# Lecture Notes in Computer Science

Edited by G. Goos and J. Hartmanis

127

Y. Wallach

# Alternating Sequential/Parallel Processing

Springer-Verlag
Berlin Heidelberg New York

# Lecture Notes in Computer Science

Edited by G. Goos and J. Hartmanis

127

Y. Wallach

# Alternating Sequential/Parallel Processing

Springer-Verlag
Berlin Heidelberg New York 1982

**Editorial Board**

W. Brauer P. Brinch Hansen D. Gries C. Moler G. Seegmüller
J. Stoer N. Wirth

**Autor**

Y. Wallach
Wayne State University, College of Engineering
Department of Electrical and Computer Engineering
Detroit, Michigan 48202, USA

CR Subject Classifications (1981): 6.2, 5.25, 5.29

ISBN 3-540-11194-8 Springer-Verlag Berlin Heidelberg New York
ISBN 0-387-11194-8 Springer-Verlag New York Heidelberg Berlin

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically those of translation, reprinting, re-use of illustrations, broadcasting, reproduction by photocopying machine or similar means, and storage in data banks. Under § 54 of the German Copyright Law where copies are made for other than private use, a fee is payable to "Verwertungsgesellschaft Wort", Munich.

© by Springer-Verlag Berlin Heidelberg 1982
Printed in Germany

Printing and binding: Beltz Offsetdruck, Hemsbach/Bergstr.
2145/3140-543210

# Lecture Notes in Computer Science

Vol. 23: Programming Methodology. 4th Informatik Symposium, IBM Germany Wildbad, September 25–27, 1974. Edited by C. E. Hackl. VI, 501 pages. 1975.

Vol. 24: Parallel Processing. Proceedings 1974. Edited by T. Feng. VI, 433 pages. 1975.

Vol. 25: Category Theory Applied to Computation and Control. Proceedings 1974. Edited by E. G. Manes. X, 245 pages. 1975.

Vol. 26: GI-4. Jahrestagung, Berlin, 9.–12. Oktober 1974. Herausgegeben im Auftrag der GI von D. Siefkes. IX, 748 Seiten. 1975.

Vol. 27: Optimization Techniques. IFIP Technical Conference. Novosibirsk, July 1–7, 1974. (Series: I.F.I.P. TC7 Optimization Conferences.) Edited by G. I. Marchuk. VIII, 507 pages. 1975.

Vol. 28: Mathematical Foundations of Computer Science. 3rd Symposium at Jadwisin near Warsaw, June 17–22, 1974. Edited by A. Blikle. VII, 484 pages. 1975.

Vol. 29: Interval Mathematics. Procedings 1975. Edited by K. Nickel. VI, 331 pages. 1975.

Vol. 30: Software Engineering. An Advanced Course. Edited by F. L. Bauer. (Formerly published 1973 as Lecture Notes in Economics and Mathematical Systems, Vol. 81) XII, 545 pages. 1975.

Vol. 31: S. H. Fuller, Analysis of Drum and Disk Storage Units. IX, 283 pages. 1975.

Vol. 32: Mathematical Foundations of Computer Science 1975. Proceedings 1975. Edited by J. Bečvář. X, 476 pages. 1975.

Vol. 33: Automata Theory and Formal Languages, Kaiserslautern, May 20–23, 1975. Edited by H. Brakhage on behalf of GI. VIII, 292 Seiten. 1975.

Vol. 34: GI – 5. Jahrestagung, Dortmund 8.–10. Oktober 1975. Herausgegeben im Auftrag der GI von J. Mühlbacher. X, 755 Seiten. 1975.

Vol. 35: W. Everling, Exercises in Computer Systems Analysis. (Formerly published 1972 as Lecture Notes in Economics and Mathematical Systems, Vol. 65) VIII, 184 pages. 1975.

Vol. 36: S. A. Greibach, Theory of Program Structures: Schemes, Semantics, Verification. XV, 364 pages. 1975.

Vol. 37: C. Böhm, $\lambda$-Calculus and Computer Science Theory. Proceedings 1975. XII, 370 pages. 1975.

Vol. 38: P. Branquart, J.-P. Cardinael, J. Lewi, J.-P. Delescaille, M. Vanbegin. An Optimized Translation Process and Its Application to ALGOL 68. IX, 334 pages. 1976.

Vol. 39: Data Base Systems. Proceedings 1975. Edited by H. Hasselmeier and W. Spruth. VI, 386 pages. 1976.

Vol. 40: Optimization Techniques. Modeling and Optimization in the Service of Man. Part 1. Proceedings 1975. Edited by J. Cea. XIV, 854 pages. 1976.

Vol. 41: Optimization Techniques. Modeling and Optimization in the Service of Man. Part 2. Proceedings 1975. Edited by J. Cea. XIII, 852 pages. 1976.

