Lecture Notes in Computer Science

Edited by G. Goos and J. Hartmanis

111

CONPAR 81

Conference on Analysing Problem Classes and Programming for Parallel Computing
Nürnberg, June 1981
Proceedings

Edited by Wolfgang Händler

Springer-Verlag
Berlin Heidelberg New York

Lecture Notes in Computer Science

Edited by G. Goos and J. Hartmanis

111

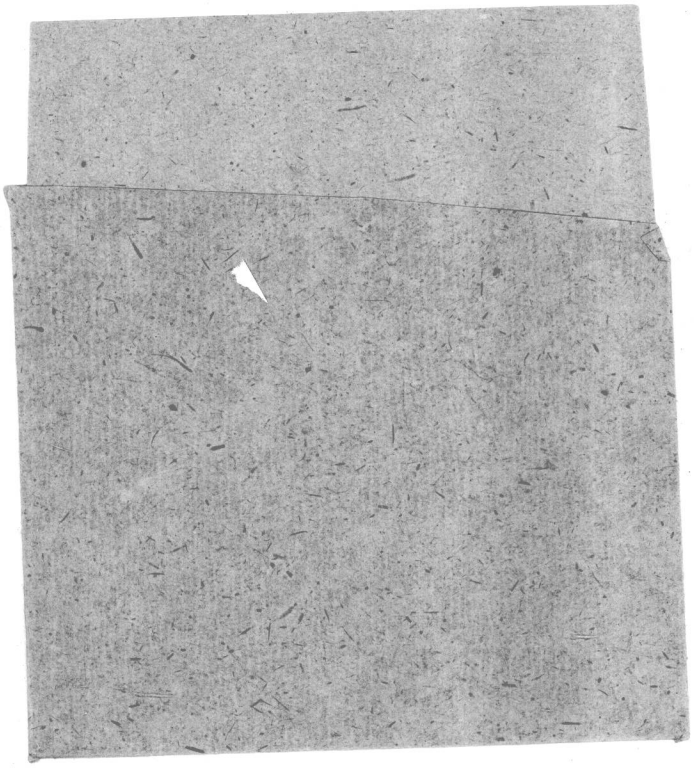
CONPAR 81

Conference on Analysing Problem Classes and Programming for Parallel Computing
Nürnberg, June 10–12, 1981
Proceedings

Edited by Wolfgang Händler

Springer-Verlag
Berlin Heidelberg New York 1981

Editor
Prof. Dr. rer. nat. Wolfgang Händler
Universität Erlangen-Nürnberg
Institut für Mathematische Maschinen und Datenverarbeitung
Martensstr. 3, 8520 Erlangen



AMS Subject Classifications (1979): 68 B 99
CR Subject Classifications (1981): 4.9

ISBN 3-540-10827-0 Springer-Verlag Berlin Heidelberg New York
ISBN 0-387-10827-0 Springer-Verlag New York Heidelberg Berlin

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, λ-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.

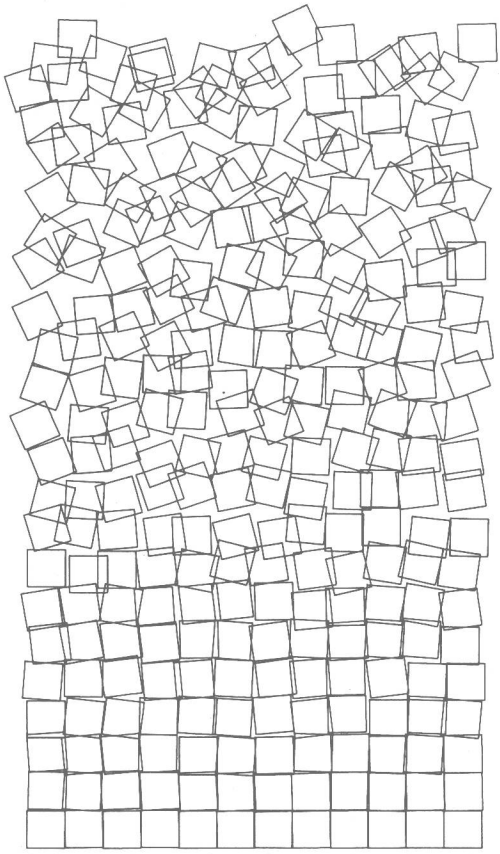
Vol. 73: Graph-Grammars and Their Application to Computer Science and Biology. Proceedings 1978. Edited by V. Claus, H. Ehrig and G. Rozenberg. VII, 477 pages. 1979.

