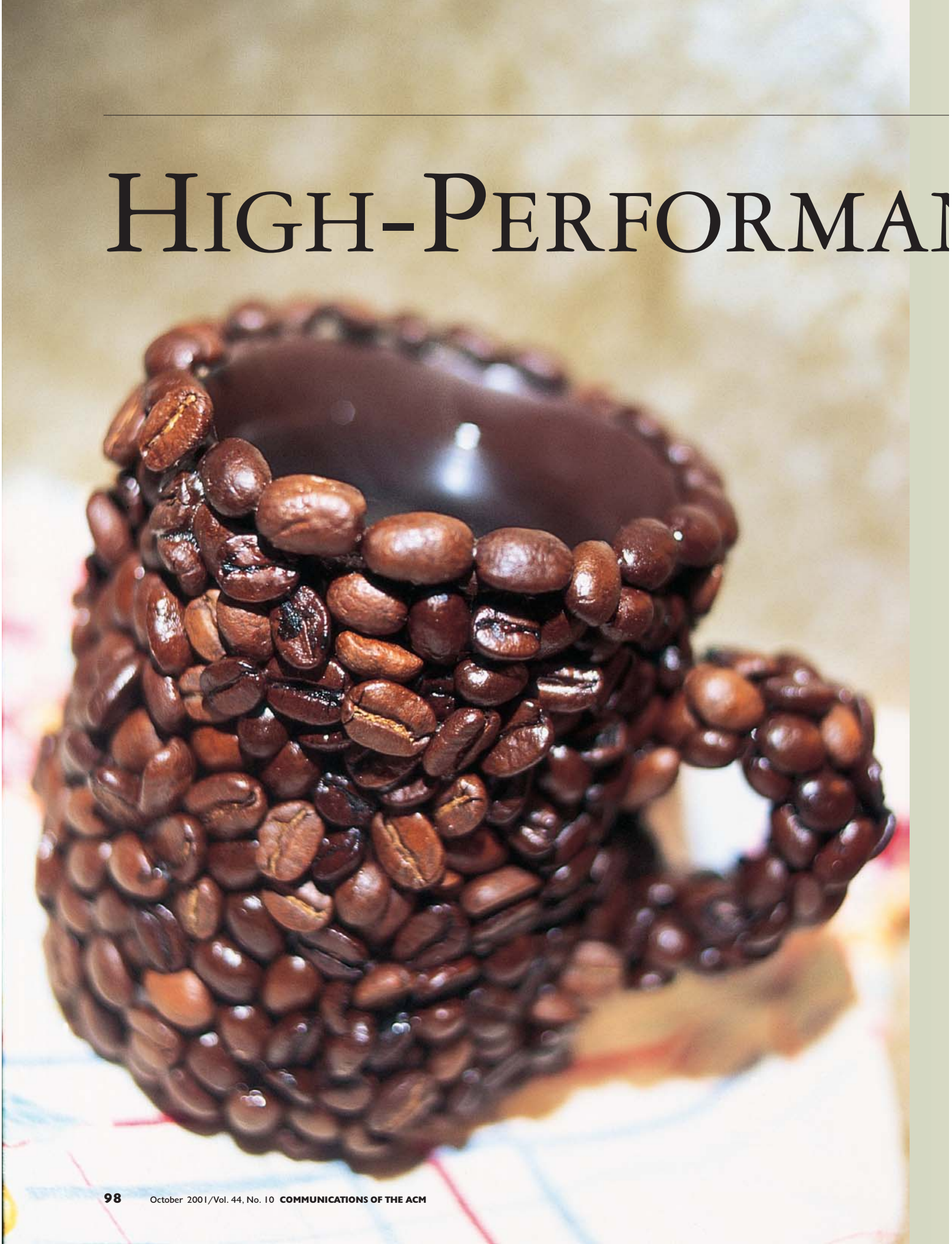
High-Performan



NCE

JAVA

❖ CHERRI M. PANCAKE AND CHRISTIAN LENGAUER, GUEST EDITORS

Despite the popular view that Java implementations are inefficient (compared to their Fortran, C, and C++ counterparts), high-performance applications, which demand execution efficiency, are finding ways to exploit Java that benefit all Java implementations.

Over the past six years, the Java programming language has captured the attention of a broad audience of programmers, software engineers, and network application users alike, largely because of its features for interactive and Internet programming. Often viewed as the modern alternative to C++, Java also has the advantage of including explicit mechanisms for parallelism and coordination, making it a natural language for distributed computing.

Java is commonly perceived as being execution-inefficient, despite the fact that inefficiency is a property of language implementations, not of the language per se. Early implementations were based on a Java Virtual Machine (JVM) that interpreted unoptimized "bytecode" in a relatively unsophisticated way. Recent developments in compilation technology (adding static analysis capabilities and just-in-time compilation) and instruction-level optimizations have done away with many sources of execution inefficiency. These days, Java is competitive with C and C++ for some applications on some platforms and is considerably safer to execute and easier to program.

As Java implementations have improved, interest has focused on its possible use for high-performance computing, or HPC. HPC applications place extreme demands on execution efficiency, requiring not just sophisticated compilers but highly tuned runtime environments that exploit the characteristics of each host architecture.