Vol. 42: James E. Donahue, Complementary Definitions of Programming Language Semantics. VII, 172 pages. 1976.

Vol. 43: E. Specker und V. Strassen, Komplexität von Entscheidungsproblemen. Ein Seminar. V, 217 Seiten. 1976.

Vol. 44: ECI Conference 1976. Proceedings 1976. Edited by K. Samelson. VIII, 322 pages. 1976.

Vol. 45: Mathematical Foundations of Computer Science 1976. Proceedings 1976. Edited by A. Mazurkiewicz. XI, 601 pages. 1976.

Vol. 46: Language Hierarchies and Interfaces. Edited by F. L. Bauer and K. Samelson. X, 428 pages. 1976.

Vol. 47: Methods of Algorithmic Language Implementation. Edited by A. Ershov and C. H. A. Koster. VIII, 351 pages. 1977.

Vol. 48: Theoretical Computer Science, Darmstadt, March 1977. Edited by H. Tzschach, H. Waldschmidt and H. K.-G. Walter on behalf of GI. VIII, 418 pages. 1977.

Vol. 49: Interactive Systems. Proceedings 1976. Edited by A. Blaser and C. Hackl. VI, 380 pages. 1976.

Vol. 50: A. C. Hartmann, A Concurrent Pascal Compiler for Minicomputers. VI, 119 pages. 1977.

Vol. 51: B. S. Garbow, Matrix Eigensystem Routines – Eispack Guide Extension. VIII, 343 pages. 1977.

Vol. 52: Automata, Languages and Programming. Fourth Colloquium, University of Turku, July 1977. Edited by A. Salomaa and M. Steinby. X, 569 pages. 1977.

Vol. 53: Mathematical Foundations of Computer Science. Proceedings 1977. Edited by J. Gruska. XII, 608 pages. 1977.

Vol. 54: Design and Implementation of Programming Languages. Proceedings 1976. Edited by J. H. Williams and D. A. Fisher. X, 496 pages. 1977.

Vol. 55: A. Gerbier, Mes premières constructions de programmes. XII, 256 pages. 1977.

Vol. 56: Fundamentals of Computation Theory. Proceedings 1977. Edited by M. Karpiński. XII, 542 pages. 1977.

Vol. 57: Portability of Numerical Software. Proceedings 1976. Edited by W. Cowell. VIII, 539 pages. 1977.

Vol. 58: M. J. O'Donnell, Computing in Systems Described by Equations. XIV, 111 pages. 1977.

Vol. 59: E. Hill, Jr., A Comparative Study of Very Large Data Bases. X, 140 pages. 1978.

Vol. 60: Operating Systems, An Advanced Course. Edited by R. Bayer, R. M. Graham, and G. Seegmüller. X, 593 pages. 1978.

Vol. 61: The Vienna Development Method: The Meta-Language. Edited by D. Bjørner and C. B. Jones. XVIII, 382 pages. 1978.

Vol. 62: Automata, Languages and Programming. Proceedings 1978. Edited by G. Ausiello and C. Böhm. VIII, 508 pages. 1978.

Vol. 63: Natural Language Communication with Computers. Edited by Leonard Bolc. VI, 292 pages. 1978.

Vol. 64: Mathematical Foundations of Computer Science. Proceedings 1978. Edited by J. Winkowski. X, 551 pages. 1978.

Vol. 65: Information Systems Methodology, Proceedings, 1978. Edited by G. Bracchi and P. C. Lockemann. XII, 696 pages. 1978.

Vol. 66: N. D. Jones and S. S. Muchnick, TEMPO: A Unified Treatment of Binding Time and Parameter Passing Concepts in Programming Languages. IX, 118 pages. 1978.

Vol. 67: Theoretical Computer Science, 4th GI Conference, Aachen, March 1979. Edited by K. Weihrauch. VII, 324 pages. 1979.

Vol. 68: D. Harel, First-Order Dynamic Logic. X, 133 pages. 1979.

Vol. 69: Program Construction. International Summer School. Edited by F. L. Bauer and M. Broy. VII, 651 pages. 1979.

Vol. 70: Semantics of Concurrent Computation. Proceedings 1979. Edited by G. Kahn. VI, 368 pages. 1979.

Vol. 71: Automata, Languages and Programming. Proceedings 1979. Edited by H. A. Maurer. IX, 684 pages. 1979.

Vol. 72: Symbolic and Algebraic Computation. Proceedings 1979. Edited by E. W. Ng. XV, 557 pages. 1979.

Vol. 73: Graph-Grammars and Their Application to Computer Science and Biology. Proceedings 1978. Edited by V. Claus, H. Ehrig and G. Rozenberg. VII, 477 pages. 1979.