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.

CONPAR 81



Die Graphik wurde mit einer Siemens-Datenverarbeitungsanlage erstellt.
Computer Graphic from: Georg Nees; Generative Computergraphik.

P R E F A C E

Wolfgang Händler
General Chairman

In its title this conference differs from some similar events dealing with parallelism in computer systems and with distributed computing. Such conferences discuss structures which are proposed for the solution of problems by the computation of particular algorithms, but are only useful for these problems. Some more sophisticated structures are useful in broader classes. Finally there are good reasons to expect in the future <Adaptable Architectures> [1, 2].

Nevertheless there is a lack in knowledge about the very nature of algorithms, their partitionability in principle into parallel constituents and about the way in which such algorithms can be dealt with, e. g. by <divide and conquer> methods. An interesting hint was given by C. R. Vick [3]:

> I've always felt that the challenge to map an inherently parallel problem space into a parallel solution space with as few artifical transformations as possible represents one of the most interesting challenges ... (1978).

The German pioneer in computing, Konrad Zuse [4], developed similar ideas (1969). He argues that the contemporary procedure is very often a roundabout way. For example one endeavours to transform an ultimately discrete problem into an analytic, i. e. continuous, approach and to discretize it again for solving it by a (digital) computer. He recommends a direct procedure using what he calls a <Computing Space>.

It seems to be a worthwile goal for CONPAR 81, to investigate general methods, examples, or case studies, which center on the problem, how parallel algorithms (as a general term) can be implemented and utilized for higher throughput, speed, and fault-tolerant computing. In such a way the Program Committee and the staff of IMMD interpreted the commission from the "Gesellschaft für Informatik" to organise CONPAR 81.

The responsibility of the Program Committee turned out to be not an easy one. In accordance with the decision of the committee not to admit 'parallel sessions' during the conference, we had to select 29 papers from a total of 80 submitted papers. Despite the good quality of some papers we had to reject them because they did not fit the declared aim of CONPAR 81.

We succeeded in getting contributions from distinguished experts in the field, accordingly announced as 'invited speakers'. In this context I welcome in particular our outstanding keynote speaker, Prof. Arthur W. Burks, who directed our attention at an early point of time to the activities of the late John von Neumann [5] concerning cellular automata and also 'growing' automata. Being one of the best known pioneers of the computer scene, A. Burks is at the same time a distinguished philosopher, which made it particularly valuable to receive his contribution.

Originally we planned to hold the conference at the Campus Erlangen-South of the University Erlangen-Nürnberg, where activities in parallelism are located. Unfortunately another conference with the same date in the city of Erlangen prevented this.

Nevertheless Nuremberg (Nürnberg) is an excellent alternative which offers a great spectrum of other opportunities, and we hope that the participants enjoy the medieval and stimulating atmosphere around the conference site.

R E F E R E N C E S

[1] Vick, C. R., S.P. Kartashev and S. I. Kartashev:
Adaptable Architectures for Supersystems, Computer 13 (1980)
pp. 17 - 35

[2] Händler, W., F. Hofmann and H. J. Schneider:
A general purpose array with a broad spectrum of applications
in: Computer Architecture, Workshop of the Gesellschaft für
Informatik, Erlangen, May 1975, Berlin, Heidelberg, New York
Springer 1976

[3] Vick, C. R.:
Research and Development in Computer Technology,
How do we follow the last Act (keynote address)
Proceedings 1978 International Conference on Parallel
Processing. IEEE pp. 1 - 5

[4] Zuse, Konrad:
Rechnender Raum (Computing Space)
Schriften zur Datenverarbeitung. Bd. 1
Braunschweig: Vieweg und Sohn 1976

[5] Burks, Arthur W. (edit.)
Essays on Cellular Automata
(To the memory of John von Neumann)
Urbana, Chicago, London:
University Illinois Press 1970

(Citation not exhaustive)

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: L. Bolc, Z. Kulpa, Digital Image Processing Systems. 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.

TABLE OF CONTENTS

SESSION 6, PARALLELISM OF NUMERICAL ALGORITHMS

PART I

PART II

PROGRAMMING AND STRUCTURE CHANGES IN PARALLEL COMPUTERS

Arthur W. Burks
Department of Computer and
Communication Sciences
The University of Michigan
Ann Arbor, Michigan 48109

1. Introduction

It is perhaps appropriate in a keynote paper to look at programming for parallel computers from a general point of view. What, we may ask, is computer architecture all about? Well, there are domains of problems to be solved and there are available hardware building blocks. The architecture of a computer is the way in which these building blocks are organized, and is to be judged on how well that organization is adapted to the given class of problems.

