ARCHITECTURAL ALTERNATIVES FOR EXPLOITING PARALLELISM

David J. Lilja

1951 - 1991 IEEE Computer Society Press

The Institute of Electrical and Electronics Engineers, Inc.

Architectural Alternatives for Exploiting Parallelism

David J. Lilja

1951-1991

IEEE Computer Society Press
Los Alamitos, California

Washington ● Brussels ● Tokyo

IEEE COMPUTER SOCIETY PRESS TUTORIAL

Library of Congress Cataloging-In-Publication Data

Lilja, David J.
Architectural alternatives for exploiting parallelism
p. cm.
Includes bibliographical references.
ISBN 0-8186-2640-2 (p). -- ISBN 0-8186-2641-0 (m/f). -- ISBN 0-8186-2642-9 (case).
1. Parallel computers. 2. Computer architecture I. Title QA76.58.L55 1992
004'.35--dc20 CIP 91-33298

1951-1991

Published by the
IEEE Computer Society Press
10662 Los Vaqueros Circle
PO Box 3014
Los Alamitos, CA 90720-1264

IEEE Computer Society Press Order Number 2642
Library of Congress Number 91-33298
IEEE Catalog Number 91EH0348-3
ISBN 0-8186-2641-0 (microfiche)
ISBN 0-8186-2642-9 (case)

Additional copies can be ordered from

IEEE Computer Society Press
Customer Service Center
10662 Los Vaqueros Circle
PO Box 3014
Los Alamitos, CA 90720-1264

IEEE Service Center
445 Hoes Lane
PO Box 1331
Piscataway, NJ 08855-1331
Technical Editor: Joydeep Ghosh

IEEE Computer Society
13, avenue de l'Aquilon
B-1200 Brussels
BELGIUM

IEEE Computer Society
Ooshima Building
2-19-1 Minami-Aoyama
Minato-ku, Tokyo 107
JAPAN

Editorial production: Robert Werner
Technical editor: Joydeep Ghosh
Copy editor: Phyllis Walker
Cover designed by Joseph Daigle/Schenk-Daigle Studios
Printed in the United States of America by Braun-Brumfield, Inc.

The Institute of Electrical and Electronics Engineers, Inc.

Architectural Alternatives for Exploiting Parallelism

ARCHITECTURAL ALTERNATIVES FOR EXPLOITING PARALLELISM

David J. Lilja

1951 · 1991

IEEE Computer Society Press

The Institute of Electrical and Electronics Engineers, Inc.

Preface

In Euclidean geometry, lines that never intersect are said to be "parallel." In computer architecture, computational tasks that are independent are said to be "executed in parallel" when they are run concurrently on different functional units or processors. The instantaneous number of these independent tasks from a single program that can be executed simultaneously is the parallelism available in that program at that instant in time. Since the number of independent tasks available to be executed varies over the course of a program's execution, a program's average parallelism may be much lower than its maximum instantaneous parallelism. Several studies have shown that there can be a significant amount of parallelism in many scientific and engineering application programs, but it remains an open question as to what type of processor architecture can best exploit this parallelism. As a result, an incredible variety of parallel computer architectures have been proposed and implemented. This tutorial surveys

- The fine-grained parallel architectures that attempt to exploit the parallelism available at the instruction-set level;
- The coarse-grained parallel architectures that exploit the parallelism available at the loop and subroutine levels; and
- The single-instruction stream, multiple-data stream (SIMD), massively parallel architectures that exploit parallelism across large data structures.

After studying the papers reprinted in this tutorial, the reader should have a clear understanding of the variety of architectures that are available for exploiting parallelism, as well as some idea of the trade-offs involved in using each of the architectures.

This tutorial is divided into four chapters, each with an introduction followed immediately by a list of relevant references. (In the reference lists, references that appear as reprints in this tutorial are marked with an asterisk.) These lists of references are necessarily incomplete, but they should provide a good starting point for anyone wishing to study a particular topic in greater depth. The introduction to Chapter 1 discusses the potential of parallel processing for reducing the execution time of a single program, beginning with Amdahl's conjecture that parallel speedup is limited ultimately by the inherently sequential component of the program. Reprinted in Chapter 1 are several papers that examine the controversial topic of how much parallelism actually is available in application programs.