Vol. 74: Mathematical Foundations of Computer Science. Proceedings 1979. Edited by J. Bečvář. IX, 580 pages. 1979.

Vol. 75: Mathematical Studies of Information Processing. Proceedings 1978. Edited by E. K. Blum, M. Paul and S. Takasu. VIII, 629 pages. 1979.

Vol. 76: Codes for Boundary-Value Problems in Ordinary Differential Equations. Proceedings 1978. Edited by B. Childs et al. VIII, 388 pages. 1979.

# FOREWORD

The fields of "parallel processing" and "complexity theory" have drawn the attention of researchers for the last 20-30 years. In essence, the idea is that speeding up computations by building ever faster computers will either stop or will not be very profitable. What can then be more natural than connecting computers in parallel? With p processors, we might get slightly less than a p-fold increase in speed, but an increase we will certainly get.

Still, after all these years and a large body of excellent results, parallel processing has not triumphed. A researcher working for a computer manufacturer even told me that "parallel processing will never work." What are the reasons for such pessimism?

(1) The first is that those systems which were built were applied to what I call "inherently parallel" problems. Thus, PEPE tracks a number of targets; each processor has a single target assigned to it on which it works independently of and concurrently with other processors. Payroll processing may be another example of an inherently parallel problem; there is no connection between my salary and that of my boss, so they can be processed independently.

Invariably, the question was asked how many such problems exist. A look at any book on numerical analysis was rather disappointing: very few algorithms were anything but purely sequential. Whether our minds work so that we do not do "B" until having finished "A", or we are taught to solve problems that way, is immaterial. The fact remains that few problems seem to be inherently parallel.

(2) Mathematicians have the habit that if they cannot have the computer they need, they define a theoretical model and leave the worry of building it to engineers. Unfortunately, the abstract system so defined suffered from not being practical. Let me give examples of what I mean.

- The model assumes that all p processors are connected to a large, common memory. Whenever two or more processors requested the same word from it, they would get if instantaneously. This is clearly not achievable and leads to strange behaviors: one system apparently slowed down with the addition of processors.

- The number of processors was to be unbounded. It had better be so, since, as shown in one example of this book, multiplying two (nxn) matrices, requires $n^3$ processors. A rather "lean" matrix of n=100 would thus require a million processors.

(3) One of the first parallel systems built, the Illiac, has "private memories" for the "slaves." Unfortunately, the connections are such that quite often only one of the 64 "slaves" or only the single master works - the rest are idle. This is the case of "sequentialization" and attempts to reduce it were not always crowned with success.

(4) A rather important objective of parallelization was that of reliability

此为试读,需要完整PDF请访问: www.ertongbook.com

and availability. The first is defined here as the ability to identify a faulty unit and amputate it, the second as the capability to proceed working despite this removal. Illiac and similar systems have neither: if a unit fails, it must be exchanged or repaired before the system can work again.

This book is going to suggest a solution to this dilemma based on a single observation:

Problems are neither inherently parallel nor completely sequential. Solutions should adapt themselves to this fact rather than fighting it.

Take it in another way. When you are walking, the mind directs the movement of your legs and hands changing from time to time the pace, direction, etc. The mind works sequentially, the legs and hands are mostly independent and you would like to minimize the movements or time. In computerese: Assemble a master/slaves systems such that the master directs sequentially the slaves which then work in parallel, rather independently on different data (stones or meadows).

The first part of the book (Chapters 1 to 4) will develop a hardware system called ASP for "Alternating Sequential/Parallel" which, as its name implies, adapts well to the central idea, is more than just a model (one system is commercially available, another is being assembled), includes "private memories" and should have high reliability/availability because of symmetry. The second part (Chapters 5 to 11) develops algorithms and programs for it. For each of these programs, the speed-up was calculated; they all approach more or less the optimum.

The development of these algorithms was so easy that the thought occurred to me that something must be wrong. On second thought, I concluded that this was only natural since the following comparison, with other approaches holds: instead of trying to eliminate completely the sequential parts of a solution we try only to minimize it. The first is not always possible, but in the second case the amount of minimization reflects only the author's ingenuity; I am sure readers will improve on some or all algorithms proposed by me. They might also add to the general approaches developed: "vertical programming" and "block schemes" as well as "tearing," "chasing" and the like.

The problems chosen are all in numerical, linear algebra because, as once observed, 75% of all scientific problems lead to linear equations. Since most work on other parallel systems also centered on linear algebra, a comparison of efficiency, speedup etc. between them and the ASP-methods is possible.

This is really a research report. Therefore the work of and on other systems is only briefly sketched - as much as needed for the comparison. Additionally, since the field of numerical solutions to linear algebra problems is very well covered in literature, it also is only sketched - wherever needed for developing ASP-algorithms. The reader is referred to the book by Young [Yo 71] for iterative methods and to the books by Young and Gregory [YG] and by Stoer and Bullirsch [SB] for all the rest. Especially the last is warmly recommended and I assume that the

reader knows the relevant material so that no repetition is needed.