This conference is directed to the issue of parallel problems. These are problems which can be solved efficiently and rapidly on a computer capable of carrying out many interacting streams of computation simultaneously. But any parallel problem can also be computed serially. Hence, to understand the domain of parallel problems we need to study its relation to other problem domains. One of the architectural choices to be made is that of specialization vs. generalization: Should special machines be made for parallel computations?

Consider the Illiac IV, the most powerful parallel computer of its time. Looking back one can ask: As a working computer, was it an economic success? Has it solved important problems that could not otherwise be solved? Can it solve parallel problems more cheaply than other computers? If the answers are negative, one should then ask: At the time, was building a machine the best way to answer these questions? Would the answers be different if the machine had been built at another time?

It is characteristic of the computer industry, and a reflection of the rapidity of the computer revolution, that these questions are highly time-dependent. The Illiac IV would have been impossible five years earlier, but much easier a chip generation later. Today the time is ripe for the development of many useful kinds of parallel computers. By its nature, parallelism requires many small computing subsystems, with many fast cross-connections between them. Both are feasible with very large-scale integrated circuits.

The situation was quite different when electronic computers were born, at which time the most efficient computers were highly serial. As Dr. Händler has stressed in his call to this meeting, there is a vast architectural distance between "the conventional Princeton-type computer" and current concepts of multiprocessing, array and cellular computers, and other parallel organizations. Let me make some historical remarks on this topic.

I will first compare the old method of rearranging the parts of a machine for each new problem, as by the use of a plugboard, with the modern method of programming a machine for each problem. Then I will discuss the architecture of the first stored program computers.

2. Machine Assembly versus Machine Programming

Imagine that you have only one problem to solve, that it is to be solved many times with only a variation of input conditions, and that your technological means consists of an indefinite number of primitive building blocks at a fairly low logical level: switches, registers, adders, connecting cables, etc. To solve your problem you assemble a machine from these primitives. In a sense, you design a new machine for each problem.

Let us call this "the machine assembly method" of solving problems. No doubt this approach seems far out, a Tinkertoy or Meccano method of computing as compared to the use of programs. But at one time and for certain purposes it was the best approach. The most powerful computers in the period 1925 to 1945 all used it: the electrical network analyzer, differential analyzer, and the ENIAC.

The competitors of these machines were the electromechanical computers of Konrad Zuse, Aiken and IBM, and Stibitz of Bell Laboratories. These were programmed with punched paper tape, and were thus limited by the slow speed of their tape readers. Machines with paper tape programs were an important step on the way to the stored program computer, but they were not in general superior to machines that used the machine assembly method of solving problems. It should also be noted that the two methods sometimes overlapped. For example, the new MIT differential analyzer, though it calculated mechanically, was set up in a few minutes from punched paper tapes. Incidentally, the differential analyzer was a parallel computer to the core, for all its parts had to work simultaneously.

Despite its being archaic, I think there is something to be learned from the concept of assembling a machine to solve a problem. For any given algorithm there are many possible machines that will execute it,

so that the basic architectural question is: Which of these machines is best? This question can be partially formalized by assigning a cost to each building block and asking: Which machine has the minimal cost? But like any formalization, this one has its limitations.

The first limitation concerns what is left out. The most important omitted characteristics are uniformity and simplicity of assembling the machine to do a particular problem, and of debugging and maintaining it. Humans are involved in these tasks, and uniformity and simplicity are aids to human understanding; moreover, the ratio of labor cost to hardware cost is rising rapidly, as we are all well aware.

The second limit on this formalization concerns complexity. Though our optimality problem can be defined precisely for any particular system of primitive building blocks, because of the complexity of computers it cannot be solved either mathematically or computationally for interesting cases. Hence machine assembly formalisms are valuable mainly as points of view. The same is true, in my opinion, of many formalizations in computer science. In the field of computer architecture it is certainly the case that formalisms are limited to a conceptual role, for actual architectures can be evaluated only by experience and simulation.

With these limitations on the value of a formal approach to architecture in mind, let us continue our comparison of the machine assembly method of solving problems with the programming method. The former created a new architecture for each problem, which might be optimized for that problem. In contrast, the architecture of a programmable computer has to be optimized over its whole class of problems. Of course, there was a greater set-up cost in the machine assembly method.

The relative merits of the two methods changed with the state of technology. While both were important from 1925 to 1945, after capacious electronic stores were developed in the mid and late 1940's, making the stored program computer possible, the programming method quickly came to dominate. Electronic analog computers with plugboards for problem set-up persisted for some time, but they are rare now.