The four sections in Chapter 2 present processor architectures that attempt to exploit parallelism at the instruction level. The first section provides an introduction to pipelined processors. The second section describes multiple instruction-issue architectures. The decoupled access/execute architectures discussed in the third section combine some aspects of both pipelining and multiple-instruction issuing. The last section of Chapter 2 introduces the dataflow concept, which potentially allows for a high degree of fine-grained parallelism.

Architectures that try to extract parallelism at higher levels, such as at the loop level and at the subroutine level, are presented in Chapter 3, along with an introduction to the massively parallel SIMD architectures. The first section in Chapter 3 discusses shared-memory multiprocessors, in which many identical processors are connected to a common memory. While communication among processors in this type of system is limited to the sharing of variables through the common memory structure, the distributed-memory multicomputer systems presented in the second section of Chapter 3 communicate using explicit messages. Also, a reprinted paper in this section presents a survey of systems that hide the message passing from the programmer by implementing a virtual shared memory on top of a distributed system. The last section of Chapter 3 presents several SIMD, massively parallel architectures in which a single sequence of instructions performs the same operation simultaneously on several thousand different data elements.

The parallel architectures presented in this tutorial have significant differences in synchronization overhead, instruction-scheduling constraints, memory latencies, and implementation details, making it difficult to determine which architecture is best able to exploit the parallelism available in a given application. Chapter 4 includes several studies that compare the performance of some of the different architectures.

Intended Audience

This tutorial is aimed at computer architects, system designers, researchers, and students who are interested in a guide for surveying and comparing the broad field of general-purpose parallel computer architectures. It also should serve as a valuable reference source for all computer professionals. Some basic knowledge of computer architecture and design will be helpful when reading this tutorial. The level of this tutorial is appropriate for graduate students and advanced undergraduate students, as well as practicing engineers.

Acknowledgments

The result of an effort such as preparing this tutorial text is as much a function of the environment in which the author finds himself as it is of the author's individual effort. Recognizing this synergism, I would like to thank all of my colleagues at the Center for Supercomputing Research and Development at the University of Illinois at Urbana-Champaign for helping to create an intellectually stimulating environment in which to research some of the problems in parallel processing. I would particularly like to thank Pen-Chung Yew for providing me with the freedom to pursue this project. I also would like to acknowledge the support of Mos Kaveh, my department head at the University of Minnesota, Minneapolis, in providing me with the time to complete this text.

Thanks to Rao Vemuri, editor-in-chief of the IEEE Computer Society Press, who helped track down this manuscript when it went astray during a change of personnel. As the coordinator of the referee process for this manuscript, Joydeep Ghosh provided some excellent reviews that significantly improved the quality of the presentation, particularly in the sections on dataflow processors, distributed-memory multicomputers, and reconfigurable and massively parallel architectures. I would also like to thank Bob Werner, Phyllis Walker, and Henry Ayling of the IEEE Computer Society Press for their invaluable assistance in editing and producing this tutorial. I hope the final result proves useful to the computer community.

David J. Lilja
Minneapolis, Minnesota
December 27, 1991

Table of Contents

Preface v
Intended Audience vi
Acknowledgments vi

Chapter 1: Introduction 1

Available parallelism

Modeling Speedup (*n*) Greater Than *n* 8
D.P. Helmbold and C.E. McDowell (*IEEE Transactions on Parallel and Distributed Systems*, Vol. 1, No. 2, April 1990, pp. 250-256)

Reevaluating Amdahl's Law 15
J.L. Gustafson (*Communications of the ACM*, Vol. 31, No. 5, May 1988, pp. 532-533)

Single Instruction Stream Parallelism is Greater Than Two 17
M. Butler et al. (*International Symposium on Computer Architecture*, 1991, pp. 276-286)

Measuring Parallelism in Computation-Intensive Scientific/Engineering Applications 28
M. Kumar (*IEEE Transactions on Computers*, Vol. 37, No. 9, September 1988, pp. 1088-1098)

Measuring the Parallelism Available for Very Long Instruction Word Architectures 39
A. Nicolau and J.A. Fisher (*IEEE Transactions on Computers*, Vol. C-33, No. 11, November 1984, pp. 968-976)

Limits of Instruction-Level Parallelism 48
D.W. Wall (*Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, April 1991, pp. 176-188)