Let me close with additional remarks:

(1) The numbering is according to sections. Thus, equation (15) in section 2 of chapter 5 is referred to as (15) throughout chapter 5, but as 5.2.15 in other chapters.

(2) The book can be used for two graduate courses. The first part on the model (architecture) of parallel systems would have to be expanded early - each section of chapters 2 and 4 could (and probably should) be expanded into a separate chapter. The second part (chapters 5 to 9) could also be expanded - especially on the use of other (non-ASP) languages and algorithms.

(3) Instead of having a motto for every chapter, I offer two here:

"Nothing will ever be attempted if all possible objections must be first overcome".

"Science seeks the truth, engineering - the compromise." The first, by Samuel Johnson explains why I have had the audacity to offer a new solution at all. The second, by me, explains why it had to be a compromise solution: I am an engineer, not a mathematician. I will also be only too happy if readers will point out to me mistakes. On the whole, I hope they find the approach to be new but correct and simple.

A large part of chapters 6, 7 and 8 is from the Ph.D. Thesis and the papers which Dr. Conrad published with me. I very much thank him for his cooperation.

Thanks go also to Dr. B.Z. Barta, Prof. K.S. Fu, Prof. W. Handler, Dr. O. Herzog, Prof. M. Schlesinger, Mr. A. Shimor, Mr. J. Tenenbaum and Mr. B. Waumans - they all read and made remarks to the manuscript. Finally, I would like to thank my students who had to endure constant changes of text throughout the last eight years. I am afraid that constant rewriting has introduced more errors than is usual - for which I ask the readers forgiveness. The main points he should gather from this book are:

- ASP is a new hardware unlike other models
- Because ASP drops the requirement of complete parallelization, it is easy to develop new and efficient algorithms. I would be glad to hear from readers who have done it in their field of work.

# NOTATION

Acronyms are used frequently in the text, but may be forgotten as soon as the particular Section is completed, except for the following:

Apps = Abstract pps
pps = parallel processing systems (nonsingular-pp's)
Kopps, Mopps and Topps = Korn's, Multibus and Tristate oriented pps resp.
ASP = Alternating Sequential/Parallel system
EPOC = Electric Power Control
Poof = Parallel, optimally ordered factorization
SIMD = Single-instruction, Multiple-data system
MIMD = Multiple-instruction, Multiple-data system
PEPE = Parallel Ensemble of Processing Elements
SMS = Siemens' Multiprocessing System
CU = Control unit
PE = Processing element
ME = Memory element

We will use capital underlined letters for matrices, lower-case underlined for vectors, lower case and Greek letters for scalars. In particular we use:

| Matrices | Vectors | Scalars |
|---|---|---|
| A - general | b - $A\underline{x} = \underline{b}$ | c = no. of nonzero terms |
| B - optimization | c - control | d = density (c/n) |
| C - general | f - function | e = error |
| D - diagonal | g - gradient | f = function |
| E - error | h - constraints | h = slice width |
| F - Frobenius | r - residual | m = number of memories |
| G - Givens' | s - state, speedup | n = dimension |
| H - Householder's | x,z - general vectors | p = number of slaves |
| I - Iteration | | q = index of slave |
| J - Jacobian | | r = residual |
| L - lower (left part of A) | | t = time |
| M - symmetric | | w = overrelaxation factor |
| O - zero | | $\alpha,\mu,\tau,\beta,\gamma$ are times of addition/subtraction, multiplication/division, transfer, synchronization and square root taking respectively. |
| P - permutation | | |
| Q - orthonormal | | |
| R - right (upper part of A) | | |
| S - similarity | | |
| T - tridiagonal | | $\lambda$ = eigenvalue |
| $\underline{a}_{.i}$ = i-th column of $\underline{A}$; $\underline{a}_{i.}$ = i-th row of $\underline{A}$ | | $\omega = \alpha + \mu$ |
| $a_{ij}$ = (i,j)-th element of $\underline{A}$ | | $\Omega$ = number of operations |
| | | $\rho$ = (time) ratio |
| $\underline{a}*\underline{b}$ = scalar product | | $\eta$ = efficiency |
| $\underline{a}^T$ = transpose of $\underline{A}$ | | $\nu$ = utilization |
| | | $\sigma$ = speedup |

Additional letters are used, but being local (to a few pages) may be forgotten as soon as you finish reading.

Springer-Verlag would like to thank the following publishers for granting permission to quote from the following original papers:

I.E.E.E.

Barnes et al:"The Illiac-IV Computer",Trans.IEEE,Vol.C-17,No8,746-760.

