Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.fw001

Supercomputers in Chemistry

In Supercomputers in Chemistry; Lykos, P., el al.;
ACS Symposium Series; American Chemical Society: Washington, DC, 1981.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.fw001

In Supercomputers in Chemistry; Lykos, P., el al.;
ACS Symposium Series; American Chemical Society: Washington, DC, 1981.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.fw001

Supercomputers in Chemistry

Peter Lykos, EDITOR
Illinois Institute of Technology

Isaiah Shavitt, EDITOR
Ohio State University

Based on a symposium
cosponsored by the ACS Divisions of
Computers in Chemistry and Physical
Chemistry and the U.S. National
Resource for Computation in Chemistry
at the Second Chemical Congress
of the North American Continent
(180th ACS National Meeting),
Las Vegas, Nevada,
August 26–27, 1980

ACS SYMPOSIUM SERIES **173**

AMERICAN CHEMICAL SOCIETY
WASHINGTON, D. C. 1981

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.fw001

Library of Congress CIP Data

Supercomputers in chemistry.
(ACS symposium series, ISSN 0097–6156; 173)

Papers presented at the Symposium on Supercomputers and Chemistry.
Includes bibliographies and index.

1. Chemistry—Data processing—Congresses.
I. Lykos, Peter. II. Shavitt, Isaiah, 1925– . III. American Chemical Society. Division of Computers in Chemistry. IV. American Chemical Society. Division of Physical Chemistry. V. Lawrence Berkeley Laboratory. National Resource for Computation in Chemistry. VI. Symposium on Supercomputers and Chemistry (1980: Las Vegas, Nev.). VII. Series.

QD39.3.E46S93 542'.8 81-17630
ISBN 0-8412-0666-X AACR2 ACSMC8 173 1-278
1981

Copyright © 1981

American Chemical Society

All Rights Reserved. The appearance of the code at the bottom of the first page of each article in this volume indicates the copyright owner's consent that reprographic copies of the article may be made for personal or internal use or for the personal or internal use of specific clients. This consent is given on the condition, however, that the copier pay the stated per copy fee through the Copyright Clearance Center, Inc. for copying beyond that permitted by Sections 107 or 108 of the U.S. Copyright Law. This consent does not extend to copying or transmission by any means—graphic or electronic—for any other purpose, such as for general distribution, for advertising or promotional purposes, for creating new collective work, for resale, or for information storage and retrieval systems.

The citation of trade names and/or names of manufacturers in this publication is not to be construed as an endorsement or as approval by ACS of the commercial products or services referenced herein; nor should the mere reference herein to any drawing, specification, chemical process, or other data be regarded as a license or as a conveyance of any right or permission, to the holder, reader, or any other person or corporation, to manufacture, reproduce, use, or sell any patented invention or copyrighted work that may in any way be related thereto.

PRINTED IN THE UNITED STATES OF AMERICA

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.fw001

ACS Symposium Series

M. Joan Comstock, *Series Editor*

Advisory Board

David L. Allara
Kenneth B. Bischoff
Donald D. Dollberg
Robert E. Feeney
Jack Halpern
Brian M. Harney
W. Jeffrey Howe
James D. Idol, Jr.
James P. Lodge
Marvin Margoshes
Leon Petrakis
Theodore Provder
F. Sherwood Rowland
Dennis Schuetzle
Davis L. Temple, Jr.
Gunter Zweig

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.fw001

FOREWORD

The ACS SYMPOSIUM SERIES was founded in 1974 to provide a medium for publishing symposia quickly in book form. The format of the Series parallels that of the continuing ADVANCES IN CHEMISTRY SERIES except that in order to save time the papers are not typeset but are reproduced as they are submitted by the authors in camera-ready form. Papers are reviewed under the supervision of the Editors with the assistance of the Series Advisory Board and are selected to maintain the integrity of the symposia; however, verbatim reproductions of previously published papers are not accepted. Both reviews and reports of research are acceptable since symposia may embrace both types of presentation.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.pr001

PREFACE

The computing milieu has been evolving in recent years in two somewhat disparate but complementary directions. First, the availability and power of relatively cheap minicomputers have increased enormously. Such computers are particularly cost effective in single-laboratory or single-department environments that have homogeneous groups of fairly sophisticated users. On the other hand, the first few of a new generation of extremely powerful supercomputers have been appearing in research facilities, and experience has begun to accumulate on how to adapt thinking and problem-solving strategies to the novel characteristics of these machines.

Chemistry has long been one of the primary application areas for computers in scientific research. Quantum chemistry, in particular, has consumed vast quantities of computer time on machines of all sizes, and its potential for expanded computer resources is notorious. Furthermore, other fields of computational chemistry have been maturing, and their increasing demands for problem solving power have also been coming up against the constraints and limitations of current machines. Thus, the minicomputer explosion has been involving more areas of chemistry and more chemical researchers in the routine use of computers, but for some significant research tasks, the capabilities of the most powerful computers are essential and often even inadequate.

The Symposium on Supercomputers and Chemistry was organized because we believed that the time was opportune to bring together computer professionals and chemical researchers from several fields to discuss the needs, the opportunities, the accumulating experience, and the novel characteristics of supercomputers in chemical research. This symposium was held at the National Meeting of the American Chemical Society in Las Vegas, Nevada, in August 1980, under the cosponsorship of the ACS Division of Computers in Chemistry, the ACS Division of Physical Chemistry, and the U.S. National Resource for Computation in Chemistry. The speakers included representatives of major computer vendors, several from major U.S. national and industrial research laboratories, representatives of national laboratories in Britain and Japan, and others from universities in the United States, Britain, and Canada.

Several papers focused on the design characteristics of recent and developing computer systems for large-scale applications. Others described the accumulating experience in the utilization of existing supercomputers in chemical research, and particularly on the way in which their vector-oriented architecture mandates new thinking on problem solving strategies and requires code reorganization in adapting existing computer programs to the new machines. Many others discussed research applications in which computational power requirements are barely met, or still unmet, by most currently available computers. Still others discussed the man–machine interface in the age of the supercomputer. Fifteen of those papers are collected in this volume.

PETER LYKOS
Illinois Institute of Technology
Department of Chemistry
Chicago, IL 60616

ISAIAH SHAVITT
Ohio State University
Department of Chemistry
Columbus, OH 43210

August 17, 1981

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.pr001

The Use of Vector Processors in Quantum Chemistry

Experience in the United Kingdom

MARTYN F. GUEST and STEPHEN WILSON

Science and Engineering Research Council, Daresbury Laboratory,
Daresbury, Warrington WA4 4AD, U.K.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch001

The purpose of this paper is to review the impact which vector-processing computers(1,2) are having on ab initio quantum chemical calculations giving special emphasis to experience gained in the United Kingdom. The advent of such powerful computational tools is having, and will continue to have, an important influence on computational quantum chemistry. Calculations which were very time consuming are becoming routine; calculations which were impossible are now tractable.

This review is necessarily selective, and is divided into several sections. Initially we give an overview of the availability of supercomputers in the U.K., and summarise the experience gained in the implementation of various quantum chemistry packages. Optimization of Quantum Chemistry codes on the CRAY-1 is considered, with integral evaluation, self-consistent-field and integral transformation phases of quantum chemical studies being considered together with some aspects of the correlation problem.

The significant impact of the CRAY-1 in several areas of electronic structure research is then outlined, with particular attention given to the evaluation of the components of electron correlation energy which may be associated with higher order excitations and to the development of basis sets suitable for accurate studies. This is followed by some concluding remarks.

Supercomputers in the United Kingdom

The CRAY-1 vector processing computer at the Science Research Council's (S.E.R.C) Daresbury Laboratory, is at the centre of a network providing large scale computational facilities for Universities in the United Kingdom. This is the only supercomputer available at present to Quantum Chemists in the U.K., and this article will therefore be restricted to experience gained on the CRAY-1, although this experience will undoubtedly be relevant to future applications on machines such as the ICL Distributed Array Processor (DAP) (see reference (2) for a detailed description) and the CDC Cyber 203/205.

0097-6156/81/0173-0001$09.50/0
© 1981 American Chemical Society

The main characteristics of the CRAY-1 computer are shown in Table I (see also reference (2)). The scalar operations are seen to be approximately twice that of the CDC 7600 and IBM 360/195. The maximum vector capability occurs for matrix multiplication, for which the measured time on the CRAY-1 is twenty times faster than the best hand coded routines on the CDC 7600 or IBM 360/195. The maximum rate is circa. 135 Mflops (Millions of floating-point operations per second) for matrices that have dimensions which are a multiple of 64, the vector register size. The rate of computation for matrix multiplication is shown in figure 1 as a function of matrix size.

The Daresbury Laboratory CRAY-1 computer is accessed by means of an IBM 370/165 which is linked to computers at the S.E.R.C's Rutherford Laboratory, C.E.R.N. and workstations in many Universities. The S.E.R.C. network in fact incorporates links to 10 mainframe and 76 minicomputers and to 44 different sites. The CRAY-1 was installed at Daresbury for an initial period of two years, extendable for a third year. The S.E.R.C. buys an average of eight hours per day from CRAY Research Inc. Ltd., and the possibility exists that the machine will be upgraded to a CRAY-1 Model S/500. Proposals are also under consideration for the installation of supercomputers at the two largest University regional computer centres - London and Manchester - and at the S.E.R.C's Rutherford Laboratory where the existing twin IBM 360/195 machines are scheduled for replacement in 1982/3. Again these machines would be accessible via workstations in a number of University departments around the U.K.

The Daresbury Laboratory has applied the CRAY-1 to its multifaceted scientific environment since June 1979. Disciplines benefitting from the availability are numerous, with a brief summary of the scientific research involved in the period June 1979 - June 1980 being given in Table II. In this table we also present some reported improvements determined on vectorisation of various packages.

Much use of the CRAY-1 computer has been made by the S.E.R.C's Collaborative Computational Projects (C.C.P.s). These projects aim to co-ordinate the development of sophisticated software in various fields of research within the U.K. The first of these projects is concerned with electron correlation effects in molecules, and is of particular interest to Quantum Chemists.

Implementation and Performance of Quantum Chemistry Packages. A Preliminary Investigation

Perhaps the first question to be considered in contemplating the use of a vector processor in Quantum Chemistry (QC) is just how much advantage is obtainable with the minimum amount of effort i.e. by simply implementing software from a scalar machine with little or no modification. The answer to this question is readily obtainable by benchmarking the machine against some standard on a variety of widely used QC packages. Such an exercise would shed

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch001

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch001

Table I: The Main Characteristics of the CRAY-1. Facts and Figures (from "The CRAY-1 Computer System", Publication No.2240008B, 1977, Cray Research Inc.)

CPU	
Instruction size	16 or 32 bits
Repertoire size	128 instruction codes
Clock period	12.5 nsec
Instruction stack/buffers	64 words (4096 bits)
Functional units	twelve:
	3 integer add
	1 integer multiply
	2 shift
	2 logical
	1 floating add
	1 floating multiply
	1 reciprocal approx.
	1 population count
Programmable registers	8 × 64 64-bit
	73 64-bit
	72 24-bit
	1 7-bit
Max. vector result rate	12.5 nsec/unit
FLOATING POINT COMPUTATION RATES	(results per second)
Addition	80 × 10^6/sec
Multiplication	80 × 10^6/sec
Division	25 × 10^6/sec
MEMORY	
Technology	bipolar semiconductor
Word length	72 bits (64 data, 8 SECDED)
Address space	4M words
Data path width (bits)	64 (1 word)
Cycle time	50 nsec.
Size	262,144 words
	or 524,288 words
	or 1,048,576 words
Organisation/interleave	16 banks (8 banks optional)
Maximum band width	80 × 10^6 words/sec
	(5.1 × 10^9 bits/sec)
Error checking	SECDED

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch001

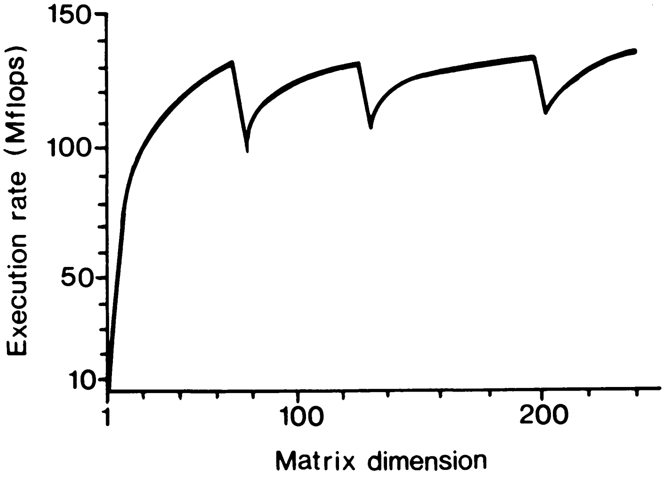


Figure 1. Matrix multiplication timing. Execution rate, in MFLOPS, is plotted as a function of vector length.

Table II: Summary of the Relative Performance of the CRAY-1 in Scientific Research at the S.R.C. Daresbury Laboratory.

Research Area	Computer used as Benchmark	Relative Performance*		
		No Modifications	After Modification	
			Total	Selected Routines
Nuclear Physics	IBM 370/165	10	50	80
Astronomy	CDC 7600	2.5	9.3	26
Protein Crystallography	ICL 2980	14		
Oceanography	IBM 370/165	11	8-34	
Atomic & Molecular Physics	IBM 370/165	4-6	16-30	
Quantum Chemistry	IBM 370/165	2-10		
Plasma Physics	CDC 7600	<15.5		
Physical Chemistry	CDC 7600	2-3	7	
Surface Physics	IBM 370/165	17		
	DEC 10	85		
	CDC 7600	4-8		

light on a variety of matters, namely (a) how man power intensive is likely to be the exercise of 'doctoring' codes to make optimum use of the potential offered by a vector processor, (b) how relevant are the algorithms commonly adopted in QC in the framework of a vector processor and, (c) as most codes are exclusively in FORTRAN, how good is the FORTRAN compiler (CFT on the CRAY). Accordingly we have implemented a variety of QC packages (3-15) on the CRAY-1 computer at the Daresbury Laboratory, as detailed in Table III.

Experience has shown that a straight implementation of code may be accomplished with 'relatively' little difficulty, although it is clear that the implementation of software which makes optimum use of the byte structure on the IBM requires somewhat greater effort (e.g. SPLICE (7)).

The machine adopted for the benchmark was an IBM 370/165, which has a CPU performance in these type of calculations of circa. 0.4 * IBM 360/195 and 0.3 * CDC 7600.

The CPU times for a variety of calculations conducted on the hydrogen sulphide molecule (H_2S) at its equilibrium separation are summarised in Table IV. All treatments employed the same near-Hartree Fock basis set, with a (13s10p2d/6s2p) primitive set contracted to 60 basis functions, denoted <10s6p2d/4s2p> (16). Note that the correlation treatments were restricted to an estimate of the valence shell correlation energy with the orbital space comprising a total of 47 active functions, corresponding to a configuration space of 4104 functions.

We stress that little significance should be attached to the absolute timings of each program : this exercise was not meant to be a benchmark of software, since the particular version of the programs implemented may well have been superceeded by more efficient versions not available at the time to the authors. The important figure is of course the relative timings between the two machines, and the picture painted by these results is regrettably somewhat depressing. It is quite clear that the figures are actually reflecting the increase in scalar performance of the two machines, and that little or no use of the vector processing capability of the machine has been realised. While this is perhaps not surprising, it does suggest that QC faces special problems not common to many of the other disciplines mentioned in Table II.

In the following section of this paper we attempt to highlight these problems, and their possible solutions for some of the steps involved in QC calculations.

Optimisation of Quantum Chemistry Codes on the CRAY

Integral Evaluation. We here confine our attention to the evaluation of integrals over Cartesian Gaussians (17). As seen in Table IV, the essentially unmodified FORTRAN source of the ATMOL3 Gaussian Integrals program compiled on the CRAY runs at approximately 6.2 times faster than the IBM 370/165 (circa. 2.4 times

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch001

Table III: Quantum Chemistry Codes Implemented on the CRAY-1 at the Daresbury Laboratory

Program	Authors	Function
ATMOL	V.R.Saunders, M.F.Guest (3)	Hartree-Fock, Integral Transformation
HONDO	M.Dupuis, J.Rys, H.King (4)	Hartree-Fock, Gradient Package
POLYATOM-2	J.W.Moskowitz, L.C.Snyder (5)	Hartree-Fock
MOLECULE	J.Almlöf (6)	Hartree-Fock
SPLICE	M.F.Guest, W.R.Rodwell, B.T.Sutcliffe (7)	Conventional CI, Bonded Functions
MOLECULE-CI	B.Roos (8)	Direct-CI: closed shells
MBPT	D.Silver, S.Wilson (14)	Many-body Perturbation Theory, 3rd-order codes.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch001

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch001

Table IV: The straight implementation of code: Timings for various Quantum Chemistry packages on an IBM 370/165 and a CRAY-1. All calculations are on the H_2S molecule.

Basis set 60 AOs, 47 in CI (Core = 5; 8 discarded)

Treatment	Package	CPU TIMING (seconds)		Improvement
		IBM 370/165	CRAY-1	
Gaussian Integral Evaluation	ATMOL	255 (210)[†]	41.2	6.2
	HONDO	332	34.3	9.7
	POLYATOM		113.0	
	MOLECULE		66.0	
Closed Shell SCF (15 cycles)	ATMOL	556 (338)	143.1[*]	3.9
	HONDO(+)	180	46.1	3.9
	POLYATOM		105.0	
	MOLECULE(++)		5.1	
Integral Transformation	ATMOL	1480 (930)	209.0[*]	7.1
Electron Correlation	MC-SCF	(1150)	155.6	>7.4
	SPLICE	1680	336.0	5.0
	MOLECULE-CI	581	101.3	5.7
	MBPT	244	66.8	3.7

[†] Figures in parentheses refer to timings with key areas of code in Assembler.

[*] Present version; SCF, 27.7 : Transformation, 26.9.

(+) Supermatrix

(++) Supermatrix plus symmetry adaption.

faster than the CDC 7600). The reason for this performance was apparent from an inspection of the central loops, few of which had vectorised. Furthermore, the average loop length of approximately 4 was not sufficient for effective use of the vector processing capability, so that when the most important loop was vectorised the speed of the machine was degraded by 10%. Since the CRAY performs optimally when the central loop is both vectorisable and of length 64 (or a multiple thereof), it is clear that the vectorisation of short loops may be counter productive. Before outlining the technique used to reorganise the code, we outline in a simplified way the original version of the code. The procedure was based on the Gauss-Rys quadrature method (14), and we explain the process for contracted Gaussian basis functions. We assume for the moment that the number of primitives within a given contraction is relatively high (e.g. >2), so that the number of different primitive contributions to a given integral will be >16. Let this number of different primitive combinations be denoted MUMAX. Also suppose that the number of Gauss-Rys quadrature points be denoted N. The maximum possible value of N is 3, 5, or 7 if P, D, or F functions respectively are involved in the basis. The central formula for evaluating a two-electron integral may be written in the form

$$I^u = \sum_{i=1}^{N} X_i^u Y_i^u Z_i^u$$

where u indexes the particular primitive combination. Thus the total integral is given through

$$I = \sum_u I^u$$

A representation of the original code to evaluate the integral is shown in figure 2. Note that although the central loop of this sequence is vectorisable on the CRAY-1, the loop range is too small for efficient processing.

The revised code is shown schematically in figure 3. Note the program actually evaluates blocks (18) of integrals simultaneously, and this feature can be used so that the equivalent of loop 3, when applied to the evaluation of the usually many integrals which are found in such a block (e.g. a [PP/PP] block contains 81 integrals), vectorises. Actually the description of the revised code presented in figure 3 has been drastically simplified for presentation purposes. The program evaluates blocks of integrals of a given type (e.g. [DP/PS]) in sequence, with an outer loop over all possible integral block types. It then arranges to compute all the auxiliaries for 64 blocks of integrals concurrently, thus allowing the computation of the auxiliaries to be vectorised with central loops of length 64 (optimal for a CRAY-1) followed by

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch001

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch001

```
    SUM=0.0

    DO 1 MU=1,MUMAX

    Compute all $X_i,Y_i,Z_i$

    for this primitive combination

    DO 1 I=1,N

  1 SUM = SUM + X(I) * Y(I) * Z(I)
```

Figure 2. Orthodox procedure for evaluating Gaussian integrals.

```
    DO 1 MU = 1,MUMAX

  1 TEMP(MU) = 0.0

    DO 2 I = 1,N

    Compute all $X_i^u,Y_i^u,Z_i^u$

    for this value of I.

    DO 2 MU = 1,MUMAX

  2 TEMP (MU) = TEMP(MU) + X(MU) * Y(MU) * Z(MU)

    SUM=0.0

    DO 3 MU = 1,MUMAX

  3 SUM = SUM + TEMP(MU)
```

Figure 3. Vectorized procedure for evaluating Gaussian integrals.