Chapter 2: Fine-Grained Parallel Architectures 61

Pipelining

Reducing the Branch Penalty in Pipelined Processors 68
D.J. Lilja (*Computer*, Vol. 21, No. 7, July 1988, pp. 47-55)

Instruction Issue Logic in Pipelined Supercomputers 77
S. Weiss and J.E. Smith (*IEEE Transactions on Computers*, Vol. C-33, No. 11, November 1984, pp. 1013-1022)

Optimal Pipelining in Supercomputers 87
S.R. Kunkel and J.E. Smith (*IEEE International Symposium on Computer Architecture*, 1986, pp. 404-411)

Multiple-instruction issuing

An Efficient Algorithm for Exploiting Multiple Arithmetic Units 96
R.M. Tomasulo (*IBM Journal of Research and Development*, Vol. 11, No. 1, January 1967, pp. 25-33)

Software Pipelining: An Effective Scheduling Technique for VLIW Machines 105
M. Lam (*Proceedings of the SIGPLAN '88 Conference on Programming Language Design and Implementation*, June 1988, pp. 318-328)

The Cydra 5 Departmental Supercomputer 116
B.R. Rau et al. (*Computer*, Vol. 22, No. 1, January 1989, pp. 12-35)

Instruction Level Profiling and Evaluation of the IBM RS/6000 137
C. Stephens et al. (*International Symposium on Computer Architecture*, 1991, pp. 180-189)

Decoupled access/execute architectures

Structured Memory Access Architecture 148
A.R. Pleszkun and E.S. Davidson (*IEEE International Conference on Parallel Processing*, 1983, pp. 461-471)

Dynamic Instruction Scheduling and the Astronautics ZS-1 159
J.E. Smith (*Computer*, Vol. 22, No. 7, July 1989, pp. 21-35)

Dataflow processors

The U-Interpreter 176
Arvind and K.P. Gostelow (*Computer*, Vol. 15, No. 2, February 1982, pp. 42-49)

Two Fundamental Issues in Multiprocessing 184
Arvind and R.A. Iannucci (*Proceedings of DFVLR Conference on Parallel Processing in Science and Engineering*, 1987)

Toward a Dataflow/von Neumann Hybrid Architecture 208
R.A. Iannucci (*IEEE International Symposium on Computer Architecture*, 1988, pp. 131-140)

An Architecture of a Dataflow Single Chip Processor 218
S. Sakai et al. (*International Symposium on Computer Architecture*, 1989, pp. 46-53)

Chapter 3: Coarse-Grained and Massively Parallel Architectures 227

Shared-memory multiprocessors

Guided Self-Scheduling: A Practical Scheduling Scheme for Parallel Supercomputers 236
C.D. Polychronopoulos and D.J. Kuck (*IEEE Transactions on Computers*, Vol. C-36, No. 12, December 1987, pp. 1425-1439)

Synchronization, Coherence, and Event Ordering in Multiprocessors 251
M. Dubois, C. Scheurich, and F.A. Briggs (*Computer*, Vol. 21, No. 2, February 1988, pp. 9-21)

The Organization of the Cedar System 264
J. Konicek et al. (*International Conference on Parallel Processing, Vol. 1: Architecture*, 1991, pp. 49-56)

The Tera Computer System 272
R. Alverson et al. (*International Conference on Supercomputing*, 1990, pp. 1-6)

An Orthogonal Multiprocessor for Parallel Scientific Computations 278
K. Hwang, P.-S. Tseng, and D. Kim (*IEEE Transactions on Computers*, Vol. 38, No. 1, January 1989, pp. 47-61)

Distributed-memory multicomputers

Multicomputers: Message-Passing Concurrent Computers 294
W.C. Athas and C.L. Seitz (*Computer*, Vol. 21, No. 8, August 1988, pp. 9-24)

Hypercube Supercomputers 310
J.P. Hayes and T. Mudge (*Proceedings of the IEEE*, Vol. 77, No. 12, December 1989, pp. 1829-1841)

Matrix Computation on Distributed Memory Multiprocessors 323
C. Moler (*First SIAM Conference on Hypercube Multiprocessors*, 1986, pp. 181-195)

