Guest Editors' Introduction

Effect of Hardware-Software Interaction on System Performance

In the constant quest for the right architecture or algorithm to improve computer applications' performance, interaction between hardware, system software, and user applications—each of which contributes greatly to total system performance—is of paramount importance. Technological advances in hardware make possible the development of machines with multiple-function units for pipeline, vector, and parallel processing. Synergistic developments in numerical analysis and software engineering provide new methods for implementing evolving architectures like vector and parallel processors, high-level language machines, and data flow computers. However, the promise of increased performance in future computing systems will remain only a promise unless hardware and software systems are properly fitted.

With respect to supercomputer architectures, system performance is measured in terms of the speed at which a processor can calculate floating point operations. Typically, applications are computationally intensive problems that require large amounts of processing time on even the fastest processors available. Making these applications suitable for different architectures requires careful consideration of algorithm and program structure to ensure optimal performance on individual machines, and this consideration must include the dependencies between the architectures and the applications programs.

The articles in this issue of Computer examine the importance of fitting computer programs to the architecture on which they will run. Further, the authors of the eight articles describe techniques for accomplishing this matching.

The first article, by M. H. MacDougall of Amdahl Corporation, describes an approach to modeling an architecture by analyzing, on an instruction level, the workload...
for which the architecture is to be developed. In particular, MacDougall advocates the development of parallel models for hardware and software to achieve optimal performance of the final system.

Lawrence Snyder's article describes the development of a programming environment designed to enable programmers to move from familiar sequential programming to parallel programming. The intent of Snyder's environment is to minimize the sources of dissimilarity between abstract algorithms and implementations of the algorithms on particular machines.

A model for predicting the execution time of numerical algorithms written for an actual processor array is provided by Loyc M. Adams and Thomas W. Crockett. Here again, the authors discuss their approach to analyzing and measuring software functions in order to optimally configure suitable hardware.

The fourth article, by Joseph A. Fisher, describes very long instruction word, or VLIW, machines as highly parallel architectures that present an alternative to multiprocessors and vector machines. Fisher details the development of a compiler that finds high levels of parallelism in ordinary sequential programs. The compiler is assigned the responsibility for matching the parallelism in a program to the parallel structure of the hardware.

Next, Robert G. Babb presents a model of computing that represents a compromise between data flow and traditional approaches to developing large-scale applications systems for multiprocessor supercomputers. Babb addresses the issue of fitting hardware and software by presenting programmers with a single model of computing that can be mapped onto a variety of high-speed architectures.

John L. Larson's article is an overview of the hardware and software available in the Cray X-MP-2 multiprocessor. He describes programming considerations that must be taken into account to achieve optimal benefit from the hardware and provides examples of processing speed gained by multitasking actual application codes for this architecture.

In the next article, the Rediflow multiprocessor is described by Robert M. Keller and Frank C. H. Lin. The authors discuss the architectural implications of software analysis and advocate the design of a multiprocessor based on the concepts of reduction evaluation and task flow, of which data flow is a special case.

Finally, in the last article, J. C. Browne emphasizes that, by understanding the execution behavior of software systems, the cost-effectiveness of computer architectures can be improved. Browne proposes research directions and techniques aimed at the coalescence of hardware and software projects to improve overall performances.

These eight articles, all of which describe efforts to produce cost-effective hardware-software systems, show the importance of hardware-software interactions on system performance. To this end, the projects discussed in these articles have created or are developing/implementing programming environments, compilers that map programs to architectures, or techniques to aid in the modeling of program behavior. Most important, however, these articles present a comprehensive discussion of (1) the reasons for recognizing and modeling this hardware-software inter-

\* action—starting at the system design phase—and (2) the benefits of properly fitting applications to architectures.

Acknowledgments

In August 1983, the Los Alamos National Laboratory hosted a workshop on the measurement of computer software performance. This issue of Computer is an outgrowth of that workshop; six of the eight following studies were presented there. We thank the Applied Mathematics Program of Basic Energy Science, Office of Energy Research, DOE, for its support of the workshop. Also, we thank the referees, whose comments and suggestions were extremely helpful.