In Supercomputers in Chemistry; Lykos, P., el al.;
ACS Symposium Series; American Chemical Society: Washington, DC, 1981.

the computation of 64 sets of integrals (also in completely optimised central loop form). Of course, since it is unlikely that the number of sets of integrals required will be exactly divisable by 64, a further piece of code is required to account for this remainder using vector loops of a length less than 64. Given a sufficiently large number of integrals to evaluate, the procedure is dominated by loops of length 64, and the revised program is observed to run at 16.2 times as fast as the CDC 7600 version. Clearly if this sort of improvement can be maintained in the other modules in the wavefunction calculation, then the prospects for quantum chemical calculations are indeed enhanced.

Several general lessons may be learned from this study. The kernel of the integrals code actually operates at approximately 30 times faster than the CDC 7600, so that the non-kernel code, which takes about 10% of the computer time on a conventional machine occupies 60% of the time in our present CRAY-1 program. The implication is that work to vectorise the non-kernel components of the calculation might now be profitable.

In order to accomplish the vectorisation of the code it is necessary to compute and store 64 sets of integrals at any one time. The store required to accomplish this varies with integral block type, as follows

Block type	Store required for X,Y,Z Auxiliaries (64 sets) (kwords)
[PP/PP]	9k
[DD/DD]	76k
[FF/FF]	336k

It is clear that we have dedicated a large amount of core in order to vectorise the code, and it is probable that many algorithms share this attribute. In general it is apparent that to drive a vector processor efficiently will require a large amount of store.

It is worth pointing out that the vectorised code is written in standard FORTRAN. The CRAY FORTRAN compiler simply recognises the vectorisable loops and translates these into hardware vector orders. An inspection of the machine code thus generated revealed that very little was to be gained by hand coding the kernels into Assembly language.

Self-Consistent-Field : The Closed Shell Case. There are presently two favoured methods for computing a 2-electron interaction matrix, $G = 2J-K$:

(i) Directly from the list of 2-electron integrals

(ii) From the so called P-Supermatrix :

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch001

$$P = [ij/kl] - 1/4\,\big([ik/jl] + [il/jk]\big)$$

In a small system, where few integrals if any are 'zero' by virtue of the distance of the interactions, the size of the Supermatrix file is the same as that of the 2-electron file (we ignore for the moment the effects of molecular symmetry), and the Supermatrix method has a clear advantage because each Supermatrix element gives rise to only two references to the G operator :

$$G_{ij} = G_{ij} + P_{ij,k\ell}\, D'_{k\ell}$$

$$G_{k\ell} = G_{k\ell} + P_{ij,k\ell}\, D'_{ij}$$

where D' is the density matrix after doubling of the off-diagonal elements whilst each two-electron integral gives rise to six distinct references (assuming all 4 indices are non-coincident)

$$G_{ij} = G_{ij} + [ij/k\ell] * D'_{k\ell}$$

$$G_{k\ell} = G_{k\ell} + [ij/k\ell] * D'_{ij}$$

$$G_{ik} = G_{ik} - 1/2[ij/k\ell] * D_{j\ell}$$

$$G_{i\ell} = G_{i\ell} - 1/2[ij/k\ell] * D_{jk}$$

$$G_{jk} = G_{jk} - 1/2[ij/k\ell] * D_{i\ell}$$

$$G_{j\ell} = G_{j\ell} - 1/2[ij/k\ell] * D_{ik}$$

Two further difficulties are faced by the method using the integrals directly

(a) The pair indices (eg ij=i*(i-1)/2 + j) must be computed from the orbital indices, (unless one proposes to follow Duke (19) and store all 6 pair indices with the integral, causing an increase in disk storage requirement by a factor of 160/96=1.67 - allowing 64 bits for the integral, 16 bits for a pair index and 8 bits for an orbital index)

(b) the cases of coincident indices must be handled correctly giving rise to conditional statements when processing integrals which have no counterpart when using supermatrices. (Billingsley (20) has suggested a procedure of ordering the integrals into blocks of similar index coincidence type and processing each block by a different algorithm to circumvent the problem).

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch001

All the above remarks are borne out by timings obtained from the CRAY version of the ATMOL closed shell SCF program, which has neither the Duke or Billingsley algorithms, and gives the following timings (quoted in CRAY clocks - 1clock = 12.5 nsecs).

	Integer and Logical Arithmetic	Floating Point Arithmetic	Total
Integral Method	23	100	123
Super Matrix Method	3	35	38

In the above the integer and logical component of the integral method, comprising the fetching of the integral, unpacking of the orbital indices, computation of the pair indices and the execution of conditionals to account for orbital index coincidences (there are actually 3 such), occurs entirely in the vector hardware. Note that the quantities i*(i-1)/2 are computed as needed, and not, as in conventional treatments, fetched from a precomputed array MAP(i)=i*(i-1)/2 because the former can be vectorised, the latter not, at least in the CRAY, due to the absence of a hardware gather order capable of implementing

```
I(J) = MAP(IND(J)) ; J=1,N
```

The corresponding integer and logical component of the Supermatrix method consists of simply fetching the Supermatrix element and unpacking the pair indices. The Floating Point arithmetic in both algorithms is peformed in the scalar hardware, the details of the algorithm being so arranged that the segmented nature of the functional units is taken advantage of. Both are coded in CAL assembly language, the best coding in FORTRAN being circa. three times slower in both cases, although our original FORTRAN versions as taken from the CDC 7600 were approximately six times slower.

The above timings represent computation rates of 7.8 and 8.3 Mflops respectively for the Integral and Supermatrix driven methods, very far from what we regard as the limit of 135 Mflops of which the CRAY is capable, with the Supermatrix method being faster by a factor of 3.2 in the non-sparse case. In order that either algorithm be non I/O bound, transfer rates of approximately 8 and 25 Mbytes will be required for Integral and Supermatrix driven methods respectively. The maximum transfer rate on a single CRAY channel is 4.4 Mbytes/sec., indicating the requirement for a sophisticated I/O code capable of driving more than one channel simultaneously, each at high efficiency.

We next comment on the effect of sparsity in the integral list (apart from that caused by point group symmetry) on the relative performance. It is obvious that sparsity considerations are

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch001

of the utmost importance when considering an extensive system because the number of integrals rises in proportion to the fourth power of the basis set size, whilst the number of integrals whose magnitude is above any preset tolerance will rise only as the square of the basis set size. In the limit that an integral is most unlikely to be finite, it can easily be shown that the size of the Supermatrix file is three times that of the integral file, so that in this limit the two methods will assume almost identical CPU times, with the Supermatrix method being three times less efficient in its use of disk channels and storage - our conclusion is that the Supermatrix method has little to offer at least in its present form, for large systems of low symmetry.

We turn now to consider the effects of point group symmetry when a symmetry adapted basis set is used. The first benefit is that a large number of integrals or Supermatrix elements will be zero because of symmetry, both lists profiting equally. In the case of the Supermatrix a further saving is immediately realised because given that one is computing a totally symmetric Fock operator from a totally symmetric density matrix, the Supermatrix elements involving index pairs which are not totally symmetric may be neglected because the element is destined to be multiplied by a density matrix element which is zero by symmetry.

The situation for the integral driven algorithm is less clear because three sets of index pairs are involved, namely

ij kl

ik jl

il jk

Each integral may be classified into one of 2^3=8 types, according to whether each member of the set of index pairs does or does not give a finite contribution to the G matrix. Clearly one of these eight types covers the case where all contributions are zero, and in this case the integral need not be loaded to file. The remaining 7 types may be loaded into blocks, and a different algorithm used for each block, just as Billingsley proposed to handle the index coincidence problem mentioned above.

In the case of high symmetry (a point group with a large number of operators, eg D_{2h}) and where few integrals are zero by reason of distance the integral file will limit to 3 times the size of the Supermatrix file, with the processing times also being three times as great. In the case of the extensive system, the procedure of symmetry adapting the basis set will give little gain if linear combinations of orbitals separated by large distances need be taken. This is because the gain in sparsity through symmetry may be strongly counteracted by a loss of sparsity because the symmetry adapted functions will exhibit greater differential overlap than their non-symmetry adapted counterparts. Indeed, in

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch001

the case when all finite integrals or Supermatrix elements must be stored (because one is contemplating generating density matrix elements and Fock operators which are not totally symmetric) it is not difficult to find examples where the process of symmetry adaptation is counter productive, and where much shorter files may be produced by storing all symmetry distinct integrals over the original non-symmetry adapted but localised basis set, generating the symmetry related forms as need arises during Fock matrix construction. Before considering the question of a more complete vectorisation of the Fock matrix build we mention that the algorithms specified above are capable of constructing a Fock matrix over 100 basis functions using a completely non-sparse integral list or Supermatrix in CPU times of 19 or 6 seconds respectively.

We now turn to an apparently very powerful procedure for vectorisation of the G-matrix build in the case that the Supermatrix is non-sparse. Suppose we do not take advantage of the fact that

$$P_{ij,k\ell} = P_{k\ell,ij}$$

in the storage of the Supermatrix, and we define a modified Supermatrix, P', where

$$P'_{ij,k\ell} = P'_{k\ell,ij} = P_{ij,k\ell} \qquad (ij \neq k\ell)$$

$$P'_{ij,ij} = 2P_{ij,ij}$$

Each element of P' generates only one reference to G,

$$G_{ij} = P'_{ij,k\ell} * D'_{k\ell} + G_{ij}$$

We now aim to compute the G-matrix 64 elements at a time (this is the optimum for the CRAY, an appropriate vector length for the CDC CYBER 205 would be the size of the G-matrix). We now order the P' Supermatrix so that the first 64 elements are $P_{1-64,1}$, the second 64 are $P_{1-64,2}$ until all $k\ell$ indices are exhausted, when comes $P_{65-128,1}$, $P_{65-128,2}$ etc. This ordering permits us to evaluate 64 elements of the G-matrix at a time, where each $k\ell$ pair index gives rise to the equation

$$\begin{array}{ll} G_{ij} = G_{ij} + P'_{ij,k\ell} * D'_{k\ell} & \\ \text{Vector} = \text{Vector} + \text{Vector*Scalar} & \end{array} \qquad (ij=n,n+63)$$

The method may be regarded as an application of our 'Algorithm 2' for matrix multiply which is described below under the heading Integral transformation.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch001

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch001

The theoretical performance of this algorithm is 135 Mflops if appropriately coded in CAL or 50 Mflops in FORTRAN. The disadvantages of the algorithm are

(a) The number of elements of P' is double that of P even in the non-sparse case. If P' is stored on backing store, an I/O rate of 560 Mbyte/sec would be required, quite unthinkable.

The only use of the algorithm is for small cases where the whole of P' can be held in store. The reason for the calculation being expensive (and hence worth running on a powerful machine) might be that a potential energy surface investigation or geometry optimisation is being undertaken. If a store of 4 MWord is available, as on the CRAY-1 models S/4200, S/4300 and S/4400, the algorithm would be viable for up to approximately 60 basis functions. Such a calculation would take 0.05 sec per cycle of the SCF iteration. The incorporation of symmetry adaption is possible without degradation of the computation rate, and might raise the applicability of the procedure to 80 basis functions.

(b) The procedure would become less CPU efficient than the orthodox Supermatrix method when less than 5% of Supermatrix elements were finite, although the method is capable of utilising sparsity in the D-matrix as some compensation.

It is worth considering in detail why the performance of the CRAY is only approximately 8 Mflops when processing sparse integral lists or supermatrices. Suppose we arrange the Supermatrix file in blocks of 64, such that the same Fock operator element is not referenced by different Supermatrix elements within the block of 64. The equation

$$G_{ij} = G_{ij} + P_{ij,k\ell}\, D'_{k\ell}$$

may now be vectorised, as follows

```
   DIMENSION P(64),IJ(64),KL(64),DD(64),GG(64)

   DIMENSION G(?),D(?)

   DO 1 I=1,64

   DD(I)=D(KL(I))             (Gather)

   GG(I)=G(IJ(I))             (Gather)

   GG(I)=GG(I)+P(I)*DD(I)     (Floating Point Mult. and Add)

1  G(IJ(I))=GG(I)             (Scatter)
```

The gather and scatter phases in the above are not directly vectorisable on the CRAY, but may be quasi-vectorised by means of loop folding techniques in the Scalar hardware. The timing for performing the above plus the second Fock matrix reference generated by each Supermatrix element will be 29 Machine clocks per element, broken down as 3 for each Supermatrix element fetch and pair index unpack, 24 for the gather and scatters and 2 for the floating point operations. The net speed up when compared with our actual implementation (which does not assume that the Supermatrix has been ordered as decribed above) is about 25%, not very remarkable. If however, the CRAY had hardware vector gather and scatter capable of 1 element of gather or scatter per machine clock, each Supermatrix element require would 11 clocks to process, broken down as 3 for fetch and index unpack, 6 for scatter and gather and 2 for the floating point operations. The speed up over our current algorithm would be a factor of 3.5, giving a computation rate of 29 MFlops. Our conclusion is that the implementation of hardware to facilitate sparse matrix processing (of which gather/scatter is the most primitive and most essential) will prove in the future to be of high importance.

We mention here an alternative procedure of Ostlund (21), who considers that the use of asynchronous iteration techniques would show considerable advantage over the currently used lockstep synchronous procedure. One interpretation of Ostlund's remarks would be to generate a multi-dimensional search procedure over the Hartree-Fock energy surface. Each direction of search would require a Fock operator to be computed. Given a reasonable number of Fock operators to be constructed, a vectorisable procedure is immediately defined which would perform at approximately 50 Mflops, or 135 Mflops if the P' Supermatrix is used, as described later in the present work. It is difficult to believe that searches in more than 20 directions simultaneously will prove profitable, since the SCF process will often converge within less than 10 cycles, particularly if starting from a good trial set of vectors, such as is often available when studying a potential energy surface (the solution for the previous point is a good starting guess for the next). Perhaps Ostlund's strategy may find application in cases which are difficult to converge.

As a final variant the SCF procedure may be solved by a Newton Raphson technique, a very important component of which comprises a partial or complete 4-index transformation of integrals at each cycle. As we show below, the integral transformation procedure is highly vectorisable. We feel that such a technique will perhaps prove profitable in slowly convergent close shell cases or complicated open shell cases.

Self-Consistent-Field : The Open Shell SCF Case. The major point here is that the advantage of the Supermatrix methods is approximately halved over the closed shell case simply because one requires J and K matrices to be computed individually, necessitating the construction and use of 2 Supermatrices. Our present code based on the integral driven algorithm performs at 10.3 Mflops when processing cases described by Roothaan (22).

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch001

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch001

General Comments on SCF Procedures. In the above we have tacitly assumed that the Fock matrix build dominates an SCF calculation. However, for very extensive systems, the Fock build will become an N^2 process, whilst items such as matrix diagonalisation, multiplication etc. will remain N^3. The latter items should vectorise rather naturally using current algorithms, but might even then become the dominant component of the calculation. Clearly much work remains to be done on the analysis of sparse matrix manipulation. It behoves the theoretician to devise algorithms which naturally produce sparse matrices, and the computer designer to produce hardware giving powerful facilities for the processing of sparse matrices.

We conclude our survey of the SCF procedure by considering the timings obtained in a closed shell SCF calculation on the transition metal complex $Cr_2(O_2CH)_4$, di-chromium tetraformate, of D_4h molecular symmetry. The basis set comprised 162 contracted Gaussian functions, with a <5s4p2d> set on chromium, a <3s2p> set on carbon and oxygen and a <2s> set on hydrogen. Computing the 2-electron interaction matrix directly from the 2-electron integral list, comprising circa. 18,200,000 integrals (53,500 CRAY blocks of disk store), and neglecting the effect of symmetry adaption resulted in an SCF cycle time of 40.2 seconds, with 28 seconds used in Fock matrix construction and 12 seconds in matrix diagonalisation, multiplication etc. Clearly the n^4 phase of the calculation would be reduced significantly by Supermatrix processing (a factor of 2, due to allowing for possible effects of sparsity in the list) and symmetry adaption (a factor of 5), suggesting a possible 3 seconds for Fock matrix construction. Vectorisation and symmetry blocking of the n^3 dependent code might, not unrealistically, lead to a twenty-fold reduction i.e. 0.5 seconds, resulting in an overall cycle time of 3.5 seconds.

Integral Transformation. The CRAY version of our 4-index transformation program is based on the Yoshimine algorithm (23), which may be broken down into 4 phases.

(a) Sort the atomic integrals $[ij/k\ell]$ so that for a given ij all $k\ell$ are available. Much of this work may be vectorised e.g. fetching of integrals, unpacking of indices, computation of pair indices, and of the sort bin number for which the integral is destined. The only component which cannot be vectorised is the actual scatter of the integrals into their respective bins (again we note a requirement for the scatter order). Because our program is capable (in the interests of saving main store or disk space, or both) of degrading to a multi-pass sort, a decision has to be made as to whether an integral does or does not contribute to a given pass. This decision making cannot be fully vectorised, although the CRAY Vector Mask feature is of very considerable assistance. Our current version is capable of sorting $1.3 * 10^5$ integrals/second, representing a speed up over the CDC 7600 version of about 4. Whilst this rate of processing is not particularly impressive, this phase of the procedure is not usually very expensive and we regard the processing rate as adequate.

(b) The transformation of the integrals to semi-transformed form, $[ij/k\ell] \rightarrow [ij/uv]$ comprises the second stage of the process, and consists of nothing more than a sequence of N*(N+1)/2 (N is the number of basis functions) matrix multiplications of the form

$$B = Q^{+} A Q$$

where Q is the molecular orbital coefficient array, A is a symmetric matrix of untransformed integrals of common ij index pair, and B is a symmetric matrix of semi-transformed integrals, again of common ij index pair. The matrix A will be sparse given that the integrals are relevant to a large molecule. Q may be regarded as essentially not sparse (any sparsity is normally caused by symmetry, which can effectively be allowed for by straightforward matrix blocking techniques upon which we do not dwell here). Consider the first phase of the matrix multiplication, namely

$$X = A Q$$

The first point is that A is normally read from disk in 'triangular' form, and must be transcribed to full square form, a process which is thoroughly vectorisable. The amount of store allowed for each column of the square form of A would normally be N words. However in the case that N is divisable by 8, we allocate N+1 words, which prevents subsequent store conflicts. A similar store allocation algorithm is applied to Q, whose dimension is in general N rows by M columns (M<N, where M is the number of active orbitals).

An excellent algorithm for performing such a matrix multiplication is available in a CRAY supplied BLAS routine - unfortunately this routine does not allow for sparsity in either A or Q. Accordingly we have constructed an algorithm which does allow for such sparsity, and which proceeds as follows. The matrix X is constructed in blocks of 64 * 64, except for the lower and right hand border where smaller sub-blocks arise. The partitioning is shown for an X matrix whose number of rows is >128 and <192, and whose number of columns is >64 and <128 in figure 4. The shaded area in Figure 4 is computed via the multiplication of the shaded blocks of A and Q as shown in Figure 5. One may now chose between two algorithms for the multiplication of the sub-blocks of A and Q to produce a sub-block of X, the dimensions of the latter being NROW*NCOL, where both NROW and NCOL are <64.

Algorithm 1

The algorithm is based on the calculation of a row of X in vector sequences. The FORTRAN code in Figure 6 gives a picture of the procedure, but must be coded in CAL, with all vectors labeled Vx being directly replaced by V registers - their dimension is always

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch001

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch001

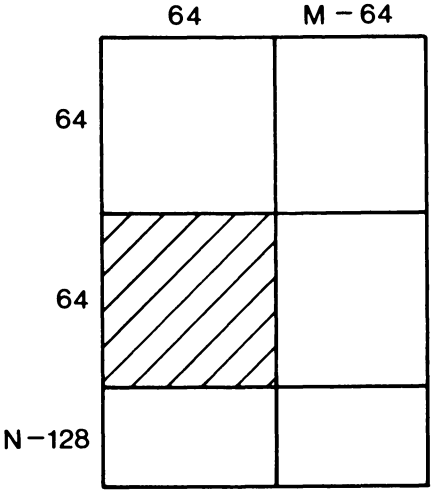


Figure 4. Partitioning of x *into 64 * 64 subblocks.*

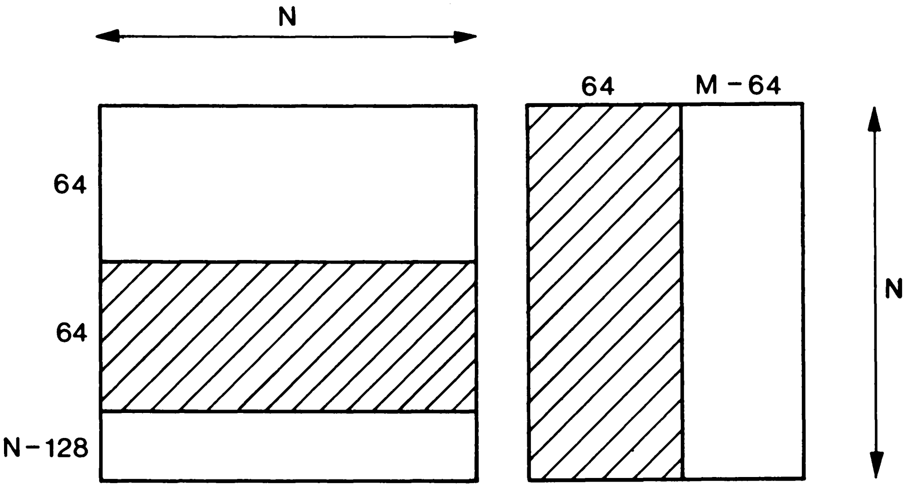


Figure 5. Partitioning of A *and* Q.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch001

```
      DO 1 I=1,NROW
      IPARIT=1
      DO 2 J=1,NCOL
 2    V1(J)=0.0
      DO 3 K=1,N
      SCALAR=A(I,K)
      IF(SCALAR)88,3,88
88    IF(IPARIT)66,77,77
77    DO 4 J=1,NCOL
 4    V2(J)=V1(J)+Q(K,J)*SCALAR
      GO TO 55
66    DO 5 J=1,NCOL
 5    V1(J)=V2(J)+Q(K,J)*SCALAR
55    IPARIT=-IPARIT
  3   CONTINUE
      IF(PARIT)33,44,44
44    DO 6 J=1,NCOL
 6    X(I,J)=V1(J)
      GO TO 1
33    DO 7 J=1,NCOL
 7    X(I,J)=V2(J)
 1    CONTINUE
```

Figure 6. FORTRAN representation of subblock multiplication code—Algorithm 1.