Davis:"The Illiac-IV Processing Element",Trans.IEEE,Vol.C-18,No9,800-816, 1969.

Gilmore:"Matrix Computations on an Associative Processor",Proc.Sagamore Conference, 272-290, 1974.

Wallach:"Parallel Processor Systems in Power-Dispatch",IEEE-Summer Power Meeting,July 74,Papers 74334-9 and C74355-6.

Batcher:"Staran/Radcap Hardware Architecture" and "The flip-network in Staran", Sagamore Conference, 147-152 and 65-71 respectively.

Conrad,Wallach:"Parallel,Optimally Ordered Factorization",PICA-Conf. (IEEE),May 1977, 302-306.

Kuck:"Illiac-IV Software and Application Programming",Trans.IEEE, Vol. C-17,758-770,1968.

Wing,Huang:"A parallel triangularization process of sparse matrices", Proc.International Conf.on Parallel Processing,August, 1977

Conrad,Wallach:"Iterative solutions of linear equations on a parallel processing system",Trans.IEEE,Vol.C-26,No2,1977,838-847.

Conrad,Wallach:"On block-parallel methods for solving linear equations Trans.IEEE,Vol.C-29,No5,354-359,1980.

Leven,Wallach,Conrad:"Mathematical programming methods for power dispatch",PICA-Conf.(IEEE-CH1317-3/79),1979, 137-141.

Feng:"Some characteristics of Associative/Parallel Processing", Proc. Sagamore Conf,1972, 5-16.

Evensen,Troy:"Introduction to the architecture of a 288-element PEPE" Sagamore Conf,1973, 162-169.

Shimor,Wallach:"A multibus-oriented parallel processing system-Mopps" Trans.IEEE,Vol.IECI-25,No.2,1978,137-142.

North-Holland Publ.Co.

Kober,Kopp,Kuznia:"SMS 101...",EuromicroJournal,1976,56-64.

Kober:"A fast communication processor for the SMS Multimicroproc-ssor System",Euromicro-Journal,1976,183-189.

Kober,Kuznia:"SMS-A Multiprocessor Architecture for High-Speed Numeri cal Calculations", Euromicro-Journal,Vol.5,1979,48-52.

Nagel:"Solving Linear Equations with the SMS 201",Euromicro-Journal , 1979,Vol.5, 53-54.

Tomann,Liedl:"Reliability in Microcomputer Arrays",Euromicr-Journal , 1980.

Wallach,Leven:"Alternating Sequential/Parallel Versions of the Simplex Algorithm", Euromicro-Journal, Vol.6,1980,237-242.

Richter,Wallach:"Remarks on a real-time,master-slaves operating system Microprocessing and Microprogramming, Vol.7,1981.

Pergammon Infotech

Infotech State of the Art Report "Multiprocessor Systems", Pergamon Infotech Limited, Maidenhead UK (1976)

Simulation Councils, Inc.

Korn:"Back to Parallel Computation", Simulation, August 74.

Technical Publishing Company, A Dun&Bradsteet Company

McIntire:"An Introduction to the Illiac-IV Computer",Datamation, April,1970.

Vol. 77: G. V. Bochmann, Architecture of Distributed Computer Systems. VIII, 238 pages. 1979.

Vol. 78: M. Gordon, R. Milner and C. Wadsworth, Edinburgh LCF. VIII, 159 pages. 1979.

Vol. 79: Language Design and Programming Methodology. Proceedings, 1979. Edited by J. Tobias. IX, 255 pages. 1980.

Vol. 80: Pictorial Information Systems. Edited by S. K. Chang and K. S. Fu. IX, 445 pages. 1980.

Vol. 81: Data Base Techniques for Pictorial Applications. Proceedings, 1979. Edited by A. Blaser. XI, 599 pages. 1980.

Vol. 82: J. G. Sanderson, A Relational Theory of Computing. VI, 147 pages. 1980.

Vol. 83: International Symposium Programming. Proceedings, 1980. Edited by B. Robinet. VII, 341 pages. 1980.

Vol. 84: Net Theory and Applications. Proceedings, 1979. Edited by W. Brauer. XIII, 537 Seiten. 1980.

Vol. 85: Automata, Languages and Programming. Proceedings, 1980. Edited by J. de Bakker and J. van Leeuwen. VIII, 671 pages. 1980.

Vol. 86: Abstract Software Specifications. Proceedings, 1979. Edited by D. Bjørner. XIII, 567 pages. 1980

Vol. 87: 5th Conference on Automated Deduction. Proceedings, 1980. Edited by W. Bibel and R. Kowalski. VII, 385 pages. 1980.

Vol. 88: Mathematical Foundations of Computer Science 1980. Proceedings, 1980. Edited by P. Dembiński. VIII, 723 pages. 1980.

