matching or by annotations. The transformations can be fully automatic or make use of profiling information and directives. These could, for example, specify the layout and distribution of data structures, semantic information that is difficult to infer from the source program, or the range of values assumed by some variables.

Compiler techniques for the optimization of domain-specific language constructs have also been developed. These techniques take advantage of semantic information about these domain-specific constructs.

The first paper in this area, "Broadway: A Compiler for Exploiting the Domain-Specific Semantics of Software Libraries," by Guyer and Lin, describes the Broadway compiler, which is designed to perform domain-specific optimizations on conventional languages. The basic assump-

tion is that library routines can be seen as domain-specific operations that extend the base language. The Broadway compiler is not designed for any specific domain, but is expected to be applicable to a range of domains. To this end, the compiler uses generic analysis and optimization algorithms that rely on annotations describing the input–output and the behavior of each routine in the library. The annotations also specify the condition under which code replacements and inlining can be applied. Thus, the annotation file is a mechanism that specializes the Broadway compiler to the domain of the annotated library. The paper presents as a case study the annotations that were used in the optimization of PLAPACK programs and presents performance results for three such programs.

The second paper in this area, by Yotov *et al.*, asks the question: "Is Search Really Necessary to Generate High-Performance BLAS?" The paper considers the BLAS code generator of ATLAS. The original ATLAS generates the BLAS code by empirical search. It tries various code variants with different degrees of blocking, loop unrolling, instruction interleaving, among others, according to a specific search strategy and selects the best as final output. The authors replace the empirical search by analytical models that are used to compute, rather than search, the parameters. Experiments show that the model-based approach can perform as well as the empirical approach in most cases. The paper may provide a first step toward future compilers that use knowledge of the underlying computing platform to a much larger extent and in more sophisticated ways than current compilers.

The third paper in this area is "Telescoping Languages: A System for Automatic Generation of Domain Languages," by Kennedy *et al.* It describes a novel strategy to generate compiler optimization passes capable of manipulating codes containing invocations to library routines. The core of the strategy is implemented in Palomar, a sophisticated module that accepts user-specified transformations, generates transformation rules, and analyzes library routines to generate multiple versions that match the transformation rules and are then made available to the program optimizer. The paper presents several application studies: MATLAB for signal processing, library maintenance using LibGen, computationally intensive statistics, image processing, and component integration.

The fourth paper in this area, "Efficient Utilization of SIMD Extensions," by Franchetti *et al.*, addresses the problem of compiling a numerical program to take optimal advantage of short vector single-instruction multiple-data (SIMD) instructions. These instructions are a recent addition to most available computer architectures and enable processors to perform several integer or floating pointing operations as fast as a single operation. Compiler support for these instructions is rather limited, which motivates the two domain-specific approaches presented in this paper. The first approach is a back-end compiler that extracts the fine-grain parallelism for vector instructions from a dataflow representation of a straightline program. The second approach, called formal vectorization, targets specifically signal transforms. Vectorization is achieved in this case by manipulating a mathematical representation of the algorithm to make explicit vector parallelism. Experiments show the success of both methods.

Dynamic (runtime) software adaptation: Finally, the fourth group of three papers uses runtime information to attempt, in some cases, to significantly improve the optimization of applications and libraries over what is possible by relying exclusively on static information. The runtime information can be used to perform on-the-fly optimization or used across several executions for long periods to progressively improve the code and enable its adaptation to changing runtime environments and input data sets.

The first paper in this area, by Reed and Mendes, is called "Intelligent Monitoring for Adaptation in Grid Applications." Grid computing takes place in an environment in which users can simply and transparently access distributed computers and databases as if they were a single system. Grid computing offers great potential for large-scale distributed application; however, ensuring and adapting performance for the grid poses problems rather different from classical platforms, due to shared resources and their unpredictable dynamic behavior. This paper offers a solution based on performance contracts and contract monitors. Performance contracts formalize and specify the relationship between application performance and resources. During execution, the contract monitors continuously verify that the contract is met; if not, the application can be readapted or rescheduled. Further, the contracts use "soft control" based on fuzzy logic to ensure robustness in decision making on the grid resources, which have an inherent high performance variability. A representative application example shows the viability of the approach.