In Supercomputers in Chemistry; Lykos, P., el al.;
ACS Symposium Series; American Chemical Society: Washington, DC, 1981.

<64. Each row of the sub-block is of course of dimension NCOL. In the interests of simplification, indexing of all sub-blocks is assumed to commence at element (1,1). The sparsity of A is utilised through the statement

```
IF(SCALAR)88,3,88
```

and can actually be better implemented if an index array is maintained which tells one where the next finite element of A is. The curious and apparently useless variable IPARIT is to ensure that coding such as

```
V1(J) = V1(J) + Q(K,J) * SCALAR
```

does not arise, because the use of a vector register as both operand and result causes the CRAY to enter a 'recursive' mode which would not give the result required. In order to minimize the 'effective' startup time of the chained vector sequences of DO 4 and DO 5 it is necessary to apply a two-way fold on DO 3, a complication not shown in Figure 6. ('Effective' startup time is defined as the time, in machine clocks, between the issue of successive chained vector sequences less the vector length). In summary, the algorithm produces vector orders of length NCOL, takes the sparsity of A into account, and produces the result X in rows. In CAL, the fetch of Q and the Floating point Multiplication and Addition all chain together to give an overall computation rate of circa. 135 Mflops.

Algorithm 2

The procedure is to produce the columns of the result in vector sequences. The algorithm comprises a simple modification of that detailed for 'Algorithm 1'. Vector orders of NROW in length are produced, and the sparsity of the Q matrix can be taken advantage of.

The program currently chooses that algorithm which produces the longer vector length, but if both lengths are equal, then 'Algorithm 1' is favoured. A refinement of this procedure is as follows: Let L_1 and L_2 denote the vector lengths respectively, P_A and P_Q denote the probability of finding a finite element in A or Q respectively and S denotes the effective startup time for the chained vector sequence. We then calculate

$$M_1 = (S + L_1) * P_A/L_1$$

and

$$M_2 = (S + L_2) * P_Q/L_2$$

If $M_1 < M_2$ choose Algorithm 1.

The final stage in the multiplication process is

$$B = Q^+X$$

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch001

This proceeds much along the lines indicated for $X = AQ$ with the added complication that B is a symmetric matrix, so that it is necessary to impose indicial restrictions. These complications cause no problem, and this phase also proceeds at the 135 Mflops rate.

The third and fourth stages of the 4-index transformation consists of a sort phase and a matrix multiply phase carried out much as on the lines indicated above.

Electron Correlation. Some categories of correlated wavefunction calculation (e.g. SCEP (24,25), APSG and certain restricted forms of MCSCF (26)) require the simultaneous construction of large numbers of Fock operators. These potentially expensive calculations map onto the CRAY vector processor rather naturally, and should produce 50 Mflop performance figures without difficulty, given that the number of Fock operators to be constructed approaches or exceeds 64. For successful application on the CDC CYBER 205, the number of Fock operators should be somewhat higher (because of the longer startup times of vector orders on that machine). The application (27) may be regarded as a natural occurrence of the Ostlund Algorithm, and will obviously require large amounts of fast memory to be available. In particular, all Fock operators and their corresponding density matrices must be held in fast store (actually it suffices to hold batches of 64 operators and density matrices on the CRAY-1).

However a method based on the 'single-reference' Supermatrix P' defined above may devised which performs at 135 Mflops (if coded in CAL). The method requires only that all the density matrices be held in store, simultaneously, halving the store requirement of the 'natural' algorithm. The P' Supermatrix should be ordered so that for a given ij pair index all kℓ pair indices are available. We are then able to compute G_{ij} through an inner product

$$G_{ij} = \sum_{k\ell} P'_{ij,k\ell} D'_{k\ell}$$

Vectorisation is achieved by computing all the G-matrices simultaneously.

$$\underset{\text{Vector}}{G^{(n)}_{ij}} = \underset{\text{Vector}}{G^{(n)}_{ij}} + \underset{\text{Scalar*Vector}}{P'_{ij,k\ell} D'^{(n)}_{k\ell}} \qquad (n = 1,m)$$

where we require m G-matrices to be built. The above vector order is executed as many times as there are finite $P_{ij,k\ell}$ elements. The method may be regarded as an application of our 'Algorithm 1' for performing matrix multiply described previously. Sparsity in the P' Supermatrix may be easily accounted for, as may symmetry adapadaption, without degradation of the compute rate.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch001

The only disadvantage of the procedure is that the disk storage requirement for P' is twice that of P. It will be noted that a particular element of m G-matrices will be produced simultaneously, and these may be output to disk as they are computed - the total store requirement for the G-matrices is thus m words. The I/O rate required by the algorithm is 560/m Mbytes/sec, not unreasonable if m is of the order of 64 or more.

Direct CI procedures of a quite general kind (28) may be formulated as a matter of constructing large numbers of J and K operators, usually from a transformed integral list, and hence directly in the m.o. basis. Algorithms as described above are also applicable to this case, and should proceed at 135 Mflops given a reasonably large external space.

In a report (29) entitled "New Methods in Computational Quantum Chemistry and their Application on Modern Super-Computers", Shavitt states that by use of the Graphical Unitary Group Approach (G.U.G.A.) (30-32) "the total computational effort required for formula determination in multi-reference direct CI calculations becomes negligible in comparison with the remaining rate determining steps, the integral transformation and the eigenvector iteration procedure". We have demonstrated in the preceding subsection that the integral transformation can be driven at 135 Mflops on a CRAY-1 computer. The diagonalisation which has to be performed in a configuration iteration calculation can be avoided by using a non-iterative approach to the correlation problem such as the many-body perturbation theory(33-34). This approach also has the theoretical advantage that, unlike incomplete configuration interaction, it is size-consistent. Algorithms written to perform many-body perturbation theory calculations can be adapted to take full advantage of the vector-processing features of the CRAY-1. This will be demonstrated in the next section where we discuss the evaluation of the triple-excitation component of the correlation energy using diagrammatic perturbation theory.

Recent Progress

In this section, we propose to illustrate how the availability of the CRAY has assisted progress in the area of molecular electronic structure. We shall concentrate on two recent advances, namely, the evaluation of the components of the correlation energy which may be associated with higher order excitations, in particular triple-excitations with respect to a single-determinantal, Hartree-Fock reference function, and the construction of the large basis sets which are ultimately going to be necessary to perform calculations of chemical accuracy, that is one millihartree.

(i) Evaluation of the components of the correlation energy associated with triply-substituted configurations.

In the vast majority of calculations of the electron correlation energy of atoms and small molecules it is usually only possible to include configurations which are singly- and doubly-excited

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch001

with respect to some single-determinantal, or even multi-determinantal, reference function. The components of the correlation energy associated with higher-order excitations are usually assumed, heuristically, to be negligible. One of the most powerful techniques currently available for describing correlation effects in atoms and molecules is the diagrammatic many-body perturbation theory(33,34). Using this approach, within the algebraic approximation, some of the most accurate correlation energies calculated to date have been obtained. The method also provides a systematic procedure for including the effects of higher-order excitations. A full set of diagrams, using the nomenclature of Brandow, for the correlation energy of a closed-shell system is given in Figure 7 through fourth order in the energy(35). In second-order and third-order, corresponding to the first four diagrams in Figure 7, only doubly-excited intermediate states arise. In fourth-order of the perturbation expansion, single, double, triple and quadruple excitations arise in intermediate states.

The fourth-order linked diagram quadruple-excitation component of the correlation energy can be calculated efficiently(14,36), since this arises from disconnected wavefunction diagrams. (It should be noted that there is also a fourth-order unlinked diagram quadruple-excitation energy which is responsible for the dominant part of the so-called size-consistency error in double excitation configuration interaction calculations but which is explicitly cancelled in many-body perturbation theory by unlinked diagrams involving only double excitations). The situation is different for the fourth-order triple-excitation component of the correlation energy. This component arises from wavefunction diagrams which are connected. The algorithm devised for evaluating this component of the correlation energy(37) depends on the seventh power of the number of basis functions c.f. the algorithms devised for evaluating the fourth-order quadruple-excitation energy described in reference (14). In evaluating the triple-excitation, fourth order energy it is possible to improve the algorithm by (i) using a spin-free formalism (ii) recognising certain permutational symmetry properties of intermediates which arise in the computation (iii) (the point which is of interest to the present discussion) by writing a FORTRAN program to take advantage of the vector processing repertoire of the CRAY-1 computer. It has thus been possible to calculate the triple-substitution component of the correlation energy for a number of closed shell atoms and molecules and to investigate the validity of heuristically neglecting these terms as is usually done in molecular calculations (38-41).

Full details of the algorithm have been given elsewhere (37). Here, we indicate the features of the calculation which take advantage of vector processing facilities. The inner most DO loop of the program forms intermediates of the type

$$T_1 = \sum_{D} X(IJDB)\ Y(DKAC)$$

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch001

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch001

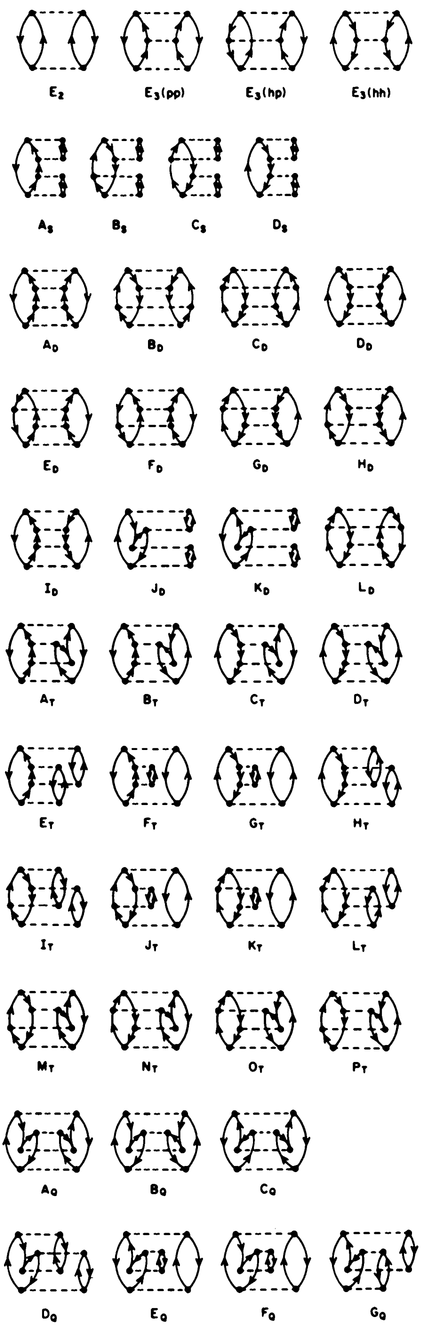


Figure 7. Diagrammatic expansion of the correlation energy through fourth order.

$$T_2 = \sum_{D} X(IJDB)\ Y(DKCA)$$

$$T_3 = \sum_{D} X(IJBD)\ Y(DKAC)$$

$$T_4 = \sum_{D} X(IJBD)\ Y(DKCA)$$

where X(IJDB) and Y(DKAC) are 'two-electron integral-like' quantities. The DO loop for constructing these intermediates is nested inside six other loops depending on the occupied indices I J K and the virtual orbital indices A B C, and it is thus clearly important that the inner loop be very efficient. The following FORTRAN code was written to evaluate the products arising in the intermediates T_1, T_2, T_3 and T_4

```
  DO 1 ID=1,N
  DD(ID)=D1(ID)*D2(ID)
  DE(ID)=D1(ID)*E2(ID)
  ED(ID)=E1(ID)*D2(ID)
  EE(ID)=E1(ID)*E2(ID)
1 CONTINUE
```

where D1 contains values of X(IJDB), D2 contains Y(DKAC), E1 contains X(IJBD) and E2 contains Y(DKCA). In Table V the times required to execute this DO loop on a CRAY-1 computer and an IBM 360/195 computer are given as a function of N. As would be expected, the timing of the IBM 360/195 computer varies approximately linearly with the number of basis functions, N. For values of N up to 64, the length of the vector register, the timings for the CRAY-1 computer vary approximately as the square root of N. In Table VI the computer times required on the CRAY-1 to evaluate the triple-excitation component of the fourth-order energy are given for a number of molecules. In all calculations the valence correlation effects were investigated. For comparative purposes, we also give the time required to calculate the correlation energy associated with double excitations, t_D, (through third-order) and that associated with quadruple-excitations, t_Q, (fourth-order). Although the present program for evaluating the triple-excitation energy component could undoubtedly be improved, it is clear that the calculation of this component of the correlation energy is relatively expensive and is going to require the use of the new generation of supercomputers to perform routine calculations with basis sets of reasonable size.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch001

Table V: CPU times on the CRAY-1 and the IBM 360/195 computer required to execute the inner loop in evaluation of triple excitation energy component.

N [a)]	Time [b)]	
	CRAY-1 [c)]	IBM 360/195 [d)]
16	3.6	41.8
32	5.2	82.9
48	6.8	124.4
64	8.4	165.6

[a)] DO-loop range defined in text
[b)] CPU time in microseconds
[c)] Using vector order repertoire
[d)] FORTRAN H extended plus compiler with OPT=2

Table VI: CPU times on the CRAY-1 computer required to evaluate the double-excitation, t_D, triple-excitation, t_T, and quadruple-excitation, t_Q, components of the correlation energy.

Molecule	N_{basis} (a)	t_D (b)	t_T (b)	t_Q (b)
FH	23	3.14	41.52	1.80
CℓH	23	2.92	41.53	1.76
OH_2	29	15.51	107.71	5.59
N_2	34	14.93	352.49	9.92
CO	34	18.32	339.00	10.30
SiS	34	18.55	352.58	10.30

(a) active orbitals
(b) seconds

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch001

Calculations have, unfortunately shown that the triple-excitation component of the correlation energy is chemically significant, that is greater than one millihartree. For example, for the water molecule, using the basis set of thirty-nine Slater functions given by Rosenberg and Shavitt (42), a fourth-order triple-excitation energy of -7.9 millihartree was obtained (39). The triple-excitation energy for a number of small atoms and molecules is shown in Table VII (48). It can be seen from the Table that this component of the correlation energy is quite large and particularly so for multiply-bonded systems, such as the nitrogen molecule. Calculations on portions of the potential energy curves of some multiply bonded diatoms have shown that the triple-excitation component of the correlation energy varies quite markedly with nuclear geometry. Furthermore, the results shown in Table VIII for the water molecule using different basis sets show that this energy has a somewhat stronger dependence on the quality of the basis set than other components of the correlation energy (41).

(ii) Basis sets for accurate molecular calculations.

Fundamental to almost all applications of quantum mechanics to molecules is the use of a finite basis set. Such an approach leads to computational problems which are well suited to vectorisation. For example, by using a basis set the integro-differential Hartree-Fock equations become a set of algebraic equations for the expansion coefficients - a set of matrix equations. The absolute accuracy of molecular electronic structure calculations is ultimately determined by the quality of the basis set employed. No amount of configuration interaction will compensate for a poor choice of basis set.

The supercomputers open up the possibility of using much larger basis sets in routine calculations and thus achieving greater precision than is currently attainable.

A recently developed concept which will enable large basis sets to be used in molecular calculations is the Universal Basis Set (43-47). To recover a significant fraction of the correlation energy it is ultimately necessary to use a moderately large basis set. Such a large basis set has a considerable degree of flexibility and can therefore be transferred from system to system without regard to the particular atoms involved and with little loss in accuracy. This approach has been demonstrated using a universal even-tempered basis set in which the orbital exponents ζ_k are defined by a geometric series

$$\zeta_{k\ell} = \alpha_\ell \beta_\ell^{k-1} \qquad k = 1,2,\ldots,N_\ell$$

where ℓ denotes the symmetry type of the basis functions. Values of α and β have been determined which are capable of giving an accurate description of first-row and second-row atoms within the Hartree-Fock model. Using the following parameters:

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch001

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch001

Table VII Fourth-order linked-diagram triple-excitation and quadruple excitation components of the electron correlation energy in a number of atoms and small molecules [a)]

System	E_{SCF}	$E_{[2/1]}$	E_{4T}	E_{4Q}
Ne	-128.54045	-210.034	-2.427	0.415
Ar	-526.80194	-161.846	-1.159	2.068
BH	-25.12890	-86.731	-0.770	1.290
FH	-100.06007	-222.427	-4.341	1.071
$A\ell H$	-242.45553	-72.745	-0.521	1.116
$C\ell H$	-460.09456	-170.724	-2.479	2.789
BeH_2	-15.77024	-68.623	-0.285	0.699
OH_2	-76.05558	-223.457	-5.539	2.018
MgH_2	-200.72028	-63.105	-0.084	0.586
SH_2	-398.70077	-173.182	-3.383	3.232
BH_3	-26.39834	-121.471	-1.327	1.562
CH_3^+	-39.24533	-134.462	-1.588	1.691
$NH_3(D_{3h})$	-56.20957	-208.747	-4.677	2.450
$NH_3(C_{3v})$	-56.21635	-210.245	-5.040	2.591
$OH_3^+(D_{3h})$	-76.33276	-221.592	-4.573	2.013
$OH_3^+(C_{3v})$	-76.33475	-223.315	-4.778	2.128
$A\ell H_3$	-243.63770	-105.091	-0.705	1.240
$PH_3(D_{3h})$	-342.41887	-168.516	-3.145	3.173
$PH_3(C_{3v})$	-342.47717	-167.680	-3.064	2.998
N_2	-108.97684	-317.938	-17.602	6.284
CO	-112.77551	-295.510	-15.596	4.536
BF	-124.15642	-254.719	-8.392	2.088
CS	-435.33679	-260.619	-18.534	7.360
SiO	-363.82790	-275.312	-16.700	4.078
SiS	-686.48488	-225.190	-11.719	6.436

[a)] The self-consistent-field energies are given in hartree; the triple-excitation energy, E_{4T}, the quadruple-excitation energy, E_{4Q}, and the [2/1] Padé approximant, $E_{[2/1]}$, are given in millihartree.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch001

Table VIII: Components of the correlation energy of the ground state of the water molecule using Gaussian basis sets of different quality†

Basis set		E_2	E_{4T}	E_{4Q}	$E_2\Delta_{11}$	$\lambda^3 E_2\Delta_{11}$	R(μ)
Primitive	Contracted						
[9s5p/4s]	(3s2p/2s) [a]	-111.836	-1.286	-0.489	-4.222	-4.055	0.27
	(3s2p/2s) [b]	-123.976	-1.274	-0.778	-4.963	-4.561	0.22
	(4s2p/2s)	-125.082	-1.285	-0.814	-5.043	-4.612	0.22
	(4s3p/2s)	-139.850	-3.218	-0.732	-5.578	-4.871	0.50
	(5s3p/2s)	-140.049	-3.232	-0.734	-5.589	-4.876	0.51
	(5s3p/3s)	-141.182	-3.271	-0.684	-5.678	-4.954	0.51
	(9s5p/4s)	-145.888	-3.673	-0.627	-5.873	-5.094	0.57
[10s6p/5s]	(5s3p/3s)	-144.324	-3.379	-0.740	-5.978	-5.160	0.50
	(5s4p/3s)	-147.882	-3.953	-0.664	-6.138	-5.268	0.58
[10s6p/5s]	(5s4p/3s)						
	+ 1d(O)	-211.140	-4.860	+1.425	-10.691	-9.063	0.52
	+ 1p(H)	-170.604	-5.320	+0.157	-7.985	-6.848	0.68
	+ 1d(O) 1p(H)	-226.080	-6.148	+2.158	-12.451	-11.025	0.60
	+ 2d(O) 2p(H)	-244.691	-7.934	+3.059	-14.578	-12.692	0.69

† in millihartree

(a) contraction coefficients from T.H. Dunning Jr., J. Chem. Phys. 53 (1970) 2823.

(b) contraction coefficients from T.H. Dunning Jr., and P.J. Hay, in "Methods of Electronic Structure Theory", edited by H.F.Schaefer III, Plenum, New York 1977.