Vol. 89: Computer Aided Design - Modelling, Systems Engineering, CAD-Systems. Proceedings, 1980. Edited by J. Encarnacao. XIV, 461 pages. 1980.

Vol. 90: D. M. Sandford, Using Sophisticated Models in Resolution Theorem Proving.
XI, 239 pages. 1980

Vol. 91: D. Wood, Grammar and L Forms: An Introduction. IX, 314 pages. 1980.

Vol. 92: R. Milner, A Calculus of Communication Systems. VI, 171 pages. 1980.

Vol. 93: A. Nijholt, Context-Free Grammars: Covers, Normal Forms, and Parsing. VII, 253 pages. 1980.

Vol. 94: Semantics-Directed Compiler Generation. Proceedings, 1980. Edited by N. D. Jones. V, 489 pages. 1980.

Vol. 95: Ch. D. Marlin, Coroutines. XII, 246 pages. 1980.

Vol. 96: J. L. Peterson, Computer Programs for Spelling Correction: VI, 213 pages. 1980.

Vol. 97: S. Osaki and T. Nishio, Reliability Evaluation of Some Fault-Tolerant Computer Architectures. VI, 129 pages. 1980.

Vol. 98: Towards a Formal Description of Ada. Edited by D. Bjørner and O. N. Oest. XIV, 630 pages. 1980.

Vol. 99: I. Guessarian, Algebraic Semantics. XI, 158 pages. 1981.

Vol. 100: Graphtheoretic Concepts in Computer Science. Edited by H. Noltemeier. X, 403 pages. 1981.

Vol. 101: A. Thayse, Boolean Calculus of Differences. VII, 144 pages. 1981.

Vol. 102: J. H. Davenport, On the Integration of Algebraic Functions. 1–197 pages. 1981.

Vol. 103: H. Ledgard, A. Singer, J. Whiteside, Directions in Human Factors of Interactive Systems. VI, 190 pages. 1981.

Vol. 104: Theoretical Computer Science. Ed. by P. Deussen. VII, 261 pages. 1981.

Vol. 105: B. W. Lampson, M. Paul, H. J. Siegert, Distributed Systems – Architecture and Implementation. XIII, 510 pages. 1981.

Vol. 106: The Programming Language Ada. Reference Manual. X, 243 pages. 1981.

Vol. 107: International Colloquium on Formalization of Programming Concepts. Proceedings. Edited by J. Diaz and I. Ramos. VII, 478 pages. 1981.

Vol. 108: Graph Theory and Algorithms. Edited by N. Saito and T. Nishizeki. VI, 216 pages. 1981.

Vol. 109: Digital Image Processing Systems. Edited by L. Bolc and Zenon Kulpa. V, 353 pages. 1981.

Vol. 110: W. Dehning, H. Essig, S. Maass, The Adaptation of Virtual Man-Computer Interfaces to User Requirements in Dialogs. X, 142 pages. 1981.

Vol. 111: CONPAR 81. Edited by W. Händler. XI, 508 pages. 1981.

Vol. 112: CAAP '81. Proceedings. Edited by G. Astesiano and C. Böhm. VI, 364 pages. 1981.

Vol. 113: E.-E. Doberkat, Stochastic Automata: Stability, Nondeterminism, and Prediction. IX, 135 pages. 1981.

Vol. 114: B. Liskov, CLU, Reference Manual. VIII, 190 pages. 1981.

Vol. 115: Automata, Languages and Programming. Edited by S. Even and O. Kariv. VIII, 552 pages. 1981.

Vol. 116: M. A. Casanova, The Concurrency Control Problem for Database Systems. VII, 175 pages. 1981.

Vol. 117: Fundamentals of Computation Theory. Proceedings, 1981. Edited by F. Gécseg. XI, 471 pages. 1981.

Vol. 118: Mathematical Foundations of Computer Science 1981. Proceedings, 1981. Edited by J. Gruska and M. Chytil. XI, 589 pages. 1981.

Vol. 119: G. Hirst, Anaphora in Natural Language Understanding: A Survey. XIII, 128 pages. 1981.

Vol. 120: L. B. Rall, Automatic Differentiation: Techniques and Applications. VIII, 165 pages. 1981.

Vol. 121: Z. Zlatev, J. Wasniewski, and K. Schaumburg, Y12M Solution of Large and Sparse Systems of Linear Algebraic Equations. IX, 128 pages. 1981.

Vol. 122: Algorithms in Modern Mathematics and Computer Science. Proceedings, 1979. Edited by A. P. Ershov and D. E. Knuth. XI, 487 pages. 1981.

Vol. 123: Trends in Information Processing Systems. Proceedings, 1981. Edited by A. J. W. Duijvestijn and P. C. Lockemann. XI, 349 pages. 1981.