The second paper in this area, "Design and Engineering of a Dynamic Binary Optimizer," by Duesterwald, presents an overview of dynamic binary optimizers. It discusses their overall organization and the techniques they apply. Dynamic binary optimizers inspect and manipulate programs during execution in order to improve the performance of the executing code and perhaps translate the code onto the native language of the target machine. The main advantage of dynamic optimizers over their static counterparts is that they can take advantage of information only available at execution time such as the value of input data and the frequency of execution of code fragments. This enables the application of optimizations that would in general be impossible to carry out statically. The paper discusses techniques to deal with one of the main challenges in the design of a dynamic binary optimizer: namely, the need to minimize the overhead ensuing from the need to maintain control over the executing code.

The third paper in this area, "A Survey of Adaptive Optimization in Virtual Machines," by Arnold *et al.*, gives a comprehensive overview on the history and state of the art in optimization for virtual machines (VMs), which are software programs that execute high-level programming languages. The most prominent example is the widely used Java Virtual Machine that executes Java programs. VM

architectures provide several advantages over static libraries, including portable program representations, dynamic program composition, and enhanced security. However, VMs face performance challenges, since most optimizations have to be deferred to runtime. The paper explains in detail the various optimization strategies that have been developed in the past three decades, including selective optimization techniques that apply an optimizing compiler at runtime to critical program components and feedback-directed optimization techniques that use dynamically collected profiling information to improve performance.

The editors take this opportunity to thank the approximately 75 authors and the more than 50 reviewers who survived and labored through several rounds of reviews to shape and "tune" (manually) the papers and the special issue.

JOSÉ M. F. MOURA, *Guest Editor*
Department of Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA 15213-3890 USA

MARKUS PÜSCHEL, *Guest Editor*
Department of Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA 15213-3890 USA

DAVID PADUA, *Guest Editor*
3318 Digital Computer Laboratory
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801 USA

JACK DONGARRA, *Guest Editor*
Computer Science Department
University of Tennessee
Knoxville, TN 37996-3450 USA

José M. F. Moura (Fellow, IEEE) received the engenheiro electrotécnico degree from Instituto Superior Técnico (IST), Lisbon, Portugal, in 1969 and the M.Sc., E.E., and D.Sc. degrees in electrical engineering and computer science from the Massachusetts Institute of Technology (MIT), Cambridge, MA, in 1973, 1973, and 1975, respectively.

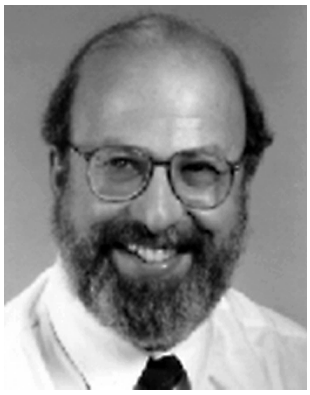
He has been a Professor of Electrical and Computer Engineering at Carnegie Mellon University, Pittsburgh, PA, since 1986. He held visiting faculty appointments with MIT (1984–1986 and 1999–2000) and was on the faculty of IST (1975–1984). His research interests include statistical and algebraic signal and image processing and digital communications. He has published over 270 technical journal and conference papers, is the coeditor of two books, and holds six U.S. patents. He currently serves on the Editorial Board of the *ACM Transactions on Sensor Networks* (2004-).