In Supercomputers in Chemistry; Lykos, P., el al.;
ACS Symposium Series; American Chemical Society: Washington, DC, 1981.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch001

1s :	$\alpha = 0.5$	$\beta = 1.55$	$N = 9$
2p :	$\alpha = 1.0$	$\beta = 1.60$	$N = 6$
3d :	$\alpha = 1.5$	$\beta = 1.65$	$N = 3$

calculations using exponential basis functions have been performed on the nitrogen, carbon monoxide and boron fluoride molecules (47) including correlation effects by taking the many-body perturbation expansion through third-order. Approximately 80% of the empirical correlation energy was recovered at the equilibrium nuclear geometry of these three species. The calculations were performed on the IBM 370/165 computer at the Daresbury Laboratory and required a considerable amount of CPU time. Clearly, such calculation could be made within a few minutes on a vector processor and would, therefore, become routine.

In 1963, Schwartz (48) emphasised the need to have a systematic scheme for extending basis sets. Ruedenberg and his co-workers (49-50) have recently reiterated this viewpoint. They developed a technique for systematically extending basis sets of the even-tempered type. In this scheme α and β are regarded as functions of the number of basis functions, N. This is necessary if the basis set is to be capable, in principle, of approaching a complete set. Specifically, Ruedenberg et al (49-50) proposed the empirical forms

$$\ln\ln\beta = b\ln N + b'$$

and

$$\ln\alpha = a\ln(\beta - 1) + a'$$

where a, a', b and b' are constants. Ruedenberg et al investigated this approach for a number of atoms within the Hartree-Fock model. The calculated energies were found to behave smoothly as a function of the number of basis functions. It was shown that the Hartree extrapolation procedure can provide an empirical upper bound to the basis set limit. Empirical lower bounds to the basis set limit can also be obtained. It is important to employ such a systematic approach in accurate studies so that it is possible to assess the convergence properties with respect to the basis set. Furthermore, as the basis set is extended, linear dependence amongst the basis functions increases. However, if a systematic approach is adopted extrapolation procedures can be employed with confidence to obtain the basis set limit. This procedure has been shown to be useful in calculations in which electron correlation effects are accounted for (51).

The use of a systematic sequence of basis sets can, of course, be usefully combined with the use of a universal basis set(52). In Figure 8, we display the results of Hartree-Fock calculations on the radial beryllium-like ions Li^-, B^+, C^{2+}, N^{3+}, O^{4+}, F^{5+}, Ne^{6+}, using the basis set given by Schmidt and Ruedenberg(56) specifiically for the beryllium atom. It can be seen that for the positive ions the basis sets give a uniform convergence rate. In

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch001

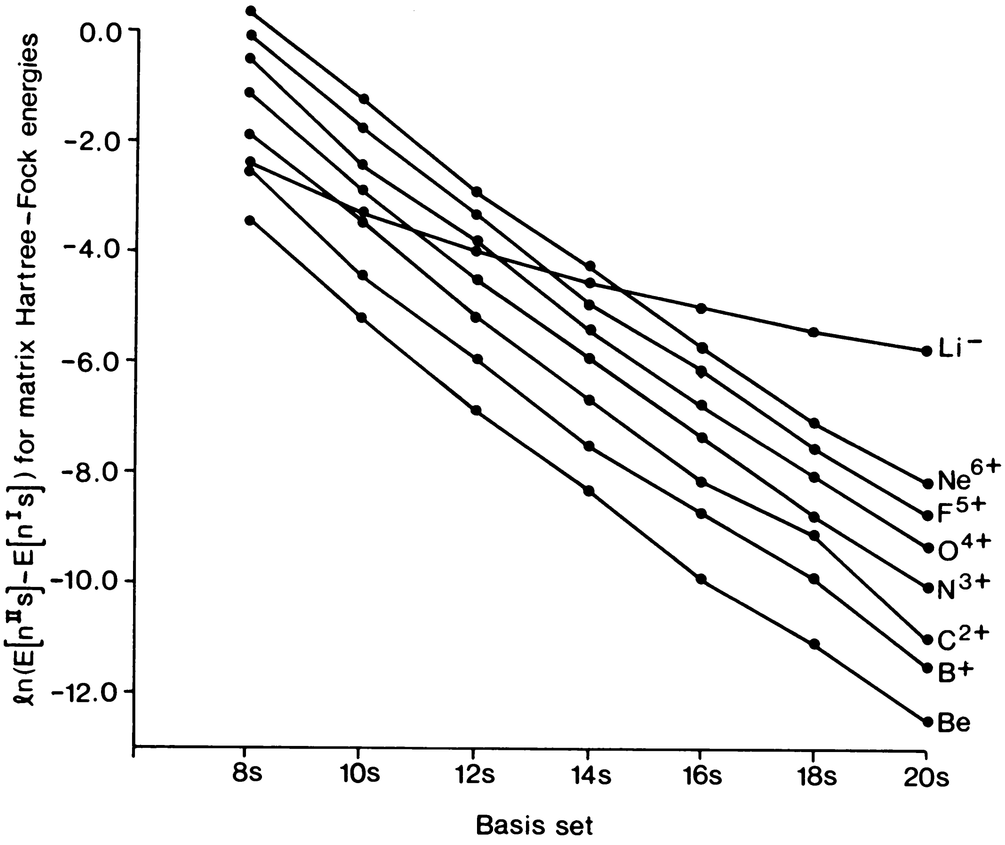


*Figure 8. Plot of ln(*E[n″s] − E[n′s]*) against basis set size for beryllium-like ions.*

Figure 9, we show the calculated correlation energies for these systems. Again with the exception of Li^-, uniform convergence is observed.

It is envisaged that by employing a universal systematic sequence of even-tempered basis sets on molecules such as N_2, CO and BF, we can employ extrapolation procedures with some confidence and investigate basis set limits. This will enable quantum chemical calculations to approach a chemical accuracy of 1 millihartree for small molecules.

Concluding Remarks

It is clear that the new generation of computers - supercomputers when used in conjunction with theoretical developments such as the linked diagram theory or the unitary group theory, is going to have a significant impact on the accuracy attainable in quantum chemistry and on the size of problem which may be treated.

The implementation of various standard quantum chemistry packages on the CRAY-1 computer at Daresbury was relatively straightforward, but significant recoding was necessary to take advantage of the vector processing features of this machine. The following points emerged from this recoding.

(1) In the CRAY-1 a Gaussian integrals program may be driven at 35 Mflops, a large closed shell SCF with a sparse list of integrals or Supermatrix at 10 Mflops, while a smaller SCF with a non-sparse Supermatrix may be driven at 135 Mflops.

Procedures such as SCEP may be performed at 135 Mflops, whilst a four-index transformation of the 2-electron integrals will also proceed at 135 Mflops in large cases, indicating the CRAY to be between 5 and 25 CDC 7600 in power when given appropriate code.

(2) To achieve the above performance figures some programs require large amounts of main memory to be available.

(3) Some applications (e.g. Fock matrix construction) will demand very high transfer rates from backing store if they are not to become severely I/O bound.

(4) A requirement for scatter/gather hardware and other aids to sparse matrix handling has been indicated, specifically in the case of SCF calculations and in the sort phase of the 4-index transformation. The requirement will, we suspect, be even more severe in the case of Configuration Interaction calculations.

(5) The CRAY FORTRAN compiler, CFT, sometimes produces nearly optimal code (as for example in Gaussian integrals evaluation); sometimes an increase in speed by a factor of 3 is observed on proceeding to the assembler, CAL. This factor of 3 is often present in rather simple applications such as matrix multiply.

In our work we have been unable to obtain more than 50 Mflops computation rate from the CFT compiler, and this reflects a lack of richness in our codes. The evaluation of a high order polynomial

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch001

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch001

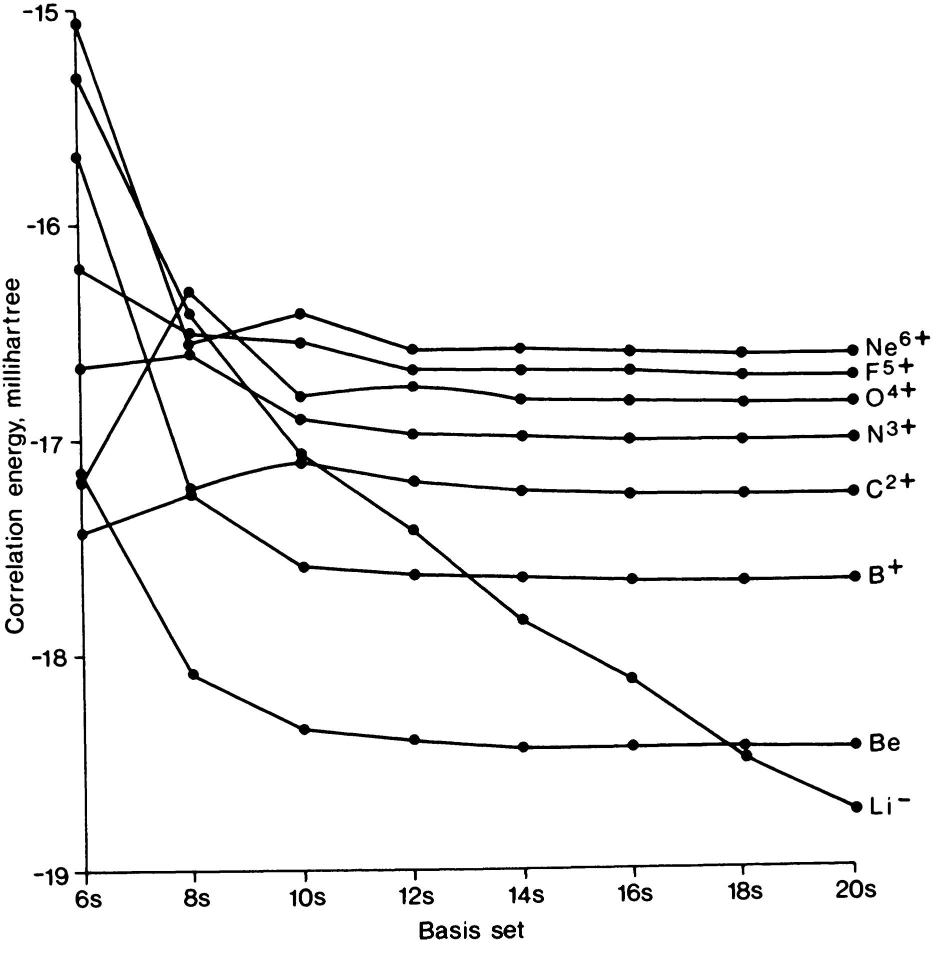


Figure 9. Plot of the correlation energy, given by the [2/1] Padé approximant to the third-order energy, against basis set size for beryllium-like ions.

$$R_i = \Big(\Big(((X_i+a)*X_i+b)*X_i+c)*X_i+d\Big) \;.......(a,b,c \text{ are constants}\Big)$$

is an example of a rich code (for each fetch of X and store of R a great deal of Floating point arithmetic occurs), and here the CFT compiler would produce code of 135 Mflops given a polynomial of sufficiently high order.

The new machines have enabled new areas of research to be investigated. We have described some of our work in the contribution of higher-order excitation energies to correlation energies using diagrammatic many-body perturbation theory and our work aimed at the development of the large basis sets which can now be employed in molecular calculations together with schemes for systematically extending them.

Acknowledgements

We wish to express our gratitude to Dr. V.R. Saunders for his invaluable assistance, without which this work would not have been possible.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch001

Abstract

The impact which vector processing computers are having on ab initio quantum chemical calculations will be considered, giving special emphasis to the experiences of United Kingdom scientists. Recent work using the CRAY-1 computer at the S.E.R.C. Daresbury Laboratory will be described. The CRAY-1 computer is at the centre of a network providing large scale computational facilities for universities in the U.K. Experience of running and subsequent vectorisation of certain standard ab initio packages will be discussed. Integral evaluation, self-consistent-field and orbital transformation phases of quantum chemical studies will be considered together with both the many-body perturbation theory and configuration mixing approaches to the electron correlation problem in molecules. Not only are improvements to the traditional algorithms which are made possible by the use of vector processors discussed, but also new areas of research which such computers open up are outlined. Future developments are considered briefly.

Literature Cited

1. Johnson, P.M., *Computer Design*, (1978) **17**, 89.
2. Hockney, R.W., *Contemp. Phys*. (1979) **20**, 149
3. Saunders, V.R., and Guest, M.F., (1976), ATMOL3 Reference Manual, Atlas Computing Divison, Rutherford Laboratory, Chilton, Didcot, Oxon OX11 0QY.
4. Dupuis, M., Rys, J. and King, H.F., *J. Chem. Phys*. (1976) **65**, 111.
5. Moskowitz, J.W. and Snyder, L.C. "Modern Theor. Chem.", Vol.3, Schaefer, H.F. III, Ed., Plenum Press, New York, 1976.
6. Almlof, J. USIP Report 74-29, University of Stockholm, 1974.
7. Guest, M.F. and Rodwell, W.R., (1977) SPLICE Reference Manual, Atlas Computing Division, Rutherford Laboratory, Chilton, Didcot, Oxon OX11 0QY
8. Dierksen, G.H.F., and Kraemer, W. MUNICH Reference Manual.
9. Roos, B., *Chem. Phys. Letts*. (1972) **15**, 153.
10. Silver, D.M., *Comput. Phys. Comm*. (1978) **14**, 71.
11. Silver, D.M., *Comput. Phys. Comm*. (1978) **14**, 81.
12. Wilson, S., *Comput. Phys. Comm*. (1978) **14**, 91.
13. Wilson, S., 1978, Daresbury Laboratory Technical Memorandum DL/SRF/TM 13.
14. Wilson, S., and Silver, D.M., *Comp. Phys. Comm*. (1979) **17**, 47.
15. Wilson, S., 1978, Daresbury Laboratory Technical Memorandum DL/SRF/TM 15.
16. For further details, see Guest, M.F. and Overill, R.E., Chem. Phys. Letts. (1980) in press.
17. Saunders, V.R., 1980, in "Proceedings of the Daresbury Study Weekend", November 1979, edited by M.F. Guest and S. Wilson, SERC Daresbury Laboratory.
18. Pople, J.A., and Hehre, W.J., *J. Comp. Phys*. (1978) **27**, 161. Saunders, V.R., 1975, in "Computational Techniques in Quantum Chemistry and Molecular Physics", edited by G.H.F. Diercksen, B.T. Sutcliffe and A. Veillard, REIDEL (Dordrecht), p347.
19. Duke, A.J., *Chem. Phys. Letts*., (1972) **13**, 76.
20. Billingsley, F.P., *Int. J. Quant. Chem*. (1972) **6**, 617.
21. Ostlund, N.S., *Int. J. Quant. Chem*. (1979) **S13**, 15.
22. Roothaan, C.C.J., *Rev. Mod. Phys*. (1960) **32**, 179.
23. Yoshimine, M., IBM Technical Report, RJ-555, San Jose, USA, 1969.
24. Meyer, W., *J. Chem. Phys*. (1975) 64, 2901.
25. Ahlrichs, R., *Comp. Phys. Commun*. (1979) **17**, 31.
26. Saunders, V.R. and Guest, M.F., in "Quantum Chemistry, the state of the Art", Saunders, V.R. and Brown, J. Eds., (S.E.R.C. 1975) p.119.
27. Zirz, C., and Ahlrichs, R., Proc. Daresbury Study Weekend on Electron Correlation, DL/SCI/R 14, November 1979, Guest, M.F. and Wilson, S. Eds.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch001

In Supercomputers in Chemistry; Lykos, P., el al.;
ACS Symposium Series; American Chemical Society: Washington, DC, 1981.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch001

28. Siegbahn, P.E.M., 1979, J. Chem. Phys. 70, 5391.
29. Shavitt, I., 1979, "New Methods in Computational Quantum Chemistry and their Application on Modern Supercomputers", Battelle Columbus Laboratories.
30. Paldus, J., 1974, J. Chem. Phys. 61, 5321.
31. Paldus, J., 1980, in "Proceedings of the Daresbury Study Weekend", November 1979, edited by M.F. Guest and S. Wilson, SRC Daresbury Laboratory.
32. Shavitt, I., 1980, in "Proceedings of the Daresbury Study Weekend", November 1979, edited by M.F. Guest and S. Wilson, SRC Daresbury Laboratory.
33. Wilson, S., and Silver, D.M., Phys. Rev. (1976) A14, 1949.
34. Wilson, S., 1980, in "Proceedings of the Daresbury Study Weekend", November 1979, edited by M.F. Guest and S. Wilson, SRC Daresbury Laboratory.
35. Wilson, S., and Silver, D.M., Int. J. Quant. Chem. (1979) 15, 583.
36. Wilson S., and Silver, D.M., Molec. Phys. (1978) 35, 1539.
37. Wilson, S., and Saunders, V.R., Comput. Phys. Commun. (1980) (in press)
38. Wilson, S., and Saunders, V.R. J., Phys. B: Atom. Molec. Phys. (1979) 12, L403; ibid, (1980), 13, 1505.
39. Wilson, S., J. Phys. B: Atom. Molec. Phys. (1979) 12, L657; ibid, (1980) 13, 1505.
40. Guest, M.F., and Wilson, S., Chem. Phys. Lett. (1980) 72, 49.
41. Wilson, S., and Guest, M.F., Chem. Phys. Lett. (1980) in press).
42. Rosenberg, B.J., and Shavitt, I., J. Chem. Phys. (1980) 63, 2163.
43. Silver, D.M., and Wilson, S., J. Chem. Phys. (1978) 69, 3787.
44. Silver, D.M., and Nieuwpoort, W.C., Chem. Phys. Lett. (1978) 57, 421.
45. Silver, D.M., Wilson, S. and Nieuwpoort, W.C., Int. J. Quant. Chem. (1978) 14, 635.
46. Wilson, S., and Silver, D.M., Chem. Phys. Lett. (1979) 63, 367.
47. Wilson, S., and Silver, D.M., J. Chem. Phys. (1980) 72, 2159.
48. Schwartz, C., Methods in Computational Physics (1963) 2.
49. Feller, D.F., and Ruedenberg, K., Theoret. chim. Acta. (1979) 52, 231.
50. Schmidt, M.W., and Ruedenberg, K., J. Chem. Phys. (1979) 71, 3951.
51. Wilson, S., Theoret. chim. Acta. (1980) (in press)
52. Wilson, S., Theoret. chim. Acta. (1980) (in press).

RECEIVED August 17, 1981.

The Academic Computer Resources in Japan and Their Contributions to Theoretical Chemistry

KATSUNORI HIJIKATA

University of Electro-Communications, Chofu-shi, Tokyo 182 Japan

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch002

Resources

The recent growth of computer resources in Japan is quite remarkable. Reduction in the cost of LSI has enabled the vendors to offer large CPU memories for low prices, inconceivable a few years ago, resulting in the revolutionary development of computer laboratories in universities and institutes.

There are three vendors of large computers in Japan, namely Hitachi, Fujitsu and NEC. In Fig.1 the largest current computers of these vendors are compared with the U.S. computers. The numbers are MIPS values, showing that these occupy good positions among non-supercomputers.

Fig.2 shows the locations of national universities and institutes organized by an inter-university computer network. The universities listed in the right upper part have computers shown in Fig.1, which are connected by DDX network of NTTPC (Nippon Telegraph and Telephone Public Corporation). The symbols in the parentheses are the vendors, and are followed by numbers showing the size of CPU memory in MB. These numbers are very uneven, just indicating the system before or after replacement. The four institutes below the map also have largest computers to be incorporated in the network in the near future. A number of remote stations around the large computers are not mere RJE's but independent computers of appreciable size. For instance, the University of Electro-Communications has M170, having the speed of 1MIPS, the CPU memory of 4MB and the discs of 2300 MB, which is connected to the computer of the University of Tokyo through an exclusive telephone line. It is at the users' disposal which to use.

The computers of private universities, not shown in the figure, are great many, some being of considerable size, but none of largest size. Quite a few are remote stations of the large computers of national universities.