The Warp Computer: Architecture, Implementation, and Performance 338
M. Annaratone et al. (*IEEE Transactions on Computers*, Vol. C-36, No. 12, December 1987, pp. 1523-1538)

Distributed Shared Memory: A Survey of Issues and Algorithms 354
B. Nitzberg and V. Lo (*Computer*, Vol. 24, No. 8, August 1991, pp. 52-60)

Reconfigurable and massively parallel architectures

An Overview of the Texas Reconfigurable Array Computer 364
M.C. Sejnowski et al. (*AFIPS National Computer Conference*, 1980, pp. 631-641)

Architecture and Applications of the Connection Machine 375
L.W. Tucker and G.G. Robertson (*Computer*, Vol. 21, No. 8, August 1988, pp. 26-38)

BLITZEN: A Highly Integrated Massively Parallel Machine 388
D.W. Blevins et al. (*Journal of Parallel and Distributed Computing*, Vol. 8, No. 2, February 1990, pp. 150-160)

The Design of the MasPar MP-1: A Cost-Effective Massively Parallel Computer 399
J.R. Nickolls (*IEEE Digest of Papers-CompCon*, February 1990, pp. 25-28)

Task Flow Computer Architecture 403
R.W. Horst (*International Conference on Parallel Processing, Vol. I: Architecture*, 1990, pp. 533-540)

Chapter 4: Architectural Comparisons 411

Comparing parallelism extraction techniques

Available Instruction-Level Parallelism for Superscalar and Superpipelined Machines 414
N.P. Jouppi and D.W. Wall (*Third International Conference on Architectural Support for Programming Languages and Operating Systems*, April 1989, pp. 272-282)

Tradeoffs in Instruction Format Design for Horizontal Architectures 425
G.S. Sohi and S. Vajapeyam (*Third International Conference on Architectural Support for Programming Languages and Operating Systems*, April 1989, pp. 15-25)

Experimental Application-Driven Architecture Analysis of an SIMD/MIMD Parallel Processing System 436
E.C. Bronson, T.L. Casavant, and L.H. Jamieson (*IEEE Transactions on Parallel and Distributed Systems*, Vol. 1, No. 2, April 1990, pp. 195-205)

About the Author 447

Chapter 1: Introduction

The past several decades have seen exponential improvements in the performance of computer systems, mostly due to increases in single-processor performance. These increases have been possible because, as a fundamental limit has been approached in one technology, a new technology has been introduced to supplement or replace the old. For example, the frequent introduction of new logic families has helped reduce supercomputer cycle times by a factor of two every four or five years over the last 30 years.[1] However, as semiconductor technologies have matured, growth in single-processor performance has shown signs of slowing. To continue to obtain the expected improvements in system performance, computer architects are incorporating more parallel processing technology into new computer designs.

In contrast with the technique of speeding up a single processor by using new semiconductor technology to produce faster logic, parallel processing attempts to increase performance by dividing a program into independent tasks that can be executed concurrently on several functional units or processors. The size of the computational tasks, called the "granularity" of the parallelism, can have a significant impact on the performance of a parallel system, since there may be some parallelism accessible at one task size, but not at another. The papers selected for this tutorial survey several classes of parallel computer architectures:

- Those that attempt to exploit the fine-grained parallelism available at the instruction-set level,
- Those that exploit the coarser-grained parallelism at the loop level and the subroutine level, and
- Those that exploit data parallelism by performing the same operation simultaneously on several thousand different data elements.

To obtain an idea of the potential performance gains of using a parallel architecture, consider a program that executes in time T_1 on a single processor. If α is the dynamic fraction of all operations in the program that must be executed sequentially, the time required to execute this part of the program on a single processor (when the single processor is part of a multiprocessor) is αT_1. The remaining fraction of the operations in the program, $(1 - \alpha)$, are said to be "perfectly parallel" and so can be executed p times faster on a multiprocessor with p processors, giving a parallel execution time of $(1 - \alpha)(T_1/p)$. The total execution time T_p is the sum of these serial and parallel times, $T_p = \alpha T_1 + (1 - \alpha)T_1/p$, and the speedup S_p for the multiprocessor is the ratio of the single-processor execution time T_1 to the multiprocessor execution time T_p, as shown in the following equation:

$$S_p = \frac{T_1}{T_p} = \frac{T_1}{\alpha T_1 + \frac{(1-\alpha)T_1}{p}} = \frac{1}{\frac{1}{p} + \alpha\left(1 - \frac{1}{p}\right)} \qquad 1$$