Dr. Moura is a Corresponding Member of the Academy of Sciences of Portugal (Section of Sciences). He was awarded the 2003 IEEE Signal Processing Society Meritorious Service Award and the IEEE Millennium Medal. He has served the IEEE in several positions, including Vice-President for Publications for the IEEE Signal Processing Society (SPS) (2000–2002), Chair of the IEEE TAB Transactions Committee (2002–2003), *Editor-in-Chief* for the IEEE TRANSACTIONS ON SIGNAL PROCESSING (1975–1999), and Interim Editor-in-Chief for the IEEE SIGNAL PROCESSING LETTERS (December 2001–May 2002). He currently serves on the Editorial Boards of the PROCEEDINGS OF THE IEEE (2000-) and the *IEEE Signal Processing Magazine* (2003-).

Markus Püschel received the Diploma (M.Sc.) degree in mathematics and the Ph.D. degree in computer science from the University of Karlsruhe, Karlsruhe, Germany, in 1995 and 1998, respectively.

From 1998 to 1999, he was a Postdoctoral Researcher in the Department of Mathematics and Computer Science, Drexel University, Philadelphia, PA. Since 2000, he has held a Research Faculty position in the Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA. He was a Guest Editor of the *Journal of Symbolic Computation*. His research interests include scientific computing, compilers, applied mathematics and algebra, and signal processing theory/software/hardware. More details can be found at http://www.ece.cmu.edu/~pueschel.

Dr. Püschel is on the Editorial Board of the IEEE SIGNAL PROCESSING LETTERS and was a guest editor of the PROCEEDINGS OF THE IEEE.



David Padua (Fellow, IEEE) received the Ph.D. degree in computer science from University of Illinois, Urbana-Champaign, in 1980.

He is a Professor of Computer science at the University of Illinois, Urbana-Champaign, where he has been a faculty member since 1985. At Illinois, he has been Associate Director of the Center for Supercomputing Research and Development, a member of the Science Steering Committee of the Center for Simulation of Advanced Rockets, Vice-Chair of the College of Engineering Executive Committee, and a member of the Campus Research Board. His areas of interest include compilers, machine organization, and parallel computing. He has published more than 130 papers in those areas. He has served as Editor-in-Chief of the *International Journal of Parallel Programming* (IJPP). He is a Member of the Editorial Boards of the *Journal of Parallel and Distributed Computing* and IJPP.

Prof. Padua has served as a Program Committee Member, Program Chair, or General Chair for more than 40 conferences and workshops. He served on the Editorial Board of the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS. He is currently Steering Committee Chair of ACM SIGPLAN's Principles and Practice of Parallel Programming.

Jack Dongarra (Fellow, IEEE) received the B.S. degree in mathematics from Chicago State University, Chicago, IL, in 1972, the M.S. degree in computer science from the Illinois Institute of Technology, Chicago, in 1973, and the Ph.D. degree in applied mathematics from the University of New Mexico, Albuquerque, in 1980.

He worked at the Argonne National Laboratory until 1989, becoming a Senior Scientist. He is currently a University Distinguished Professor of Computer Science, Computer Science Department, University of Tennessee, Knoxville. He also has the position of Distinguished Research Staff member in the Computer Science and Mathematics Division at Oak Ridge National Laboratory (ORNL), Oak Ridge, TN, and is an Adjunct Professor, Computer Science Department, Rice University, Houston, TX.

He specializes in numerical algorithms in linear algebra, parallel computing, the use of advanced computer architectures, programming methodology, and tools for parallel computers. His research includes the development, testing and documentation of high quality mathematical software. He has contributed to the design and implementation of the following open-source software packages and systems: EISPACK, LINPACK, the BLAS, LAPACK, ScaLAPACK, Netlib, PVM, MPI, NetSolve, Top500, ATLAS, and PAPI. He has published approximately 200 articles, papers, reports and technical memorandums, and he is coauthor of several books.

Prof. Dongarra is a Fellow of the American Association for the Advancement of Science (AAAS) and the Association for Computing Machinery (ACM) and a Member of the National Academy of Engineering.