The new computer system of the University of Tokyo is really something. It has 8 units of M200H, 4 loosely coupled pairs,

0097-6156/81/0173-0039$05.00/0
© 1981 American Chemical Society

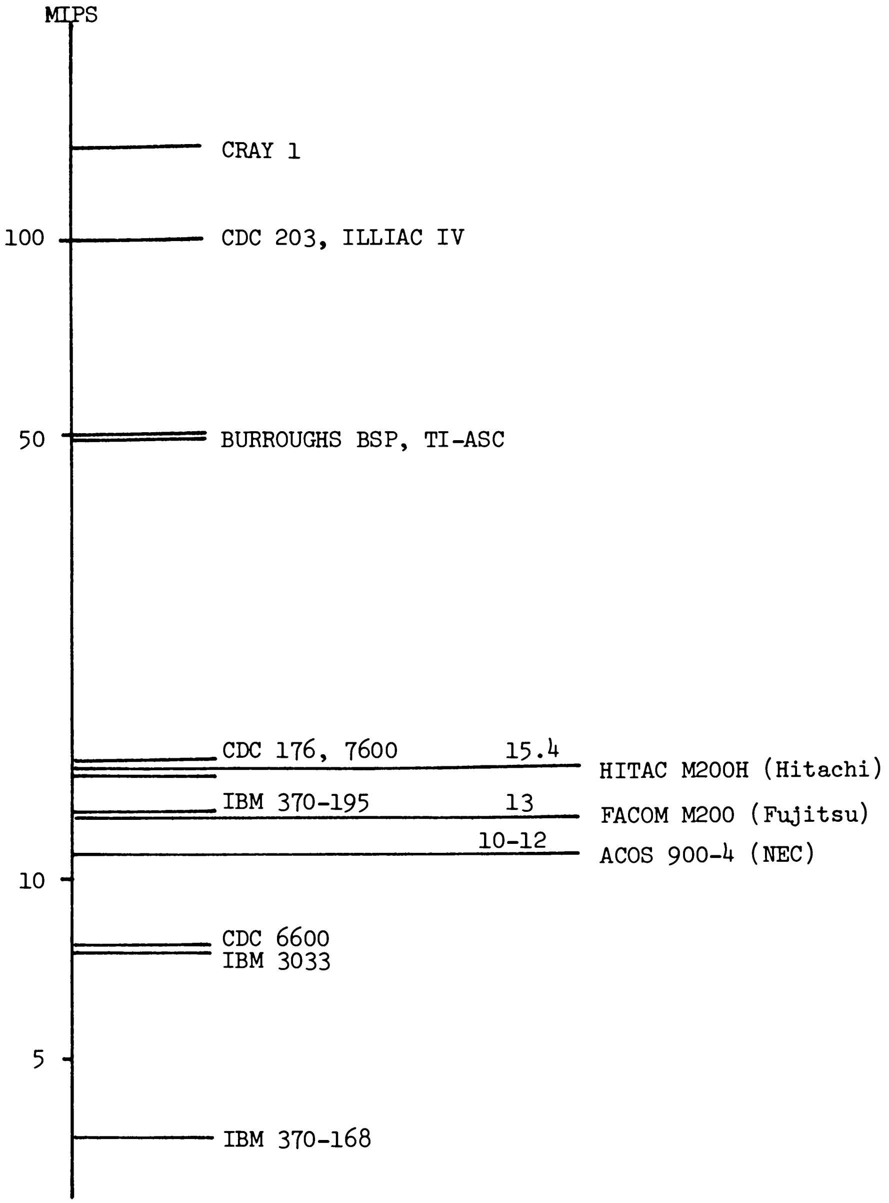


Figure 1. Japanese vs. US computers.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch002

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch002

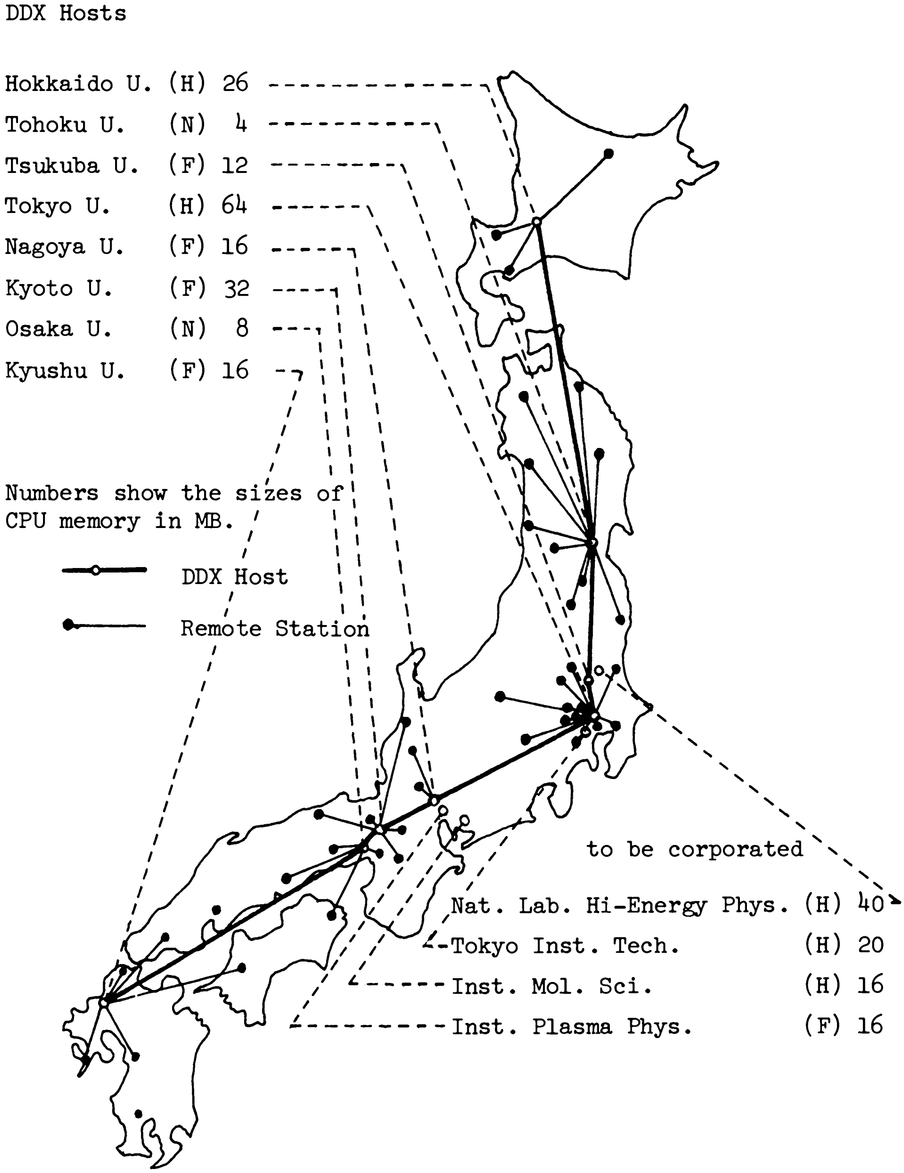


Figure 2. Computer resources of national universities and institutes.

total memory being 64MB, drums of 120MB, disks of 36GB, and a MSS of 36GB. 5 communication control processers are connected by 229 lines to the remote stations and TSS terminals (supposedly 2000). With abundant peripheral equipments, it is probably the largest computer center in the world at this moment. The replacement was started in May and will be finished within this year (1980).

Since the renewal of the system is set in the budget almost once in six years, starting with the University of Tokyo, we may easily imagine the future of the national computer resources.

Contributions to Theoretical Chemistry

Before the replacement, the computer system of the University of Tokyo was 4 units of H8800, each having the speed of 3MIPS. In this computer's statistics of 1979, (April 1979 through March 1980), we can find the following numbers:

Total CPU Time	32,129,646 sec
No. of Jobs Processed	881,575
No. of Users	4,811

which means that the length of an average job was 36 sec and an average user used the machine for 1.8 hrs a year. A job taking more than 30 min and a data set larger than 2MB were not allowed in any circumstances. The outstanding top user consumed 53 hours.

The situation was not so severe in less populated districts, say Hokkaido, where Ohno's group finished the system program JAMOL II and the calculations of excited states of O_2 and of metal-porphines. Still they had to manipulate magnetic tapes acrobatically because of the limited machine-time for one job. Most of us had either to be satisfied with semi-empirical -NDO calculations or to come to the U.S.

It is not exaggeration, therefore, to say that ab initio calculations were not commonly commenced untill the computer center of IMS (Institute for Molecular Science) started in Jan. 1979. It was a hard job to persuade the people (inside and outside the institute) of the existence and the scale of our problems, but we finally made it.

Then, what is the result of the blessed one year? Table I shows the machine-time request collected in March 1980, the total amounting to 4356 hours. New projects must be accepted in September, and of course the staff of IMS must hold their own machine-time. Therefore, the main business of the administration committee is to make retrenchment for these requests. The average request 51.2 hrs is three times as large as 13.6 hrs in the last September. (In 1979, IMS machine was M180. The average request 34 hrs is converted to M200H.) 10 projects are of solid states, partly on band calculations and partly on

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch002

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch002

Table I

Machine-Time Requests to the Computer of the Institute for Molecular Science
(1980)

(a) Gross Classification

	hrs	No. of Projects	Av. hrs
Ab Initio	2508	51	49.2
Solid State, Surface	440	10	44.0
Molecular Dynamics	530	4	132.5
Polymers & Proteins	550	5	110.0
Others	328	15	21.9
Total	4356	85	51.2

(b) What in Ab Initio?

	hrs	No. of Projects	Av. hrs
Organic & Bio-chem. oriented	1030	15	68.7
Simple Molecule	473	11	43.0
Potential Surfaces	660	15	44.0
Complex Salts	190	6	31.7
Molecular Force Fields	155	4	38.8
Total	2508	51	49.2

(c) IMS Staff's Machin-Time (not included in (a) and (b))

Ab Initio	1720	5	344
Solid State, Surface	500	2	250
Others	10	1	10
Total	2230	8	278.8

In Supercomputers in Chemistry; Lykos, P., el al.;
ACS Symposium Series; American Chemical Society: Washington, DC, 1981.

surface states. This group has potentiality to compete with ab initio calculations, and in the near future a border line must be set up against the projects inappropriate to IMS. The projects on molecular dymanics are very few, but naturally take longer machine-time. Once proved feasible, however, enormous expansion is expected in this field. Polymers and Proteins mean simulations for folding and bridging, also taking long time. "Others" are supplementary means of experiment, such as to draw electron density diagrams in X-ray spectroscopy.

Among the ab initio calculations, many are oriented to large molecules and potential surfaces in reactions. There are almost no work on diatomic molecules and very few are principle-oriented. As a whole the projects of this institute seem to be along the IMS's guide-lines, "Molecular Designing of Chemically Interesting Materials" and "Energy Transformation by Molecular Interaction". These calculations are mostly by means of modified Gaussian 70, and JAMOL II, III and COMICAL 2 developed by Hokkaido group.

Some members of the atomic collision research group are using IMS computer, but the majority of this group are the customer of the computer of the Institute of Plasma Physics. The database activity of this institute is very appreciable and large calculations are planned here.

A few people are working on Hylleraas type and James-Coolidge type wavefunctions of many-electron systems. Their present calculations are preparatory, but will expand enormously in the production steps. We should not forget that these calculations are most important from the basic point of view.

Prospects and Problems

In the new computer system of the University of Tokyo, the 8 central processers are divided: 2 for supervision, 2 for TSS, 2 for batch-jobs and 2 for large-scale computations. Thus for the first time the large computer of a university is open to our field. The university has ordered the vendor to replace by 1983 the last 2 by a supercomputer and started to discuss on the processer desirable for the expected large computations. If the other universities follow this way, they will share a fair amount of the computations now concentrated to IMS. Of course IMS itself must take hold of a supercomputer as soon as possible. A tight cooperation, therefore, has been established between IMS staff and the supercomputer group of Hitachi Ltd.

M200H has an integrated array processer (IAP). Through the benchmark test in IMS, however, we found out that at best only 30% of our program was vectorized, gaining less than 20% of the total machine time. It is quite natural, therefore, this array processer is out of present users' concern. Since the vendors are most seriously persuing vectrozation, however, excellent products will be offered very soon.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch002

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch002

Consequently, we must have our package programs prepared for vectorization. Is this a laissez-faire business of users? Or is this to be done by vendors? It is a waste of time for several users to revise the programs independently. Our vendors, on the other hand, have never touched with application programs. The maintenance and dissemination of the programs has been unsolved problem in Japan and is to be pointed out again in this occasion.

Our inter-university network was planned originally for academic information exchange, and is functioning as the medium of information retrieval and a number of databases. The other important function of the network is the medium for common use of the resources. From the administrative point of view, it is a complicated problem how to charge the users machine time rate and network fee. In the University of Tokyo the rate is 7 yen per second and the other universities normalizes their rates to equalize the cost-performaces, while IMS and other institutes do not charge at all. We shall have the chaos when they join the network. Furthermore, if the average project of IMS is processed by a university machine, the charge will exceed the amount allowed for a normal university employee. We must either reduce the rate or to request the increase of research funds.

The problem is not limited to molecular science, but in common with other fields, such as hydrodynamics, structural mechanics, nuclear physics, geophysics, etc. Policies must be established before the completed network facilitates the common use of national computer resources. Thus we need a national organization to manage the following affairs:

1) Selection and integration of projects in accordance with social and academic importance,
2) Financial support for the projects and the management of computer network,
3) Development and maintenance of the system programs for large computations, and
4) Publication of the results in commonly appreciable form.

The organization may well be a part of National Information Center for Science and Technology which has been contemplated by the Government for these years.

Acknowledgements

The author collected the informations from the Computer Centers of the University of Tokyo and of the Institute for Molecular Science where he is a member of the Administration Committees. He is glad to acknowledge his indebtedness to these institutes. He also expresses his thanks to Prof. Peter Lykos and Prof. Isaiah Shavitt for their invitation to the Symposium, and to the Ministry of Education, Science and Culture for the travel grant.

RECEIVED March 18, 1981.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch003

3

Supercomputer Requirements for Theoretical Chemistry

ROBERT B. WALKER, P. JEFFREY HAY, and HAROLD W. GALBRAITH

Los Alamos National Laboratory, Theoretical Division, Los Alamos, NM 87545

The electronic computer is undeniably an essential component in the tool bag of the modern theoretical chemist and, with ever improved accessibilty to more powerful computing, it is tempting to feel a sense of euphoria about current computer capabilities. At Los Alamos, we have a truly impressive resource of large computers -- at present we have a choice of two 1-million word CRAY machines and four CDC 7600 machines. However, even with this powerful a computing environment, the main objective of this talk is to ask whether or not this current euphoria is really justified. Have we yet reached the stage where we can, with current computers, address problems which are traditionally thought of as chemistry? The answer is, in many cases, no.

In this talk, we survey three areas of theoretical chemistry which receive considerable attention at the Theoretical Chemistry and Molecular Physics Group at Los Alamos. These three areas are (1) molecular electronic structure calculations, (2) chemical dynamics calculations, and (3) quantum optics and spectroscopy. In introducing each category, we will note the types of mathematical algorithms used to solve problems typical of each area. We then present examples of types of calculations which we feel are at the current state-of-the-art. Finally, we will present a wish list of problems in each category which we would like to be able to study, but are simply beyond current computing capabilities.

The program of this symposium makes it clear that there will be several talks to follow which will concentrate specifically on problems associated with electronic structure calculations -- we will only skim over the subject for now. We will concentrate more heavily on problems in chemical dynamics, and conclude with problems in quantum optics.

0097-6156/81/0173-0047$05.00/0
© 1981 American Chemical Society

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch003

Molecular Electronic Structure Calculations

In determining the quantum mechanical structure of a molecule, there are three major steps (roughly equal in difficulty) to be attacked computationally (See Fig. 1.). We first define a set of atomic basis functions centered at each nucleus and then compute a large number of integrals over these basis functions. This information feeds into the construction of the Fock matrix. The eigenvalues of the Fock matrix are associated with the total energy of the system. The eigenvectors define the occupied orbitals of the system and the eigenvalues define the orbital energies. This information is used to construct a new Fock matrix, which is again diagonalized. This procedure is repeated interatively until the total energy of the system is minimized and the orbitals are constant from one iteration to the next.

Reasonable determinations of molecular structure can be obtained at this SCF-level of calculation. However, for an accurate determination of the structure and properties of molecules, correlations between the motions of the many electrons of the system must be included. To do this, many-electron wavefunctions are computed using sums of products of these one-electron orbitals. This process is the configuration interaction (CI) method; getting accurate CI wavefunctions and energies requires an enormously large basis of SCF functions. The Hamiltonian matrix in this basis is constructed and diagonalized to get the accurate CI wavefunctions and energies. The CI matrix tends to be both very large and very sparse. Efficient computer codes must take into account the sparsity of the CI matrix both in its construction and diagonalization phase.

Let's turn our attention to the computational needs of the structure problem (See Fig. 2). Defining n as the number of atomic basis functions employed, there are several characteristic matrices to consider. The Fock matrix, which we repeatedly construct and diagonalize until convergence, is only an n x n matrix; unfortunately, to construct this matrix at each iteration of the SCF procedure, we have to process the $n^4/8$ two-electron integrals. At the current state of the art, the number of atomic basis functions n tends to be about 100. This limitation is not so much because of the difficulty of diagonalizing 100 x 100 matrices, but because of the IO limitations inherent with processing the tens of millions of integrals at each iteration.

It is easy to see why the CI step is time consuming. Although the Fock matrix is only n x n, the size of the CI matrix goes more like n^4! Now the CI matrix is very sparse, as indeed it has to be, if we are to get some of the eigenvectors and eigenvalues of matrices which can get to be as large as 10000x10000. Most of the electronic structure work at LASL is concentrated on the CDC 7600 machines. We are currently adapting our programs to use the CRAY machines efficiently, but it appears the CRAY's will not be more than 10 times as powerful as a 7600. It isn't hard to think of problems which would overwhelm the CRAY's.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch003

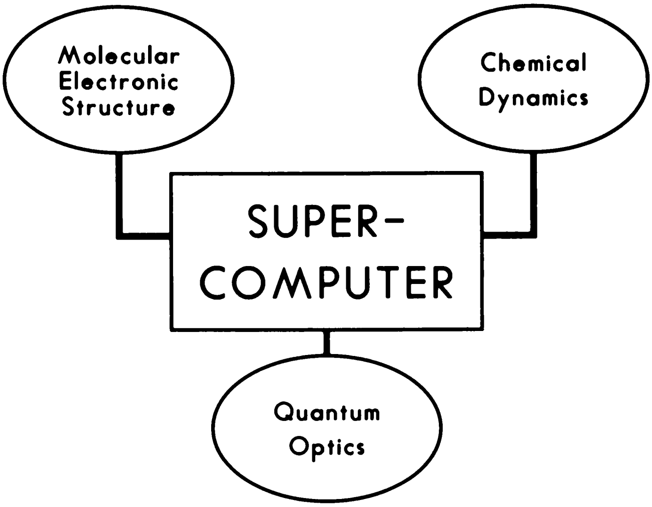


Figure 1. Schematic representation of three areas of theoretical chemistry. The relationship between each of these areas and the modern supercomputer is discussed.

● Let n = # atomic basis functions

Matrix	Size	Limits	
		7600	CRAY-1
Fock	n × n	100	300
2 e^- $\int$	$n^4/8$	10^7	10^9
CI	~ $n^4 \times n^4$	10^4	10^6

Figure 2. Computational requirements for molecular electronic structure calculations.

In Fig. 3, we consider a few problems of interest to the structure chemist in the area of transition metal chemistry. The molecules we consider here are of interest for their bonding properties and their electronically excited states. The $Re_2Cl_8^-$ ion has a strong (almost quadruple) Re $\equiv$ Re bond -- we estimate that we can do a fair job of determining the energy of this ion with about 1/2 hour of CRAY time, using about 100 orbitals in a split-valence basis. Not all the electrons in this system would be treated explicity. At LASL, we regularly treat large Z atoms using effective core potentials to eliminate the innermost core electrons from the calculation. (1) Using about 250 orbitals, we can treat this mixed-valence ruthenium-pyrazine complex. This complex is of interest because of its metal-organic bonding and the fact that the two ruthenium atoms are not equivalent to each other, even at the Hartree-Fock level. Of interest also is this bridged rhodium complex, which we estimate we can tackle with about 350 orbitals. This molecule has the interesting property that, when dissolved in water, it liberates hydrogen gas in the presence of sunlight. This would be a very tough problem, even for the CRAY -- we estimate 60 hours of CRAY time to determine the structure.

One of the more relevant duties of the structure chemist is to provide potential energy surfaces to the chemical dynamicist. The potential energy surface is determined by computing the electronic energies of the molecular system as a function of the nuclear geometry. In addition, if several electronic states participate in the collision dynamics, it may also be desirable to have available certain matrix elements between electronic surfaces. Now dynamicists tend to be rather demanding -- at least by request if not also by need. Consequently, we arrive at this first law of potential surface calculations -- the structure chemist gets bored with running his program long before he can satiate the dynamicist. (Paraphrased from Fig. 4.) But look what happens -- even if the dynamicist compromises to the point that he settles for 10 points per nuclear degree of freedom, it nevertheless requires a bundle of structure calculations to generate a surface for a relatively simple A+BC type reaction. Now suppose we had a dynamicist who dared to study a four-body reaction, like AB+CD. Then imagine a structure chemist willing to compute a million points on a potential surface. It shouldn't be surprising that there is at present only one potential surface which has been computed at enough points and enough accuracy to satisfy the dynamicist -- the simplest of all neutral molecular systems -- the $H+H_2$ surface computed by Bowen Liu and Per Siegbahn. (2) Because of the high symmetry of this system, they have calculated a surface at about 250 points (instead of the 1000 estimated).

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch003

Molecule	n*	Estimated CRAY time (hrs)
$Re_2Cl_8^{2-}$	104	0.5
$[[(NH_3)_5Ru]_2\text{–pyrazine}]^{5+}$	252	20
$[Rh\text{–}(NC\text{–}C_3H_6\text{–}CN)_4\text{–}Rh]^{2+}$	340	60

*split-valence basis using effective potentials

Figure 3. Examples of problems in transition-metal chemistry.

Law of Nature: Dynamicists will always want more points on a potential energy surface than one is willing to calculate.

Response of Electronic Structure Practitioners: Dynamicists will usually settle for 10 points / degree of freedom.