A large majority of today's Java programmers might well ask: Why is it important to push for a high-performance version of Java? The quest for more efficient Java implementations should be of interest to the broad Java community, if for no other reason than the fact that the most frequent Java criticism is the relative inefficiency of both its past and current implementations (compared with those of Fortran, C, and C++). That initial limitation has been the basis for recent developments in static analysis, just-in-time compilation,

CAREN ROSENBLATT

and JVM optimization. In pursuing high-performance Java, the HPC community is adding to these efforts with innovative work on faster remote method invocation and the exploitation of high-speed networks to improve performance—work that will ultimately benefit all implementations of Java.

Moreover, non-HPC-minded programmers should recognize how increasing emphasis on performance will likely change the Java language itself. Over the past few years, a consortium called the Java Grande Forum (www.javagrande.org) has become active in representing the interests of the HPC community to Java developers. In many cases, the features needed for HPC are being expressed in class libraries, such as the ones designed to provide efficient support for multidimensional, rectangular arrays. Other proposals extend the base language, including:

Floating-point computations with hardware support. A new modifier, `strictfp`, enforces (locally) the bit-reproducible behavior prescribed globally by the original Java semantics but which is not supported by current floating-point hardware.

Machine-specific performance optimizations. Another new modifier, `fastfp`, would indicate that the compiler may optimize arithmetic expressions by using machine-specific instructions, a practice not permitted by Java's current specification.

The Java Grande Forum succeeded in getting `strictfp` adopted into the Java 1.2 specification, and a Java Specification Request for the adoption of `fastfp` is under review. A current debate in the HPC community concerns how complex numbers should be added to the language. These extensions should be of interest to many programmers outside the HPC community as well.

HPC programmers might wonder: Why should Java be proposed as a vehicle for HPC? HPC applications have typically been developed in Fortran and C, whose compilers are robust and reflect decades of research in optimization techniques. Java offers several potential advantages; the most apparent is access to many new resources, not least of which are a wide selection of class libraries and a growing number of trained programmers. One programming approach is simply to mimic traditional HPC techniques in Java, making the compiler produce essentially the same type of target code expected of a Fortran, C, or C++ compiler. Some organizations are doing this to take advantage of a new generation of programmers whose preferred language is Java.

A more ambitious approach is to develop new HPC techniques to do things that could not be done easily with more traditional languages. For example, one distinct Java advantage is support for secure, portable, dynamic target code. This makes it a promising vehicle for irregular applications, such as those requiring thread-based parallelism or runtime compilation and optimization. But can Java deliver the expected execution efficiency? In its present form, it does so only to a limited extent. Many of the related performance obstacles are due to current implementations, which are still somewhat immature, while others are imposed by Java's semantics.

The most apparent Java advantage is access to new resources, including a wide selection of class libraries and a growing number of trained programmers.

How can Java be made more suitable for HPC? Should high-performance applications adapt to the limitations of the language, or should the language be extended to improve its suitability? If the former, how can we tune Java implementations so they exploit the performance potential of the language to the fullest? If the latter, what form should these extensions take? Should the addition/modification of language features be permitted, or should HPC developers (try to) restrict themselves to class library extensions?

These and similar questions were explored by 35 researchers at a week-long seminar last year at Schloss Dagstuhl, Germany. Afterward, they were invited to team up with various research groups to address what has been done, and what should still be done, to make Java more appropriate for HPC. This issue includes three of the joint articles that resulted. (Another, "Java and Numerical Computing," by R. Boisvert et al., was published in *Computing in Science and Engineering*, Mar./Apr. 2001).

Java's suitability for high-performance scientific computations. The salient design problems are the

absence of rectangular, multidimensional arrays in Java, and the fact that, in Java, complex numbers and arrays are objects, rather than primitive data types. This design choice incurs storage allocations and frequent runtime checks due to Java's semantics enforcing a high level of security. While the recent introduction of `strictfp` alleviates problems with the initial too-precise floating-point semantics, Java's equally precise exception model remains a challenge to high performance. Moreira et al. explore the related questions and propose solutions involving the addition of new Java packages and compilation techniques.

Java's ability to achieve high-performance communication rates in a distributed environment. Today's most popular paradigm for distributed computing—single-program-multiple-data (SPMD) parallelism with message passing—does not integrate well with Java's OO model. However, using multiple JVMs executing in parallel can serve to efficiently implement Java's own communication mechanism—remote method invocation in multithreaded programs operating on a distributed shared memory. Kielmann et al. compare two such JVMs: Hyperion and Manta. And Getov et al. explain how Java's diverse forms of communication can be used to advantage for large distributed applications involving the computational Grid—the emerging wealth of interlinked, heterogeneous computing resources on the Internet.

The clearest lesson from these articles, as well as from the seminar, is that Java has significantly more potential for HPC than is commonly believed. True, when Java is used and implemented naively, there can be problems with efficiency, but they are not inherent in the language itself and therefore can be overcome. Exploiting Java's full potential for parallel and distributed computing requires efficient JVMs and high-performance communications.

As the three articles here demonstrate, Java for high-performance computing is on the horizon. C

Cherri Pancake (pancake@cs.orst.edu; pancake@nacse.org) is a professor of computer science and Intel Faculty Fellow at Oregon State University, Corvallis, Director of the Northwest Alliance for Computational Science and Engineering, and Chair of the Parallel Tools Consortium.

Christian Lengauer (lengauer@fmi.uni-passau.de) is Chair for programming in the Department of Mathematics and Informatics of the University of Passau, Germany.

Thanks are due the International Conference and Research Center for Computer Science at SchlossDagstuhl (www.dagstuhl.de) funded by the German states of Saarland and Rheinland-Pfalz. The IBM T.J. Watson Research Center provided additional financial support for the seminar.