Vol. 124: W. Polak, Compiler Specification and Verification. XIII, 269 pages. 1981.

Vol. 125: Logic of Programs. Proceedings, 1979. Edited by E. Engeler. V, 245 pages. 1981.

Vol. 126: Microcomputer System Design. Proceedings, 1981. Edited by M. J. Flynn, N. R. Harris, and D. P. McCarthy. VII, 397 pages. 1982.

Voll. 127: Y.Wallach, Alternating Sequential/Parallel Processing. X, 329 pages. 1982.

此为试读,需要完整PDF请访问: www.ertongbook.com

CONTENTS

# 1. INTRODUCTION

Aims: To define the terms throughput, speedup, availability, reliability, parallelism, pipelining, etc. To start classification of systems and introduce "Alternating Sequential/Parallel" - ASP processing. To set-up a list of topics to be discussed later.

## 1.1 AN OVERVIEW

This book deals with parallel hardware and software and it is only fitting to start it by mentioning the reasons for wanting to connect processors and execute programs in parallel. What we try to achieve are:

- Higher speeds
- Lower costs
- Better reliability and availability as well as
- Modularity.

Let us discuss each of these goals before proceeding to the problems of parallel systems.

The speed of computers has increased by leaps and bounds since their introduction in the 1940's. Unfortunately, it seems difficult or costly to increase the speed at the same pace much longer. A simple calculation will show that because of basic physical laws we seem to be approaching the upper limit of the speed at which a digital computer can transfer information. To this end, let us compare the (hardware) speeds in the past to those possible in the future. An addition (32 bit-words) required about 300 milliseconds in 1944 (on a relay computer), 300 microseconds in 1954 (on a tube computer) and 300 nanoseconds in 1964 (on CDC 6600). Suppose we try to build a computer with an addition time of 300 picoseconds. Since the speed at which an electrical signal travels is 0.03 cm/psec, it would propagate only 9 cm (less than 4") during the entire execution time of the instruction. It will be difficult to achieve such extremely small propagation delays or reduce so much the distances (except, maybe in VLSI). If we want to increase the speed, an attempt must be made to find other solutions rather than always reducing the circuit execution or propagation time.

One such direction is connecting processing elements in parallel. Stated simply, we hope that connecting p processors will increase the overall speed acceptably close to p times that of a single processor.

The idea is not new: In 1842 Manabrea described the lectures by Babbage [MM61] and wrote, "... the machine can...give several results at the same time, which will greatly abridge the whole amount..." of time. This amounts to parallelism, but since in modern times a number of terms were coined for parallelism, we should sort them out before we discuss parallel systems in more detail.

In general purpose (computer center) environments, the term "throughput" is used. It means simply the number of different and separate jobs that a processor is able to process in a given time. Hopefully with p processors it will be higher than with a single processor (and also correspondingly more expensive). Note though, that if we install p separate computers, the throughput will increase p-fold, but the turnaround time for any individual user will not change appreciably.

Another term used in connection with parallelism is that of "multiprogramming". The idea is to process a number of jobs in a single cpu and a number of I/O devices in an overlapped fashion. The turnaround time should be considerably lower despite the fact that only a single cpu is available.

In multiprogramming the attempt is made to decrease the time to put through a number of jobs. In contradistinction, with "speedup", the time it takes to put a single program through the system should be reduced.

In order to distinguish them even more, we might differentiate between "concurrency" and "parallelism". Concurrency should mean that unrelated events are executed at the same time, i.e., data is transferred from a disk to a line printer while the processor works on the same or some other program. Processing all bits of a word or p parts of a single program simultaneously is parallelism.

We can define space- and time-related parallelism. In the first case a number of geographically distributed processors may work on the same job, whereas, in the second case various stages of processing the same instructions are "pipelined".

To sum up: In this book we are concerned with achieving speedup (not throughput) by space parallelism at a reasonable cost and not with concurrency or pipelining.

As long as the only processors available on the market were of the "dinosaur" variety, the cost of connecting a number of them was prohibitive. Moreover, the speedup does not increase necessarily with price. Since priority arbiters will allow only a single access of a processor to a memory block at any given time, some processors will have to wait ("memory contention"). Hence, the speedup will increase less than p, the price by p and the solution may be counterproductive.

With the appearance of minicomputers, the price/performance comparison of uni/multiprocessors has changed. Minicomputers (and even more so, microprocessors) appear to be at least an order of magnitude more cost-effective than the larger machines. There are a number of apparent reasons [Fu 76] for this phenomenon.