	Points
Triatomics	10^3
Tetratomics	10^6

Figure 4. Potential energy surface calculations for the chemical dynamicist.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch003

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch003

Quantum Chemical Dynamics

Let's now turn our attention to the requirements of the chemical dynamicist. Here we consider only quantum mechanical approaches to chemical reaction dynamics, and only mention that there also exists a considerable computational technology which treats chemical dynamics by using classical mechanics.

The only type of chemical reaction we are likely to ever be able to solve rigorously in a quantum mechanical way is a three-body reaction of the type A+BC → AB+C. (See Fig. 5.) The input information to the dynamicist is the potential energy surface computed by the quantum structure chemist. Given this potential surface, we treat the nuclear collision dynamics using Schrödinger's equation to model the chemical reaction process.

As was mentioned earlier, there is only one fully ab initio potential energy surface for chemical reaction available to the dynamicist. This surface is appropriate for an A+BC reaction where A,B, and C are all three hydrogen atoms or hydrogen isotopes (H,D,T). Fig. 6 shows a contour map of the collinear part of this surface (all three nuclei lie on a single line); the essential features of the surface topology are the entrance valley, the product valley, and the activation barrier separating these two valleys. Motion perpendicular to each valley corresponds to vibration of the reactant or product molecule, and motion parallel to the floor of the valley measures progress of the reaction, from reactants to products. The classical mechanical solution to chemical reaction dynamics is accomplished in fact by solving for the motion of a point mass particle on this hypersurface. Reaction corresponds to a trajectory which starts out in the reactant valley, crosses the barrier, and ends moving out into the product valley.

Quantum mechanically, the reactive dynamics is expressed in a more wavelike language. By solving Schrödinger's equation, we treat the problem where an initial probability wave of reactants is sent in towards the activation barrier from reactants. When the wave hits the barrier, part of it is reflected and part of it is transmitted. The reflected part of the wave corresponds to non-reactive collision events, and the transmitted part corresponds to reaction.

The actual equations we solve are called the close-coupled equations. (See Fig. 7.) They are obtained from the Schrödinger equation in the following way: (1) we first define all but one of the coordinates of the system to be "target" coordinates and the final coordinate is called the "scattering coordinate" or "reaction coordinate." The reaction coordinate tells us where we are in our journey along the potential surface from the reactant valley towards the product valley. Basis functions are defined which describe motion in all the target coordinates. These basis functions are square integrable for the target coordinate degrees of freedom, but the function which describes motion in the reaction coordinate is determined numerically. The equations

- $A + B{-}C \rightarrow A{\cdots}B{\cdots}C \rightarrow A{-}B + C$
- Requires potential energy surface(s) from electronic structure calculations
- Solve Schrodinger equation for dynamics

$$H\Psi = E\Psi$$

Figure 5. Quantum chemical dynamics. Scope and method of currently tractable problems.

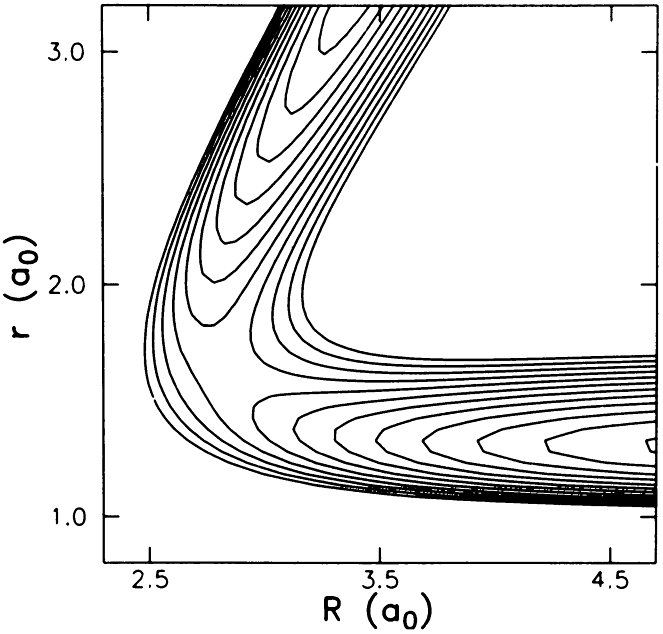


Figure 6. Contour map of $H+H_2$ collinear chemical potential energy surface.

- Separate all (3N−3) coordinates into one scattering coordinate and (3N−3)−1 internal coordinates
- Expand wavefunction using square integrable basis functions for (3N−3)−1 coordinates and solve numerically for function of scattering coordinate.
- Leads to a set of coupled linear second order differential equations. One equation for each "channel."

Figure 7. How close coupled equations are obtained in chemical dynamics problems.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch003

for these scattering functions are the close-coupled equations. These equations are a set of coupled second order linear ordinary differential equations. The difficulty in solving problems in quantum chemical dynamics is simply this -- how many coupled equations are there? The answer is that there is one equation for every "channel" in the close coupling expansion.

Each channel is defined by a unique set of quantum numbers for the target degrees of freedom. There are five such labels for each channel. They are (1) J -- the total angular momentum and (2) M, its projection on an axis fixed in space. In addition there are labels (3) n for the vibrational motion of the molecule, (4) j for the molecular rotational degree of freedom, and (5) ℓ for the atom-molecule orbital angular momentum. The equations for one set of (J,M) are uncoupled from equations for other values of (J,M). The equations for a function labeled by one value of (n,j,ℓ) are coupled to values of all the other functions labeled by (the same or) different values of (n,j,ℓ). The number of coupled equations we have to solve therefore depends on the number of molecular vibration-rotation states we have to treat in the scattering dynamics at each collision energy.

In the next paragraph, we present a rudimentary look at the algorithm we use to solve these coupled equations. This method is called R-matrix propagation; (3) and although there are several other methods equally capable of solving the coupled equations, we use R-matrix propagation as an example because it illustrates the kind of computer algorithms we require. The R-matrix itself contains the scattering information we need; the final R-matrix is assembled in a recursive fashion using the analytic solution of the scattering problem over a small region of the scattering coordinate. The algorithm works in the following way: given (1) an old R-matrix associated with the solution of the scattering problem over one region of space; and given (2) a sector R-matrix which defines the scattering solution over a small incremental region of space, we can (3) assemble a new R-matrix which is associated with the solution of the scattering problem over the (old + incremental = new) region of space. The recursion equation is a matrix equation of order n,

$$\underset{\sim}{R}_2 = \underset{\sim}{r}_{22} + \underset{\sim}{r}_{21}(\underset{\sim}{R}_1+\underset{\sim}{r}_{11})^{-1}\underset{\sim}{r}_{12}$$

where there are n channels in the close-coupling expansion of the wavefunction. As you can see, this recursion formula involves very standard matrix operations -- multiplication and inversion. The analytic solution of the coupled equations in the incremental region is defined in terms of the eigenvalues and eigenvectors of the coupling matrix. So you can see that the basic numerical algorithm we require our supercomputer to handle effectively are standard matrix operations -- multiplication, diagonalization, and inversion. All these algorithms go asymptotically as n^3 -- and so the complexity of the quantum dynamicist's problem is measured (as we said previously) by the size (n) of the close coupled equations.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch003

So let's return our attention again to the question of the size of the coupled equations and consider some examples. Many of the chemical reactions we are interested in are dominated by an activation barrier which separates the reactant and product valleys of the potential energy hypersurface. (See Fig. 8.) The energy of this activation barrier locates the general energy range of interest to the reaction dynamicist -- because there isn't very much reaction at energies below the barrier height, where only quantum tunnelling processes can contribute to reaction. But, as we show schematically in Fig. 8, there may be several molecular energy states below the activation barrier. All these states, at the very least, must be included in the close coupling expansion.

Let's now consider several examples. The simplest of all reactions is the $H+H_2$ reaction. The H_2 vibrational levels are fairly widely spaced, but we must also include the rotational manifold of levels associated with each vibrational level. (See Fig. 9.) Now, it is this rotational manifold of levels (and the degeneracies of states associated with each vibration-rotation level) which ultimately breaks the bank in the size of the close coupling expansion.

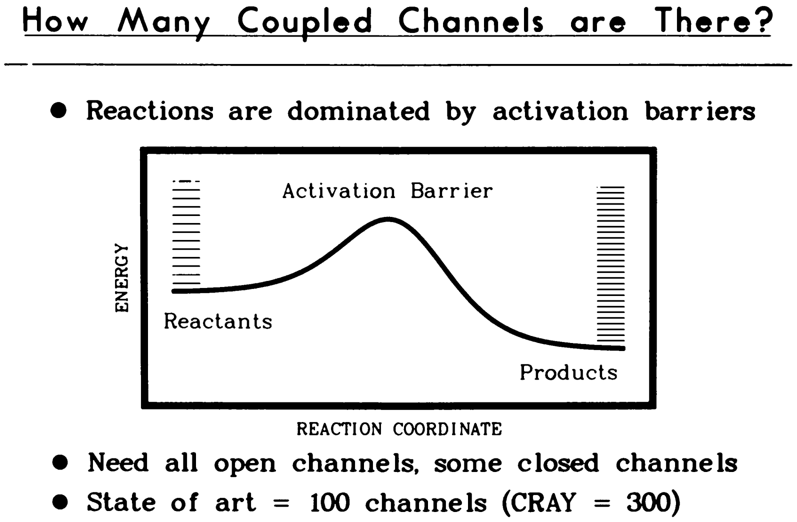
In order to treat quantum dynamical problems, it will be necessary to introduce approximations which reduce the size of the set of coupled equations. Two promising approximations are the centrifugal sudden (CS) (4, 5) approximation and the infinite order sudden (IOS) approximation. (5, 6) The CS approximation removes the coupling between the j and ℓ angular momenta, thereby reducing the size of the coupled equations from n to approximately $n^{1/2}$. In this approximation, each (n,j) energy level generates only one channel instead of (2j+1) channels. The more drastic IOS approximation appears to be promising for systems in which the molecular species rotates very slowly on the scale of the collision time. This approximation removes in effect all the rotational levels from the system.

For the $H+H_2$ system, we estimate that we can just about solve this easiest of all problems with current state-of-the-art computers at the 100-channel level. If we can use the CS approximation for this system, we can in fact go to quite high scattering energies.

But remember that $H+H_2$ is the simplest of all reactions. Moving more in the direction of true chemistry, consider next a reaction for which only two nuclei are hydrogens (instead of three) -- the $F+H_2$ reaction. This reaction is over 1 eV exothermic in going from the reactant valley, over a small (1 kcal) barrier, to the product valley. The exothermicity of reaction means that there are several energetically accessible (open) vibrational channels for this system even at the threshold for reaction. If we include all the rotational levels with each vibration, and the proper (2j+1) rotational degeneracies, we have an unthinkably large number of coupled equations to solve -- over 1200 channels. (See Fig. 10.) To solve this problem, we must

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch003

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch003



How Many Coupled Channels are There?

- **Reactions are dominated by activation barriers**

Activation Barrier
ENERGY
Reactants
Products
REACTION COORDINATE

- **Need all open channels, some closed channels**
- **State of art = 100 channels (CRAY = 300)**

Figure 8. Schematic representation of chemical potential energy surfaces. Counting of states below reaction barrier for both reactants and products gives a minimal estimate of numbers of coupled equations to be solved.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch003

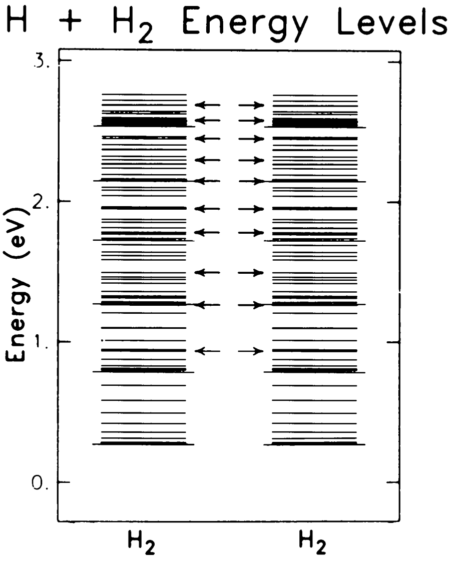


Figure 9. Counting of channels for the H_3 reaction. Reaction barrier is at 0.4 eV; state-of-the-art calculations are performed to slightly above 1 eV. Arrows are drawn whenever another 100 coupled channels are required.

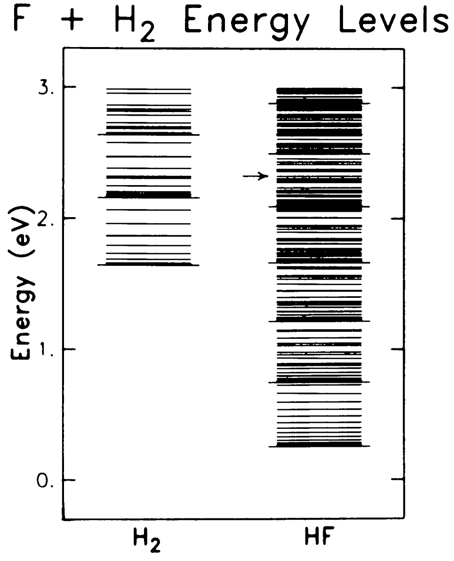


Figure 10. Counting channels for the FH_2 reaction. Conditions are as in Figure 9, except that arrows count states for CS approximation (each vibration–rotation level counts only once). Reaction threshold is at 1.65 eV.

use approximations such as the CS approximation, which reduces the problem to the much more manageable 100-channel level.

For our final quantum dynamics example, consider what happens when we substitute a lithium atom for one of the remaining hydrogens -- the Li+FH reaction. (See Fig. 11.) This semi-empirical potential surface (collinear) shows a narrow entrance channel vibrational valley, a shallow well in the entrance channel, a barrier, and a broad product vibrational valley. Even using the CS approximation, the energy level diagram for this reaction makes this problem accessible only to the full power of a CRAY level machine. Anyone foolish enough to tackle the problem rigorously will have to face a 10000-channel system at energies just above threshold!

The moral of our story of the quantum chemical reaction dynamicist should be perfectly clear -- at least two hydrogens are the dynamicists best friend. Indeed, our current supercomputers may seem to be a bit less super.

Quantum Optics

The interaction of molecules with electromagnetic radiation is of fundamental interest to the chemist. When the electromagnetic field is relatively weak, we can describe these interactions using perturbation theory. The study of single photon transitions induced between molecular states by weak fields is the province of the molecular spectroscopist. But now, with the ever more powerful radiation fields available from laser technology, we are in a position to study the interaction between molecules and electromagnetic radiation at intensities too large for perturbation methods to work. The somewhat broader field of quantum optics seeks to describe the time evolution of molecules in the presence of these intense fields. Of course, before we can follow the migration of energy among the various degrees of freedom of a (possibly large) molecule, we must first know what the electronic, vibrational, and rotation energy states of the molecule are in the absence of any radiation field. The effect of the field is to move population from the initial molecular state into other molecular states in a time dependent way. The solution of this problem can be obtained by solving the time-dependent Schrödinger equation, so long as the molecule we are studying is modeled at zero pressure. At finite pressures (when collisions are present) the Schrödinger picture is too difficult to solve directly; in this case, we can model the incoherent (phase destroying) effects of collisions upon the coherent excitation induced by the electromagnetic field by resorting to a Bloch equation (or density matrix) formalism. Collisions are modelled by decay rates not only in the diagonal (but also the off-diagonal) terms of the density matrix. We also have one further constraint in developing methods to treat problems in quantum optics -- because laser pulses last for a relatively long time in comparison to the time associated with molecular vibration

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch003

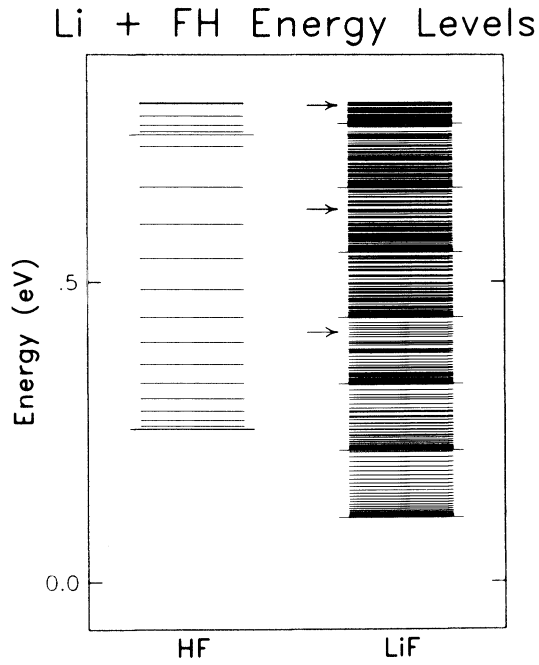


Figure 11. Counting channels for the LiFH reaction. As in Figures 9 and 10, arrows count states for CS approximation. Reaction threshold is near 0.6 eV.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch003

and rotation, we must solve our time-dependent Schrödinger or Bloch equation in a way which gives answers efficiently for long times.

Consider for a moment the Bloch equation for the density matrix, $\underset{\sim}{\rho}$,

$$i\dot{\underset{\sim}{\rho}} = \underset{\sim}{\rho}\underset{\sim}{H} - \underset{\sim}{H}\underset{\sim}{\rho} - i\underset{\sim}{\Gamma}\underset{\sim}{\rho}$$

The Bloch equation gives the time derivative of the density matrix ρ in terms of its commutator with the Hamiltonian for the system, and the decay rate matrix Γ. Each of the matrices, ρ, H, and $\underset{\sim}{\Gamma}$ are n x n matrices if we consider a molecule with n vibration-rotation states. We solve this equation by rewriting the n x n square matrix ρ as an n^2-element column vector. Rewriting ρ in this way transforms the H and Γ matrices into an n^2 x n^2 complex general matrix $\underset{\sim}{R}$. We obtain

$$\dot{\underset{\sim}{\rho}} = \underset{\sim}{R}\underset{\sim}{\rho}$$

The solution of the transformed equation is obtained by exponentiating this R matrix. To efficiently exponentiate this matrix we must first diagonalize it, exponentiate the eigenvalues, and back transform with the eigenvectors. This back transformation procedure is repeated for every time at which we wish to know the molecular populations.

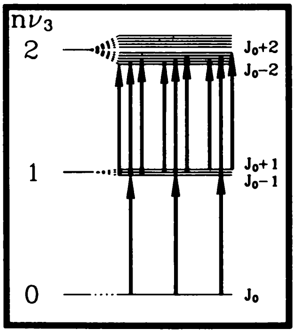
A typical problem of interest at Los Alamos is the solution of the infrared multiple photon excitation dynamics of sulfur hexafluoride. This very problem has been quite popular in the literature in the past few years. (7) The solution of this problem is modeled by a molecular Hamiltonian which explicitly treats the asymmetric stretch ν_3 ladder of the molecule coupled implicitly to the other molecular degrees of freedom. (See Fig. 12.) We consider the the first seven vibrational states of the ν_3 mode of SF_6 ($6\nu_3$); the octahedral symmetry of the SF_6 molecule makes these vibrational levels degenerate, and coupling between vibrational and rotational motion splits these degeneracies slightly. Furthermore, there is a rotational manifold of states associated with each vibrational level. Even to describe the zeroth-order level states of this molecule is itself a fairly complicated problem. Now if we were to include collisions in our model of multiple photon excitation of SF_6, we would have to solve a matrix Bloch equation with a minimum of 84^2 x 84^2 elements. Clearly such a problem is beyond our current abilities, so in fact we neglect collisional effects in order to stay with a Schrödinger picture of the excitation dynamics.

In the Schrödinger picture, we can include the diagonal elements of the Γ matrix, which model the coupling of the explicitly treated ν_3-ladder states with the other implicitly treated molecular states. The exponentiation of the coupling matrix in the Schrödinger picture requires the diagonalization of an n x n complex general matrix. Populations at several times are

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch003

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch003

- **$SF_6 + Nh\nu + M \longrightarrow SF_6^{\dagger} + M$**
- **Asymmetric stretch (ν_3) ladder dynamics**



- **Vibrational degeneracy is .5(N+1)(N+2)**
- **Include up to $6\nu_3$ in H, get $84^2 \times 84^2$ matrix to diagonalize**

Figure 12. Schematic of multiple photon excitation dynamics of SF_6. Groups of levels show lowest three ν_3 vibrational states. Higher states are split by rotational interactions with vibrational motion.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch003

computed by the back transformation method, and a quadrature over time of those populations gives the leakage of amplitude into the SF_6 quasicontinuum degrees of freedom. This whole process is repeated for each new initial rotational state J_o, laser frequency ν, and laser intensity I. Our calculations at LASL can require up to 10 hours of 7600 time for each laser power of interest.

There are several other interesting topics in quantum optics which we would like to be able to study. For example, we would like model problems in double resonance spectroscopy, where there are two electromagnetic fields with possibly different polarizations simultaneously interacting with a molecule. This problem resembles the multiple photon excitation problem in that there is population migration along ladders of states, but in this case there can be a vastly larger number of quantum levels to treat -- on the order of 2(2J+1). At room temperature, the most probable value of J for SF_6 is about 60, which implies a 250 state calculation.

Finally, we also mention a substantially more complex problem -- that of laser pulse propagation through an absorbing medium. In this case we are asking not only what happens to the molecule in the presence of an electromagnetic field, but also what happens macroscopically to the field in the presence of the molecule. The solution of this problem requires treating the multiple photon dynamics problem self-consistently with a solution of Maxwell's equation over a grid of points in space.