Ideally, the sequential component is negligible (that is, $\alpha \approx 0$), making the speedup the same as the number of processors: $S_p \approx p$. However, for $\alpha > 0$, the limit as $p \to \infty$ in Equation 1 is $S_p \to 1/\alpha$. This result, known as Amdahl's Law,[2] says that no matter how many processors are used to execute a program, the maximum speedup is limited by the program's inherently sequential component, α. For example, if 10 percent of the total operations executed in a program must be executed sequentially (that is, $\alpha = 0.1$), the maximum speedup S_p for the program is $1/0.1 = 10$. There have been some reports of obtaining speedups greater than p when using p processors,[3-5] but it appears[6*] that the causes of these reports are attributable to such factors as overhead reductions, cache size effects, memory latency hiding, and the use of randomized algorithms.

There are arguments that, as more processors are added to a system, the system is more effectively used by solving a larger version of a problem in the same amount of time than by trying to reduce the execution time of a fixed-size version of the problem. This approach defines a new concept, called "scaled speedup,"[7,8*] in which the value of α is reduced by increasing the number of operations in the parallel part of the program. In addition, there are many enhancements to and variations of Amdahl's Law that incorporate more details into the system models than the simple one used here[9,10] and that relate efficiency and speedup.[11-14]

The actual parallelism available in an application program is limited by its dependences.[15] A dependence between two computational tasks is a conflict that prevents the tasks from executing simultaneously. That is, a dependence from one task to another implies that the first task must complete its execution before the dependent task can begin executing. Dependences can be categorized into the following three types:

(1) Resource dependences are a physical limitation imposed by the architecture and the hardware of the particular machine on which the program is to be executed. This type of dependence occurs when two tasks need to use the same resource at the same time, forcing one to wait for the other to complete its execution. Any real machine has a limited number of functional units and limited memory bandwidth, for example, which may prevent exploiting all of the parallelism available in an application.

(2) Control dependences, in which a statement cannot execute until the result of an earlier conditional statement has been resolved, are a function of the algorithm and of the programming language used to implement the algorithm.

(3) Data dependences, also known as "hazards,"[16] are read-write conflicts between two operations that prevent the operations from executing concurrently. In a flow dependence (read-after-write hazard), the result of one operation is read by a subsequent operation. An output dependence (write-after-write hazard) exists between two statements when they both write to the same variable. Two statements are said to have an "antidependence" (write-after-read hazard) when a later statement writes to a variable that is read by an earlier statement. Output dependences and antidependences occur when variables and registers are reused by the programmer or the compiler; given a sufficient number of temporary storage locations, many of these two kinds of dependences can be eliminated by renaming variables.

Available parallelism

The question of how much parallelism is available in application programs is somewhat controversial. Table 1 summarizes the results of several studies that show that there is a wide range of potential parallelism in actual application programs. This wide range is due partly to the significantly different architectural assumptions made in each of the simulation studies and partly to actual differences in inherent parallelism. These assumptions include differences in memory delays, functional unit latencies, register and memory conflicts, synchronization costs, specific application programs tested, compiler quality, and other such implementation details. However, it appears that when basic block boundaries are ignored — so that the entire program is available for exploiting parallelism — engineering and scientific programs can exhibit a high level of inherent parallelism. (A "basic block" is a sequence [or block] of instructions in a program that has no branches into or out of the block.) Computation that is less numeric than that in "typical" scientific and engineering application programs has relatively little parallelism, even when basic block boundaries are ignored. While compiler transformations and algorithm changes may significantly increase the available parallelism in an application,[29] limiting parallelism extraction to a basic block limits speedups to a maximum of about two to four.

Table 1. Reported speedup values.