"1. Continuing advances in semi-conductor technology favor the small processor. LSI (Large-Scale-Integration) memory, arithmetic and logic chips have been able to dramatically cut the cost of producing minicomputers. Recent LSI advances such as the Intel 3000 bit-slice processing element and the DEC LSI-11 will continue to drive down the price of minicomputers. The larger processors that rely on specialized logic to speed-up arithmetic functions, prefetch and buffer instructions, overlap instruction execution, etc. are less able to exploit the present

(LSI) technology.

2. Economics of scale. A product line that produces on the order of 10,000 minicomputers a year (or $10^5$ to $10^6$ microcomputers a year) will not have the overhead per computer that a product line has that produces 50 to 100 (large uniprocessor) computer systems a year.

3. Pricing policies bury the cost of software development for the large computer systems in the price of the hardware."

We might add here some remarks, mostly to account for recent developments. So, for instance INTEL commenced the iAPX432 -a 32-bit processor (thus opening the "floating point" market for microcomputers). On the other hand when discussing prices, one should not forget software; software costs more than hardware. Both hardware and software costs consist of design and implementation costs which are reduced by high volume. Even in conventional computers (say, micro) large quantities can be sold so that both hardware and software costs can be ammortized. The point is, that parallel systems use mass-produced LSI-units and may achieve it easier.

The analysis in [Fu 76] shows a multiprocessor to have a cost performance advantage of three to four over uniprocessor systems when implementations using similar technologies are considered. This comparison is shown to be very sensitive to memory prices and considerable attention should be given to normalizing memory costs between the two systems.

The above considerations apply even more now because of the rapid development and cost-decrease of microprocessors. Grosh's Law and Minsky's Conjecture (which claimed that the cost of parallel processors would not be competitive), were dispelled in [TP 73]. In particular, it is noted that a parallel "Pepe"-system with enough processing elements to provide the "Mips"-capability of a CDC7600 would cost only 10% of a CDC7600. Parallel processing nowadays would use microprocessors or VLSI units, stay in the framework of present-day technology and therefore be even more cost-effective than indicated above.

Another reason for judging a parallel processing system (to be called from now on a pps) to be more priceworthy is as follows. Up to the present, computers were bought and installed as single units. Whenever a larger or faster computer was needed, the old computer had to be abandoned and a new one bought (or leased). It seems a much better approach to use a pps and add processing power when and if needed. This may be called "incremental augmentation" or "modularity". It allows matching of a specific architecture to the needs of the customer.

The most important advantages of pps' are not their speedups or cost/effectiveness, but their availability and reliability. This applies to process-control applications with which this book is concerned.

In process control a failure can have a much more dramatic effect on the safety of the staff, the damage value (in M$) and on the loss of vital data than in a "computer center" environment. This explains why it is so important to have a system

available all the time and be able to rely on its proper functioning.

A unicomputer is as reliable as the weakest of its parts. Also, when a single computer is used, its failure is catastrophic, whereas in pps the remaining processors are potentially available and could work undisturbed, yielding what is called (gracefully) degraded service. This seems to have been achieved in the case of the STAR-system [RLT 78].

It is sometimes required to remove a part of a pps either semipermanently for maintenance or because a fault was detected in one of its parts. The system should nevertheless be available and continue its service at only a slightly slower pace. Moreover, if possible, this continuity of service should be achievable without expenditure on redundant components as is done in some telephone exchanges or other "fault tolerant" systems. In these systems the hardware is duplicated or triplicated and all results are compared. Reliability is ensured by modular redundancy and majority voting (say 2 out of 3) but is expensive and thus increases the cost/effectiveness. We will try other solutions.

The definitions we will use are as follows: Reliability is the capability to identify and remove a faulty component. Availability is the ability to provide service even after the removal of a faulty component. We discuss them in more detail later.

The reasons for parallel processing, namely speed, cost/effectiveness, reliability and availability are interrelated. For instance, it seems difficult to put a price on the increased reliability achieved through the use of parallelism in air-traffic controllers. Even the shut-down of flight reservations because of an unavailable computer has a price. The increased speed of power dispatch computers could have possibly prevented a blackout so that speed also influences price.

We have mentioned the advantages of pps, namely, availability, reliability, speed, cost/effectiveness and modularity. Still, the progress of parallelization is not impressive [En 77]. What are the reasons?

It was once stated [BS 76] that parallel processors are not being installed, despite their apparent advantages, primarily because:

"1. The basic nature of engineering is to be conservative. This is a classical deadlock situation: we cannot learn how to program multiprocessors until such systems exist; a system will not be built before programs are ready.

2. The market does not demand them. Another deadlock: how can the market demand them since the market does not even know that such a structure can exist? IBM has not yet blessed the concept."

To this inertia we would like to add the following problems attributed to pps':

- Restricted area of applications
- Unavailability of "parallel" mathematics
- Ignorance about possible problem decomposition
- Large expenditures projected for language and program development.

此为试读,需要完整PDF请访问: www.ertongbook.com