Conclusions

In summary, our intention has been to give examples of the types of problems we are interested in at LASL. Our appetite for computationally difficult problems has not been dulled by the current availability of computer resources. In the area of molecular electronic structure calculations, we need computers for which there can be written efficient algorithms to diagonalize large matrices, and in the case of CI calculations, we need efficient indirect addressing capabilities (gather-scatter operations) in order to process these matrices whose elements are 99% zeroes. Either we need efficient IO capabilities in order to process the lists of millions of integrals, or it has to be cheaper to calculate these integrals as we go along.

In the area of quantum dynamics, we need again computers capable of efficiently performing standard types of matrix operations (inversion, diagonalization, multiplication) on large matrices of the order of several hundreds.

And in the area of quantum optics, we need similar types of capabilities - standard matrix manipulations - but now our matrices are complex general instead of real symmetric.

In each of the fields discussed here, state-of-the-art calculations require the full capabilities of modern computers.

Newer supercomputers will need to be several orders of magnitude more powerful to efficiently attack many of the problems currently facing the theoretical chemist.

Literature Cited

1. Hay, P. J.; Wadt, W. R.; Kahn, L. R.; Bobrowicz, F. W. J. Chem. Phys. 1978, 69, 984.

2. Liu, B. J. Chem. Phys. 1973, 58, 1925. Siegbahn, P.; Liu, B. J. Chem. Phys. 1978, 68, 2457.

3. Light, J. C.; Walker, R. B. J. Chem. Phys. 1976, 65, 4272. Stechel, E. B.; Walker, R. B.; Light, J. C. J. Chem. Phys. 1978, 69, 3518. Light, J. C.; Walker, R. B.; Stechel, E. B.; Schmalz, T. G. Comput. Phys. Comm. 1979, 17, 89.

4. Pack, R. T J. Chem. Phys. 1974, 60, 633. McGuire P., Kouri, D. J. J. Chem. Phys. 1974, 60, 2488.

5. Kouri, D. J. in "Atom-Molecule Collision Theory: A Guide for the Experimentalist," ed. R. B. Bernstein; Plenum Press: New York, 1979; p. 301-358.

6. Khare, V.; Kouri, D. J.; Baer, M. J. Chem. Phys. 1979, 71, 1188. Barg, G.; Drolshagen, G. Chem. Phys. 1980, 47, 209. Bowman, J. M.; Lee, K. T. J. Chem. Phys. 1980, 72, 5071.

7. Galbraith, H. W.; Ackerhalt, J. R. in "Laser Induced Chemical Processes," ed J. I Steinfeld; Plenum Press: New York, to appear.

RECEIVED August 4, 1981.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch003

The Integration of Chemical Rate Equations on a Vector Computer

THEODORE R. YOUNG and MARIE C. FLANIGAN[1]

Naval Research Laboratory, Laboratory for Computational Physics, Code 4040,
Washington, DC 20375

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch004

With the advent of vector processors over the last ten years, the vector computer has become the most efficient and in some instances the only affordable way to solve certain computational problems. One such computer, the Texas Instruments Advanced Scientific Computer (ASC), has been used extensively at the Naval Research Laboratory to model atmospheric and combustion processes, dynamics of laser implosions, and other plasma physics problems. Furthermore, vectorization is achieved in these programs using standard Fortran. This paper will describe some of the hardware and software differences which distinguish the ASC from the more conventional scalar computer and review some of the fundamental principles behind vector program design.

The ASC can run in both scalar and vector mode. Its vectorizing capability will be demonstrated using the integrator VSAIM (Vectorized Selected Asymptotic Integration Method). This was developed at NRL to solve "stiff" sets of coupled ordinary differential equations and has been used extensively to solve sets of atomic and chemical rate equations in atmospheric combustion and solar physics models. Execution times for running VSAIM using various vector lengths will be compared with another version of the code, optimized to run in scalar mode on the ASC. These comparisons will demonstrate the difference in execution times obtained in vector and scalar modes for the same algorithm on the ASC. Such comparisons illustrate the impact of vector length on computing speed.

Benchmarks of selected pieces of code run on the ASC and on scalar machines (CDC 7600, Cyber 174, IBM 360) are also included. These examples will illustrate not only the power of certain vector instructions but also the difference in programming styles required to take advantage of the ASC vector hardware. General guidelines for writing vector code will be summarized briefly.

---

[1] NAS–NRC Resident Research Associate, 1978–80. Current address: Allied Chemical Company, Buffalo, NY

0097-6156/81/0173-0065$05.00/0
© 1981 American Chemical Society

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch004

ASC System Hardware (1)

The ASC differs from the conventional scalar computer in that it is a pipeline computer with a full set of hardware vector instructions in addition to the standard scalar instructions. The vector hardware includes arithmetic operations such as add, subtract, multiply, divide, vector dot product, as well as vector instructions for shifting, logical operations, comparisons, format conversions, normalization, merge, order, search, peak pick, select, replace, MIN, and MAX. Although an ASC may have one to four pipes, the configuration described below will be that of the two pipe machine at NRL.

The ASC Central Processor (CP) is made up of three types of components (see Figure 1): The Instruction Processing Unit (IPU), 2 Memory Buffer Units (MBU) and 2 Arithmetic Units (AU). These are organized such that each MBU-AU pair forms an 8-level pipe off of the 4-level IPU.

The "pipeline" structure allows instructions to be processed concurrently in all levels of the pipe in both scalar and vector mode. The eight levels of the MBU-AU pair under optimum conditions can each produce an output every CPU clock cycle (80 nsec). Pipe levels unnecessary to a particular instruction are bypassed. Figure 1 also illustrates how different sections of the arithmetic pipeline are utilized for execution of a particular instruction, i.e., floating-point addition and fixed-point multiplication.

The Instruction Processing Units' primary function is to access, interpret and route instructions to the pipelines. The Load Look-Ahead feature provides the IPU with advance address information to facilitate uninterrupted instruction processing. Inter-dependent data structures are recognized by the hardware scan of the instruction stream so that independent operations can be distributed to separate AUs.

The NRL ASC has a 32 bit word size and a million words of high speed central memory (CM) with a 160 nsec cycle time. All references to memory, whether fetching or storing, are made in 8-word increments (octets). Buffering in multiple octets for each pipeline (via the MBU) provides a smooth flow of data to and from each AU.

The pipelines achieve their highest sustained flow rate in vector mode. In this situation, a single instruction is interpreted and a single operation performed on many pairs of operands. For example, if A, B, and C are arrays, only one vector instruction is required for computing the sum C(I) = A(I) + B(I), I=1,100. The A and B values stream continuously into the pipes, additions are performed in discrete steps and results flow back to CM at the rate of one per CPU clock cycle per pipe. This is in contrast to scalar execution which requires five instructions to be executed 100 times; 2 fetches, 1 add, 1 store, and 1 counter incremention. If all eight words of the memory fetch and store are data for a vector instruction, the AU computes at full speed. If however

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch004

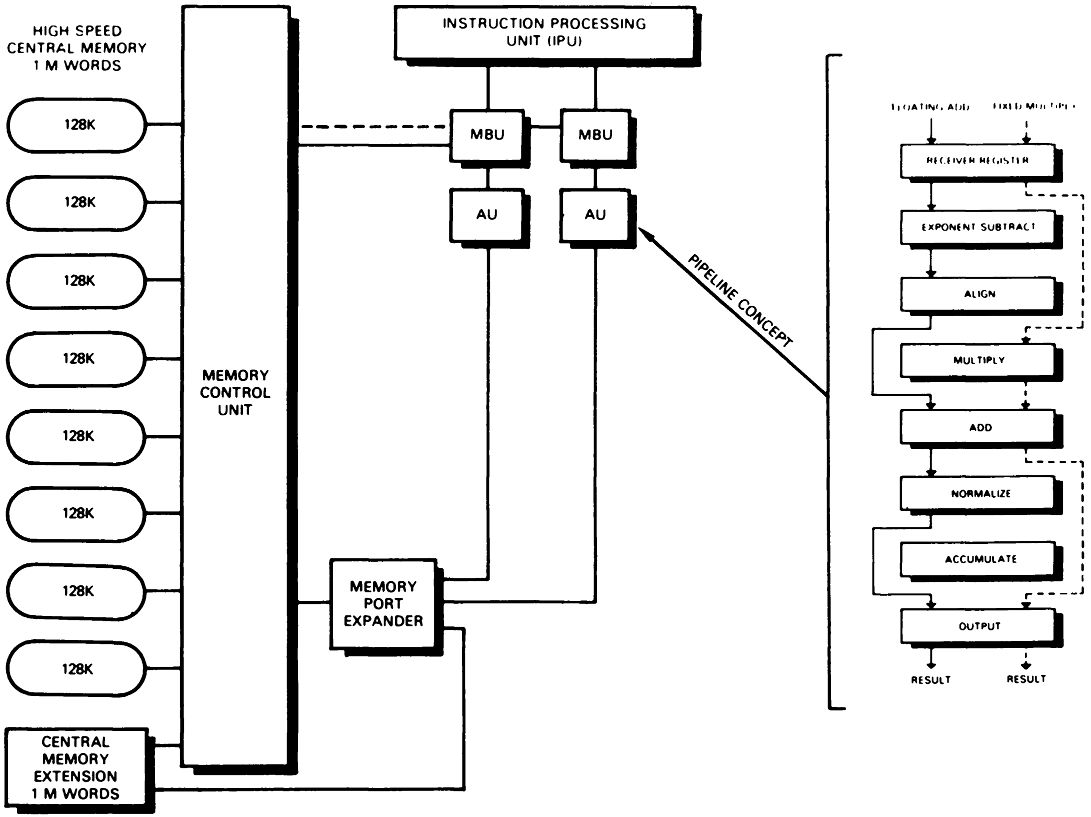


Figure 1. ASC central processor and memory configuration.

only part of the values in each octet can be utilized by the vector instructions, execution speed can be seriously degraded. Contiguous storage of operands in memory is thus an important factor in achieving full vector speed. Each pipe then has the capacity of accessing 25 million operands and producing 12.5 million results per second.

From this brief description it can be seen that executing in vector mode requires fewer instructions to be interpreted while guaranteeing the optimum flow of calculations and data through the pipes. Thus, less time is spent by the CP waiting for the next instruction or next piece of data; i.e., idle "bubbles" in the pipe are minimized. The programmer receives substantial help in vectorizing code from the NX compiler which orders code for optimal pipe usage. More will be said about the features of this compiler in the next section.

There are two types of disc storage available on the ASC: Head-per-Track (H/T) which has a transfer time of 491 K words/sec and Positioning-Arm-Disk (PAD) storage whose transfer rate is 200 K words/sec. Codes that require large data transfers to and from disk can make use of the Queued Direct Assess Method (QDAM) which is a collection of buffered, asynchronous disk I/O subroutines. The transfer rate using QDAM is 2 million words/sec. (2).

ASC Software (3)

The Operating System (OS) provided by TI for the ASC is a comprehensive and advanced general purpose system. The OS operates entirely within the Peripheral Processor, scheduling and allocating system resources in a multiprogramming environment as required by the users while permitting full utilization of the Central-Processor for users' problems and applications. Services provided by the OS include:

a. Job Specification Language (JSL) for job control and file management,

b. System library files which contain language processors, utilities, JSL macros, and applications routines,

c. Local and remote batch and interactive processing,

d. Job scheduling,

e. System recovery, and

f. Accounting information,

The well developed software which comes with the ASC is a very important and valuable feature. The Fortran compiler is of particular interest to the scientist who would rather spend his time doing science than optimizing code to achieve maximum utilization of the hardware. Machines which have very impressive hardware capabilities are often delivered with little software support. This philosophy fosters wasteful duplicate efforts when several installation sites are involved.

The ASC NX compiler supplied with the OS is a very sophisticated multi-pass program which compiles and optimizes a wide variety of source code.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch004

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch004

In addition to standard register utilization and optimization, the NX compiler treats all do-loops as independent blocks of code and searches each for operations which are possibly vectorizable. Whenever conditions permit the compiler will replace scalar instructions with vector instructions and allocate Vector Temporary Space (VTS) as needed. Regardless of the compiler dispensation of the potentially vectorizable statements, the programmer is notified and furnished with valuable documentation (i.e. VTS required, Loops reordered, Statement SSS not vectorized, etc.) and may even be given helpful hints on possible ways to achieve vectorization of code which, with minor changes, will vectorize. Further, the NX compiler performs an extensive dependency analysis on each block of code in order to isolate independent instructions which can be routed simultaneously through either pipe, thereby maximizing AU throughout.

A sample of various kinds of Fortran statements which automatically vectorize on the ASC have been included in Table I. The asterisk which appears next to certain statements indicates VTS required.

Table I
Software Vectorization

```
      DO 8 I = 1, N
 1    A(I) = B(I) + C(I)
*2    A(I) = B(I) - C(I) + D(I)
*3    A(I) = SQRT(B(I)**2 + C(I)**2)
      X = 0.0
      DO 7 J = 1, M
 4    X = X + BB(J,I)
 5    AA(I,J) = 0.0
      DO 6 K = 1, M
 6    AA(I,J) = A(I,J) + BB(K,I) * CC(K,J)
 7    BITS(I,J) = LSHF (WORDS(J), INDEX(I))
 8    A(I) = MIN(A(I), B(I))
```

* VTS REQUIRED

ASC Hardware and Software Performance

The objective of this section is to provide some benchmark comparisons between the ASC and some of our more conventional scalar machines, show some of the performance capability of the existing software provided with the ASC and present a few techniques useful for achieving vector optimization on the ASC. The latter may be useful to those involved with other types of hardware with parallel processing capabilities. Three machines with varying capabilities will be compared to the ASC under various circumstances. They are an IBM 360/91 and the CDC 7600 and Cyber 174.

The amount of optimization achieved in a given program will depend heavily on the nature of the problem being solved. Thus, program design and logic as well as specific algorithms used can strongly impact vectorization and thereby execution speed. Within these constraints, there are certain general guidelines for writing vector code using Fortran on the ASC (4). When attempting to vectorize a code, one wishes to make use of as many vector hardware instructions as possible. To do this, the code should be written to operate on arrays of data rather than on individual points. A vector instruction is employed as a Vector Parameter File (VPF) set up by the compiler which stores information in CM regarding the operation, input-output arrays, and indexing for stepping through the arrays. The vector instruction is then applied to all elements of the arrays as specified.

Table II lists the execution times for a DO-Loop which performs a simple addition of two vectors A and B. Times are given for both scalar and vector additions using one or two pipes as well as times for the same operation on the Cyber 174. Note that, for vector operations, increasing the array size is much less expensive than the corresponding increase on a scalar machine. For very small arrays (e.g. $N \geq 5$) scalar execution on the ASC is faster because Vector overhead consumes a much higher percentage of the execution time.

Table II
Execution Time for Vector Addition *

```
DO 10 I = 1, N
10 C(I) = A(I) + B(I)
```

			ASC+ Vector	
N	Cyber 174	ASC+ Scalar	1 Pipe	2 pipe
1	19.8	4.8	7.3	7.4
10	125.5	20.6	8.3	9.0
50	589.5	93.7	11.9	11.1
100	1168.9	183.0	16.2	13.6
500	5970.7	899.5	49.2	32.6
1000	11678.8	1795.0	92.3	51.7

* Execution Time in μsec

+ See Reference (4)

In general, the amount of temporary space required by an optimized algorithm is proportional to the degree of optimization achieved. The ASC compiler automatically allocates temporary

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch004

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch004

space when lines of source code contain multiple operations or standard mathematical library functions. The extra memory required for the Fortran Vector Math Library functions is well worth it as indicated by Table III which gives timing comparison for some of the available intrinsic Fortran functions both on the Cyber 174 and the ASC in scalar and vector modes.

Table III
Math Library Functions*

		ASC	
	Cyber 174	Scalar	Vector
SQRT	48	14.9	1.5
SIN	90	17.2	1.7
EXP	91	15.4	1.6
LN		17.8	1.7

* For 1000 calculations, time per operation in μsec.

In order to achieve maximum vectorization of array operations there can be no interruptions in execution, such as a conditionality which excludes some members of an array from a particular operation. However, since conditionalities are often required, some means of incorporating them in a vectorizable manner must be found. To illustrate the effect of conditionality and vector length on the ASC consider the following Fortran:

```
(1)    DO 100 I = 1, N
(2)     IF(A(I) . GT. Z) A(I) = Z
(3) 100 C(I) = A(I) * B(I) + A(I)
```

In this form neither statement (2) or (3) is vectorized even though (3) is a potentially vectorizable line of code. By rewriting this code as two independent loops, vectorization will occur in the second loop with a substantial decrease in execution time. A much greater reduction in execution time can be achieved however, as total vectorization occurs by replacing statement (2) with a statement which utilizes the standard ASC vector instruction MIN instead of the IF test.

To show the effects of conditionality and vector length on execution speed, the three variations of this code shown in Figure 2 were executed using various vector lengths. Figure 2 shows that, for vector lengths less than five, the scalar code ran the fastest. For short vectors, vector overhead, in the form of time required for loading and executing the vector parameter file, overcomes any gain in efficiency realized by the vectorization. However, a dramatic decrease in execution time is apparent as vector lengths greater than five are used. For vectors of length 1000, the scalar code ran a factor of 16 slower than the fully vectorized code while the partially vectorized code ran a factor of 12 more slowly. It is particularly useful to note that the extra effort required to go from partial vectorization to full vectorization is worth it.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch004

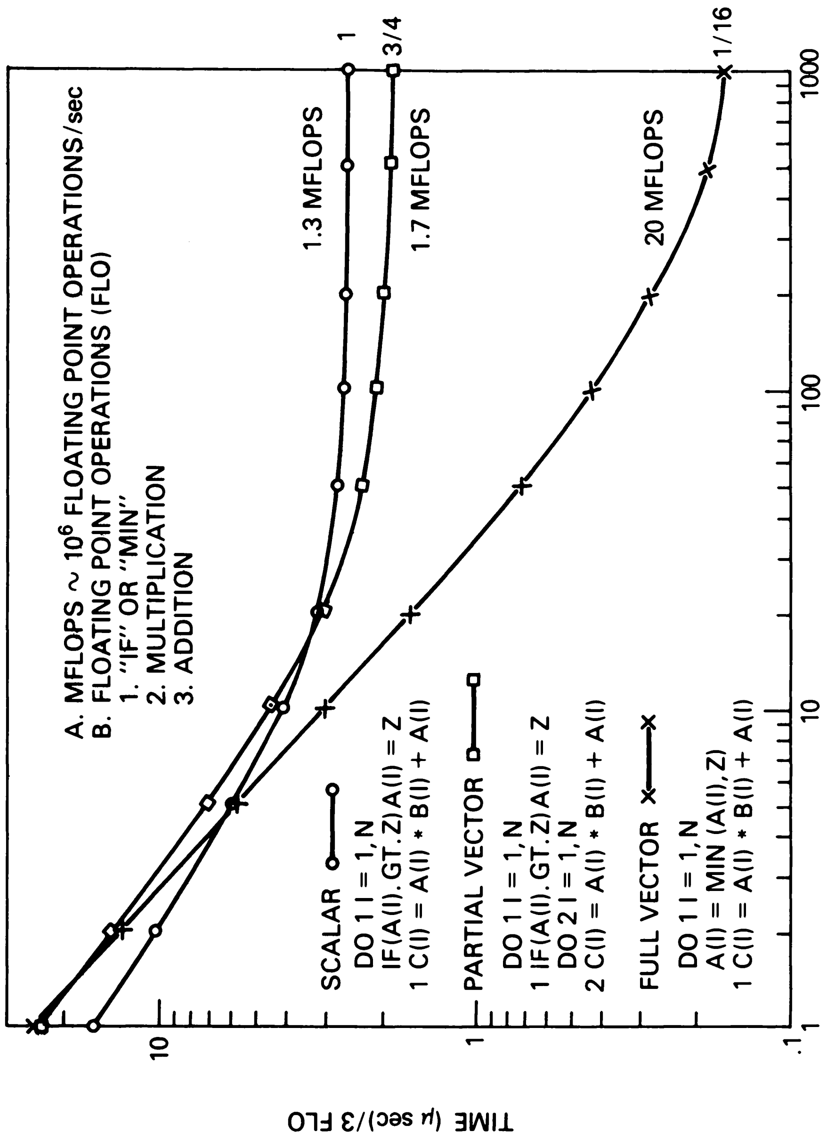


Figure 2. The effects of optimization and vector length on machine speed.

In Supercomputers in Chemistry; Lykos, P., el al.;
ACS Symposium Series; American Chemical Society: Washington, DC, 1981.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch004

Finally, most doubly or triply subscripted array operations can execute as a single vector instruction on the ASC. To demonstrate the hardware capabilities of the ASC, the vector dot product matrix multiplication instruction, which utilizes one of the most powerful pieces of hardware on the ASC, is compared to similar code on an IBM 360/91 and the CDC 7600 and Cyber 174. Table IV lists the Fortran pattern, which is recognized by the ASC compiler and collapsed into a single vector dot product instruction, the basic instructions required and the hardware speeds obtained when executing the same matrix operations on all four machines. Since many vector instructions in a CP pipe produce one result every clock cycle (80 nanoseconds), ordinary vector multiplications and additions (together) execute at the rate of 24 million floating point operations per second (MFLOPS). For the vector dot product instruction however, each output value produced represents a multiplication <u>and</u> an addition. Thus, vector dot product on the ASC attains a speed of 48 million floating point operations per second.