Researchers	Speedup Mean[a]	Speedup Range	Programs tested	Functional units	Basic blocks?
Kumar (1988)[21*]	1400	475-3500	4 scientific	∞	No
Arvind et al. (1988)[18]	590	463-745	2 scientific	∞	No
Butler et al. (1991)[19*]	171	17-1165	9 mixture	∞	No
Riseman and Foster (1972)[24]	37	7.8-120.5	7 mixture	∞	No
	1.66	1.22-2.98	7 mixture	∞	Yes
Nicolau and Fisher (1984)[23*]	28	3-988.5	22 scientific	∞	No
	2.2	1.37-5.58	22 scientific	∞	Yes
Wall (1991)[27*]	24	6.5-60.4	19 mixture	∞	No
	2.7	1.3-17.5	19 mixture	∞	Yes
Lilja and Yew (1991)[22]	8.7	1.5-646	20 scientific	∞	No
Acosta et al. (1986)[17]	2.79	NA[b]	Livermore loops	∞	Yes
Smith et al. (1989)[25]	2.5	NA	13 nonscientific	—[c]	No
	4.1	NA	13 nonscientific	—[d]	No
Jouppi and Wall (1989)[20*]	2.2	1.6-3.2	8 mixture	8	Yes
Tjaden and Flynn (1970)[26]	1.86	1.2-3.2	31 mixture	∞	Yes
Weiss and Smith (1984)[28*]	1.6	1.2-2.0	Livermore loops	—[e]	Yes

a. This value is either the mean reported in the study, or the geometric mean calculated from the reported values.
b. NA stands for "not available."
c. One of each type of functional unit; limited instruction look-ahead.
d. One of each type of functional unit; limited instruction look-ahead; flow dependences only.
e. Cray-1 functional units.

Based on the studies summarized in Table 1, it is not apparent what type of architecture can best exploit the available parallelism nor even how much parallelism exists. As a result, an incredible variety of architectures has been proposed to try to exploit whatever parallelism is available. Pipelined processors and multiple instruction-issue processors, discussed in Chapter 2, exploit the fine-grained parallelism available at the instruction-set level. The multiprocessors presented in the first two sections of Chapter 3 exploit coarse-grained parallelism by distributing independent loop iterations and subroutines to different processors. In addition, massively parallel single-instruction stream, multiple-data stream (SIMD) architectures discussed in the last section of Chapter 3 exploit the fact that — in some problems — the same operation is repeated many times over a large array of data. In these SIMD machines, a single instruction-sequencer controls the simultaneous operation of thousands of identical processors, with each processor acting on a different data element.

The parallel architectures presented in this tutorial use significantly different techniques for synchronizing, scheduling instructions, reducing memory delay, and implementing the hardware, making it difficult to determine which architecture is best able to exploit the parallelism available in a given application. By presenting a few studies that compare the performance potential of several of the different parallel architectures, Chapter 4 provides some insight into making this determination.

References (Chapter 1)

An asterisk following a reference citation below indicates the inclusion of that paper in this tutorial.

Available parallelism

1. J. Worlton, "Some Patterns of Technological Change in High-Performance Computers," *Supercomputing '88*, 1988, pp. 312-320.

2. G.M. Amdahl, "Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities," *AFIPS Conf. Proc., Spring Joint Computer Conf.*, Apr. 1967, pp. 483-485.

3. K. Li, "IVY: Shared Virtual Memory System for Parallel Computing," *Int'l Conf. Parallel Processing, Vol. II: Software*, 1988, pp. 94-101.

4. B.R. Preiss and V. C. Hamacher, "Semi-Static Dataflow," *Int'l Conf. Parallel Processing, Vol II: Software*, 1988, pp. 127-134.

5. J. Sanguinetti, "Performance of Message-Based Multiprocessor," *Computer*, Vol. 19, No. 9, Sept. 1986, pp. 47-55.

6.* D.P. Helmbold and C.E. McDowell, "Modeling Speedup (*n*) Greater Than *n*," *IEEE Trans. Parallel and Distributed Systems*, Vol. 1, No. 2, Apr. 1990, pp. 250-256.

7. J.L. Gustafson, G.R. Montry, and R.E. Benner, "Development of Parallel Methods for a 1024-Processor Hypercube," *SIAM J. Scientific and Statistical Computing*, Vol. 9, No. 4, July 1988, pp. 609-638.

8.* J.L. Gustafson, "Reevaluating Amdahl's Law," *Comm. ACM*, Vol. 31, No. 5, May 1988, pp. 532-533.

9. X. Zhou and J. Staudhammer, "New Speedup Function for Supercomputing," *Proc. Int'l Symp. Mini and Microcomputers*, Dec. 1988.