The plugboard method of problem set-up is forever dead, because manual and electromechanical technologies are outdated. Of course, we still use these technologies at a high architectural level, as when we add a second processor, a new terminal, etc., or augment a minicomputer by plugging in another disk drive. But these changes, being high-level, are infrequent, whereas the changes made by the plugboard method were low-level, and hence frequent.

However, the idea of radically restructuring a machine for a new problem, or a new group of problems, still has merit. I intended the

expression "machine assembly method of solving problems" to cover any such radical restructuring. The plugboard method of earlier computers, and the interconnection method of the differential analyzer, were electrical and mechanical ways of machine assembly, suitable for their era but now outdated.

Compiling a machine at the machine assembly level is much more difficult than compiling a program. However, technology changes rapidly. Modern chip manufacture is much more like book printing than was the construction of vacuum tube machines. Printed books will eventually be replaced by hard computer copy. Maybe these kinds of development will someday make it economical to produce specific computers for particular problems. We can imagine an automated manufacturing system that receives an algorithm as input and produces as output a computer which executes that algorithm efficiently. But even if that never comes to pass, there may be other ways of radically restructuring computers. I will return to this topic near the end of my paper.

3. The Architecture of the First Stored Program Computers

The stored program computers grew directly out of the ENIAC. The architectures of the two machines were radically different in a way that is relevant to this conference, so it may be instructive to say a few words about them.

The ENIAC was highly parallel and highly decentralized. It had twenty-seven different computing and input-output units, each with its own local program controls. There was a central program control unit for supervisory management. For its version of the machine assembly method of solving problems, it had a vast plugboard-like switch running past all the units; to structure the system to solve a problem, the operator set switches on the program controls and manually interconnected the units to one another via the switches in a manner appropriate for that problem.

In theory it was possible to set up the ENIAC so that all the units operated simultaneously. In practice, this potentiality for parallelism was not as useful as had been anticipated. The ENIAC's complete parallelism was rarely employed, and arranging it for partial parallelism added to the burden of the operator. Thus users of the first general-purpose electronic computer met two problems basic to parallel processing: Which algorithms can be efficiently executed in parallel, and which not? What is the best way to plan a parallel computation and to structure or program a machine to execute it?

The ENIAC's parallelism was relatively short-lived. The machine

was completed in 1946, at which time the first stored program computers were already being designed. It was later realized that the ENIAC could be reorganized in the centralized fashion of these new computers, and that when this was done it would be much easier to put problems on the machine. This change was accomplished in 1948. Since the original ENIAC employed the machine assembly method of solving problems, very little additional equipment was required to convert it to a centrally programmed machine. Thereafter the plugboard of the ENIAC was never modified, and the machine was programmed by setting switches at a central location. Thus the first general-purpose electronic computer, built with a parallel decentralized architecture, operated for most of its life as a serial centralized computer! Ironically, the fact that the ENIAC incorporated the machine assembly method of problem solving made it very easy to transform the machine into a computer that did not use this method.

The jump from the ENIAC to the stored program computer was an important historical event, for modern computers are revolutionizing human life. The question of who invented the stored program computer is therefore of interest, and as you may know, has been hotly debated for a long time. This is not the place for a careful analysis of the contributions of the participants, but since the architecture of the Princeton machine is still taken as a paradigm, albeit an out-of-date paradigm, a few remarks are appropriate.

First of all, the "time is ripe" theory of discovery and invention applies to the stored program computer. This is the theory that whenever an appropriate combination of materials, methods, problems, and needs coalesces, invention is likely to result. Frequently, when the time is ripe, there are independent discoveries of the same item, though not always. Many examples come to mind of independent and nearly simultaneous discovery. I'll mention a few cases that involved contributors from our host country: the invention of the adding machine (Schickard and Pascal), the creation of the calculus (Newton and Leibnitz), invention of the telegraph (Gauss, Morse with Henry), the discovery of Neptune through calculations from the perturbations of Uranus' orbit (Leverrier with Galle, Adams), the invention of the general-purpose electromagnetic computer (Zuse, Aiken with IBM, Stibitz with Bell Laboratories), and the conception of the general-purpose electronic computer (Schreyer with Zuse, Atanasoff with Mauchly and Eckert).

The stored program computer was not a case of independent discovery on the part of several people or institutions. But it did arise out of a background of pre-electronic and electronic digital computing in the United States which involved much original discovery, some of it