10. X. Zhou, "Bridging the Gap Between Amdahl's Law and Sandia Laboratory's Results," *Comm. ACM*, Vol. 32, No. 8, Aug. 1989, pp. 1014-1015.

11. M.L. Barton and G.R. Withers, "Computing Performance as a Function of the Speed, Quantity, and Cost of the Processors," *Proc. Supercomputing '89*, 1989, pp. 759-764.

12. D.L. Eager, J. Zahorjan, and E.D. Lazowska, "Speedup Versus Efficiency in Parallel Systems," *IEEE Trans. Computers*, Vol. 38, No. 3, Mar. 1989, pp. 408-423.

13. A.H. Karp and H.P. Flatt, "Measuring Parallel Processor Performance," *Comm. ACM*, Vol. 33, No. 5, May 1990, pp. 539-543.

14. J.R. Zorbas, D.J. Reble, and R.E. VanKooten, "Measuring the Scalability of Parallel Computer Systems," *Proc. Supercomputing '89*, 1989, pp. 832-841.

15. D.J. Kuck, *The Structure of Computers and Computations*, John Wiley and Sons, New York, N.Y., 1978, pp. 135-141.

16. P.M. Kogge, *The Architecture of Pipelined Computers*, Hemisphere, New York, N.Y., 1981, p. 220.

17. R.D. Acosta, J. Kjelstrup, and H.C. Torng, "An Instruction Issuing Approach to Enhancing Performance in Multiple Functional Unit Processors," *IEEE Trans. Computers*, Vol. C-35, No. 9, Sept. 1986, pp. 815-828.

18. Arvind, D.E. Culler, and G.K. Maa, "Assessing the Benefits of Fine-Grain Parallelism in Dataflow Programs," *Proc. Supercomputing '88*, Nov. 1988, pp. 60-69.

19.* M. Butler et al., "Single Instruction Stream Parallelism is Greater Than Two," *Int'l Symp. Computer Architecture*, 1991, pp. 276-286.

20.* N.P. Jouppi and D.W. Wall, "Available Instruction-Level Parallelism for Superscalar and Superpipelined Machines," *Third Int'l Conf. Architectural Support Programming Languages and Operating Systems*, Apr. 1989, pp. 272-282.

21.* M. Kumar, "Measuring Parallelism in Computation-Intensive Scientific/Engineering Applications," *IEEE Trans. Computers*, Vol. 37, No. 9, Sept. 1988, pp. 1088-1098.

22. D.J. Lilja and P.-C. Yew, "The Performance Potential of Fine-Grain and Coarse-Grain Parallel Architectures," *Hawaii Int'l Conf. System Sciences, Vol. I: Architecture*, 1991, pp. 324-333.

23.* A. Nicolau and J.A. Fisher, "Measuring the Parallelism Available for Very Long Instruction Word Architectures," *IEEE Trans. Computers*, Vol. C-33, No. 11, Nov. 1984, pp. 968-976.

24. E.M. Riseman and C.C. Foster, "The Inhibition of Potential Parallelism by Conditional Jumps," *IEEE Trans. Computers*, Vol. C-21, No. 12, Dec. 1972, pp. 1405-1411.

25. M.D. Smith, M. Johnson, and M.A. Horowitz, "Limits on Multiple Instruction Issue," *Third Int'l Conf. Architectural Support Programming Languages and Operating Systems*, Apr. 1989, pp. 290-302.

26. G.S. Tjaden and M.J. Flynn, "Detection and Parallel Execution of Independent Instructions," *IEEE Trans. Computers*, Vol. C-19, No. 10, Oct. 1970, pp. 889-895.

27.* D.W. Wall, "Limits of Instruction-Level Parallelism," *Fourth Int'l Conf. Architectural Support Programming Languages and Operating Systems*, Apr. 1991, pp. 176-188.

28.* S. Weiss and J.E. Smith, "Instruction Issue Logic in Pipelined Supercomputers," *IEEE Trans. Computers*, Vol. C-33, No. 11, Nov. 1984, pp. 1013-1022.

29. D.J. Kuck et al., "The Effects of Program Restructuring, Algorithm Change, and Architecture Choice on Program Performance," *Int'l Conf. Parallel Processing*, 1984, pp. 129-138.