Table IV
Hardware Performance

<u>FORTRAN</u>

```
   DO 10 1 = 1, N
   DO 10 J = 1, N
   A (I,J) = 0.0
   DO 10 K = 1, N
10 A(I,J) = A(I,J) + B(K,I) * C(K,J)
```

<u>ASC</u> (2-PIPE)
(CPU/MEM - 80/160 nsec)

INSTRUCTION	CYCLES
VDPF	1

SPEED : 48 MFLOPS

<u>IBM</u> 360/91#
(CPU/MEM = 60/780 nsec)

INSTRUCTION	CYCLES
LOAD	1
MULTIPLY	3
ADD	2
LOOP	1

SPEED : 3 MFLOPS

<u>CDC</u> 7600#
(CPU/MEM = 27.5/275 nsec)

INSTRUCTION	CYCLES
LOAD	1
MULTIPLY	5
ADD	4
LOOP	0

SPEED : 4 MFLOPS

<u>CDC</u> CYBER 174
(CPU/MEM = 50/400 nsec)

INSTRUCTION	CYCLES
LOAD	21
MULTIPLY	58
ADD	11
LOOP	22

SPEED : .36 MFLOPS

Courtesy of N. Winsor

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch004

The Chemistry of Reactive Flow on the ASC

The Selected Asymptotic Integration Method (5) has been utilized for many years at NRL for the solution of the coupled "stiff" ordinary differential equations associated with reactive flow problems. This program has been optimized for the ASC.

The algorithm consists of the judicious application of one of two integration formulas to each equation in the system and the choice of formula is based on the time constant for each equation evaluated at the beginning of each chemical time step. Species with time constants too small are treated by the stiff method and the remaining species are treated by a classical second order method. The algorithm is characterized by a high degree of stability, moderate accuracy and low overhead which are very desirable features when applied to reactive flow calculations.

In reactive flow calculations, the transport and chemical reaction parts of the equations are separated by time step splitting techniques (6) and solved separately and sequentially for each transport time step. Therefore, when combined with transport, the choice of solution formula is made for each equation at each chemical time step at each mesh point for each hydro time step. The integrator, for stability, is allowed to reduce the time step independently at each mesh point to appropriate values less than the transport time step.

This integration method can be optimized for the ASC in two steps. The first is to construct the code so that vectorization over each set of equations occurs. Here the main problem is the decision process associated with the application of the "stiff" or "normal" formulas to each equation. If these formulas are implemented in the usual fashion with an IF test in the appropriate DO Loops the smooth flow of contiguous data from core through the CPU will be inhibited and scalar code will result. Optimization of this process can be accomplished by calculating both formulas and applying a multiplicative factor 0 or 1. The following example of Fortran code illustrates this technique.

```
   DO    10   I = 1, N
   D(I) = .5 + SIGN (.5, TAU - T (I))
   FN(I) = Normal Formula
   FS(I) = Stiff Formula
10 RESULT(I) = FN(I)*(1.0 - D (I)) + FS(I)*D(I)
```

The entries in the array D will have values of 0 or 1 depending on the values of T relative to TAU. The SIGN function, a standard Fortran function, applies the sign of the second argument to the absolute value of the first argument in vector form on the ASC. Even though the technique employs more operations than the scalar form, a significant improvement in efficiency is realized since both formulas are relatively uncomplicated and applied with nearly equal frequency. The increased efficiency with the application of this technique is generally limited in reactive flow problems since many of the reactive schemes employed have only a few species and the derivative function evaluator remains mostly scalar. Even though the short vectors may be efficient on the ASC the Vector start up overhead and especially the time associated with the scalar derivative function evaluator are major drawbacks.

The second step further improves the efficiency of the method by processing many independent sets of equations simultaneously. This becomes a problem of organization and bookkeeping but provides the opportunity for complete vectorization of the derivative function evaluator. Yet each set of equations at each grid location is advanced with its own independent time step even though many sets are treated simultaneously. At the end of each chemical step the integrator checks for sets that may have finished. The finished products are restored to their origin and sets of new initial conditions are procured in place of the finished set. Thus for reactive transport calculations large numbers of mesh points may be chemically advanced simultaneously and efficiently on the ASC.

Figure 3 compares the efficiency of the vector version VSAIM rate equation solver to the optimized but single mesh point integrator CHEMEQ. Improvement is obtained when only 5 sets are treated in parallel. Improvements of over a factor of six can be seen for long vectors.

It is often desirable to optimize code which by its nature can not be vectorized on the ASC. Chemical reaction rates are often complicated piece wise functions which can be very inefficient calculations. Appropriate values of these functions can be stored in tables in a fashion whereby the table location information can be calculated directly from the independent variable and the tabular information interpolated to give an accurate representation of the original function. The table location index array calculation for many independent but simultaneous evaluations of these functions can be fully vectorized. However, to complete the interpolation, code which involves random core references is generated which can not be vectorized. The ASC complier recognizes the situation and inserts software which significantly improves the scalar index fetch time over straight random core references. The result is a table look up algorithm which involves random core references and runs at nearly full vector speed. Values can be obtained at a rate of one every 2.0 μsec which compared favorably with ASC Vector Math Functions which average a result every 1.5 - 1.7 μsec.

The techniques just described have been extensively used in modeling reactive flow problems at NRL. Efficient solution of the coupled ordinary differential equations associated with these problems has enabled us to perform a wide variety of calculations on H_2/O_2 and CH_4/O_2 mixtures which have greatly extended our understanding of the combustion and detonation behavior of these systems. In addition numerous atmospheric problems have been studied. Details on these investigations are provided in references (7) and (9).

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch004

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch004

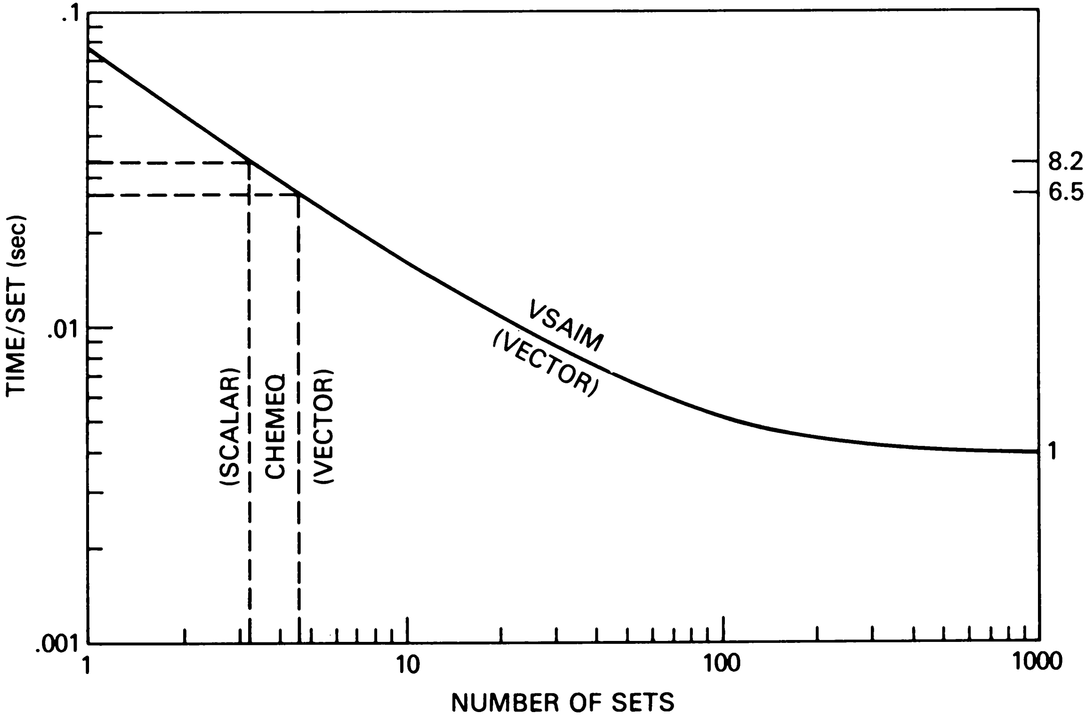


Figure 3. Ionospheric deionization timing test using 14 equations per set and 30 reactions per set.

Acknowledgements

The authors would like to acknowledge the material supplied by Drs. I. Haber and N. Winsor and thank them for many useful discussions. This work has been sponsored by the Naval Research Laboratory in part through the Office of Naval Research and in part by the Naval Material Command.

Liturature Cited

1. "Description of the ASC System (Part II), System Hardware", January 1975.
2. Brock, Harvey, Research Computation Division, Naval Research Laboratory. (private communication)
3. "Description of the ASC System (Part IV), The Operating System", January 1975.
4. H. Brock, B. Brooks, M. Miller, "Guide to Vectorization on the Naval Research Laboratory, Texas Instruments Advanced Scientific Computer", Vol. 1, NRL Memorandum Report 4102, Naval Research Laboratory, Washington, D.C.
5. T.R. Young, Jr., CHEMEQ - A Subroutine for Solving Stiff Ordinary Differential Equations, NRL Memorandum Report 4901, February 1980.
6. R.D. Richtmyer and K.W. Morton, Difference Methods for Initial Value Problems, 2nd Edition, Sec. 8.9, Interscience Publishers.
7. E. Oran and T.R. Young, "Numerical Modeling of Ionospheric Chemistry and Transport Processes" Journal of Physical Chemistry, 81, 2463, (1977).
8. E. Oran and J. Boris, "Detailed Modeling of Combustion Systems" NRL Memorandum Report 4371, November 1980.

RECEIVED August 19, 1981.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch004

5

The Use of a High-Speed Vector Processor Machine for Chemical Kinetic Sensitivity Analysis

DAVID EDELSON, LINDA C. KAUFMAN, and DANIEL D. WARNER[1]

Bell Laboratories, Murray Hill, NJ 07974

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch005

Over the past ten years the numerical simulation of the behavior of complex reaction systems has become a fairly routine procedure, and has been widely used in many areas of chemistry. [*1*] The most intensive application has been in environmental, atmospheric, and combustion science, where mechanisms often consisting of several hundred reactions are involved. Both deterministic (numerical solution of mass-action differential equations) and stochastic (Monte-Carlo) methods have been used. The former approach is by far the most popular, having been made possible by the development of efficient algorithms for the solution of the "stiff" ODE problem. Edelson has briefly reviewed these developments in a symposium volume which includes several papers on the mathematical techniques and their application. [*2*]

A desirable corollary to the simulation of a complex reaction system is the study of the dependence of this behavior on the parameters of the assumed model. This "sensitivity analysis" is yet an order-of-magnitude larger problem than the simulation itself, and has until recently been unthinkable for large mechanisms where just the solution of the kinetic equations taxed the resources of the most advanced machines. However the recent appearance of improved algorithms together with the availability of high speed vector machines with extensive core storage have cast this problem in a new light. In this paper we explore the impact of these developments on the advancement of this computation. Our results are most encouraging and indicate that sensitivity analysis will, within the near future, become as commonplace as mechanistic simulation already has.

1. MATHEMATICAL BACKGROUND

The ODE problem posed by the kinetics of a chemical mechanism of r reactions in n species may be written as the usual mass action product

$$\frac{dn_i}{dt} = \sum_{j=1}^{r} \nu_{ij}\alpha_j \prod_{l=1}^{n} n_l^{\nu_{lj}} = f_i(\mathbf{n}, \alpha, t)\ ;\ i=1,2,\ldots n \qquad (1)$$

where the n's are the species concentrations, the α's are the reaction rate

---

[1] Current address: Clemson University, Department of Mathematical Sciences, Clemson, SC 29631.

0097-6156/81/0173-0079$05.00/0
© 1981 American Chemical Society

coefficients (which might be further decomposed into several parameters, e.g. the Arrhenius form) and the ν's are the molecularities. In the related sensitivity analysis it is desired to determine the first-order dependence of the solution of (1) upon the parameters α, i.e. $\partial n_i/\partial \alpha_j$. The formulation of this problem may be derived from (1) by differentiating with respect to the α's and then changing the order of differentiation under the assumption that the functions $n(t)$ and their derivatives are continuous:

$$\frac{\partial \mathbf{n}}{\partial \alpha_j} \equiv \mathbf{n}_{\alpha_j}\ ;\ j=1,2,\ldots r$$

$$\frac{d\mathbf{n}_{\alpha_j}}{dt} = \frac{d}{dt}\frac{\partial \mathbf{n}}{\partial \alpha_j} = \frac{\partial}{\partial \alpha_j}\frac{d\mathbf{n}}{dt}$$

$$= \frac{\partial \dot{\mathbf{n}}}{\partial \alpha_j} = \frac{\partial}{\partial \alpha_j}\mathbf{f}(\mathbf{n},\alpha,t)$$

$$= \frac{\partial \mathbf{f}}{\partial \mathbf{n}}\frac{\partial \mathbf{n}}{\partial \alpha_j} + \frac{\partial \mathbf{f}}{\partial \alpha_j}$$

$$\dot{\mathbf{n}}_{\alpha_j}(t) = \mathbf{J}(t)\mathbf{n}_{\alpha_j}(t) + \mathbf{f}_{\alpha_j}(t)\ ;\ j=1,2,\ldots r \qquad (2)$$

where the matrix

$$\mathbf{J} = |J_{ij}| = \left|\frac{\partial \dot{n}_i}{\partial n_j}\right|$$

is the $n \times n$ Jacobian of Eq. (1), and the subscript α_j denotes differentiation with respect to parameter α_j. For a mechanism of r reactions involving n species, Eqs. (1) and (2) comprise a problem of $n(r+1)$ simultaneous differential equations.

The direct method (DM) for solution of this set of equations was proposed by Atherton *et al.* [*3*], and in a somewhat a modified form by Dickinson and Gelinas [*4*] who solved r sets of equations each of size $2n$ consisting of Eq. (1) coupled with a particular j—value of Eq. (2). Shuler and coworkers [*5*] took an alternative approach in the Fourier Amplitude method in which a characteristic periodic variation is ascribed to each α, and the resulting solution of (1) is Fourier analyzed for the component frequencies. These authors estimate that $1.2r^{2.5}$ solutions of Eq. (1) together with the appropriate Fourier analyses are required for the complete determination of the problem. Since even a modest reaction mechanism (e.g. in atmospheric chemistry or hydrocarbon cracking or oxidation) may easily involve 100 reactions with several tens of species, it is seen that a formidable amount of computation can result.

A somewhat more economical approach to this problem was devised by Rabitz and his coworkers [*6*, *7*, *8*] who solved Eq. (2) through the use of the associated Green's function (GF Method)

$$\mathbf{n}_{\alpha_j}(t) = \int_0^\infty d\tau\ \mathbf{K}(t,\tau)\mathbf{f}_{\alpha_j}(\tau) \qquad (3)$$

where the kernel $\mathbf{K}$, the Green's function, is obtained from the solution of the differential equation

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch005

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch005

$$\dot{\mathbf{K}}(t,\tau) - \mathbf{J}(t)\,\mathbf{K}(t,\tau) = 0 \qquad (4)$$

with initial conditions

$$\mathbf{K}(t,t) = \mathbf{I} \qquad (4a)$$

This is an $n \times n$ matrix of ODEs; however since n is usually considerably less than r, the problem is substantially smaller than the DM above. Furthermore, the columns of **K** are independent of each other, so that the computation may be further reduced to the solution of n sets of n ODEs. The integrals represented by Eq. (3) may be evaluated while the ODE solution is progressing by some simple means such as the trapezoidal rule, and thus represent an insignificant part of the computational effort.

Several extensions to the sensitivity analysis have been made, e.g. sensitivities to initial conditions, spatial parameters in reactive flow [*9*], higher order and derived sensitivities [*7*], but for the purpose of this study only the linear sensitivity problem with respect to the rate parameters will be considered.

2. COMPUTATIONAL DETAILS

The reaction mechanism used for this study was the alkane pyrolysis scheme of Edelson and Allara. [*10*] This consists of 98 reactions involving 38 chemical species. It is not so large that it overburdens the computers used; yet is of a size sufficient to yield meaningful timing of the program modules.

Implementation of the calculation followed in general the scheme of Dougherty et al. [*8*] The kinetic problem Eq. (1) was solved separately using the BELLCHEM code consisting of a chemical compiler followed by a stiff ODE solver using the method of Gear. [*11*] Results were stored on disc for subsequent input to the sensitivity calculation.

The solution of the matrix of ODEs of Eq. (4) was first performed one column at a time (hereafter referred to as Scheme I) using the implicit midpoint rule with extrapolation. [*12*] This method was chosen because of its ability to cope with extremely stiff problems. It was coded in a highly modular fashion using basic tools from the Bell Laboratories PORT Library [*13*], thus allowing detailed timing of every phase of the computation. The **J** matrix was calculated as needed with the same machine-compiled routine used by BELLCHEM in the solution of Eq. (1). Required values of $\mathbf{n}(t)$ were retrieved from disc and linearly interpolated between time points. The Jacobian required for the solution Eq. (4) is $\mathbf{J}^T$ and is readily obtained. The integrals Eq. (3) were computed by intercepting the ODE solution at every time point and applying the trapezoidal rule.

Dougherty and Rabitz [*8*] point out that for many applications it is not necessary to compute the entire sensitivity matrix, but only those columns for species considered to be of interest, such as those susceptible to measurement. There are however, certain advantages to computing the entire Green's Function matrix, principally the ability of time scaling in cases where sensitivities are required at several points in time. [*6*] For the purpose of this paper, the entire matrix was computed.

An alternative to the serial column-by-column computation is the computation of the entire **K** matrix as a unit (Scheme II). This would seem to afford

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch005

economies in computation time (at the expense of a somewhat larger core requirement) based on the following considerations. In Scheme I, assuming the time mesh for the ODE solution were the same for each column, then the identical Jacobian matrix is repeatedly decomposed for each column at each time step. By computing **K** as a unit the decomposition need only be done once at each step. Depending on the order selected by the step size and order monitor, we would expect about 2 to 3 decompositions per step and about 12 to 24 solves; *i.e.* we expect 6 to 8 solves for every decomposition. In fact the actual ratio for the results reported is about 7. Let $S_I n$ denote the total number of steps used by Scheme I and let S_{II} denote the total number of steps for Scheme II. The amount of work for a single decomposition is $O(n^3/3)$ while it is $O(n^2)$ for a solve per column. The total work should be $O(S_I n[n^3/3 + mn^2])$ for Scheme I and $O(S_{II}[n^3/3 + mn^3])$ for Scheme II, where m is the solve/decomposition ratio. If $S_I \approx S_{II}$ then the linear algebra part of Scheme II should be about 2.7 times faster than Scheme I for $n=38$.

The computation was originally implemented on a Honeywell 6080 with a 2K high speed cache, and subsequently transferred to a Cray-1. Honeywell computations were done in double precision (18 decimal digits) while single precision was employed on the Cray (14 decimal digits). The number of significant bits in the mantissa of a floating point number (63 vs 48 respectively) are sufficient for the computation on either machine; Honeywell single-precision (27 bits) would not suffice. The effect of the somewhat lower precision on the Cray might be to introduce some noise into the solution and require a few more steps to be taken. Arithmetic operations on both machines are implemented in hardware; relevant parameters are given in Table 1.

TABLE 1. COMPARISON OF HARDWARE FEATURES

	Honeywell	**Cray**
Word Size	36 bits	64 bits
Available Memory	256K	1M
Cache/Buffer	2K	384
Timing, μs:		
Clock Cycle	0.50	0.0125
Memory Cycle	0.50	0.0500
Addition		
Floating s.p.	1.70	0.0750
Floating d.p.	1.70	
Multiplication		
Floating s.p.	3.10	0.0875
Floating d.p.	6.20	

3. RESULTS

Timing studies performed on the Scheme I program are summarized in Table 2. The Cray program was run twice, with the compiler vector option turned off for one trial to assess separately the effect of the increased speed of the computer and the effect of vectorizing the arithmetic. The subroutines that consumed most of the processor time (function computation, and decomposition and solution of linear equations) showed a 30-50 times improvement due only to the increased speed of the machine. A further improvement of up to a factor of 5 was realized by turning on the vector processor; this is highly dependent on the specific computation as well as coding details. Overall improvement in running time for the entire computation was a factor of about 80. Cost effectiveness improvement was estimated at a factor of 20.

Scheme II was run on both machines, giving the results shown in Table 3. Disappointingly the expected economies failed to materialize. Although the linear algebra work was reduced, S_I, the average number of steps for Scheme I, was 42.3; while S_{II}, the number of steps for Scheme II, turned out to be 90. This doubling of the number of steps offset the gain in the linear algebra and so magnified the integration overhead that Scheme II actually ran slower than Scheme I. Note that the size of the time step required in the integration of the ODEs is determined by the most rapidly varying component in the solution at each time. Since Scheme II has 38 times as many components as Scheme I, the large increase in the number of steps is accounted for by the necessity of accommodating the worst case, even though the other components do not require such accuracy.

The subroutines accounting for the major part of the processor time were then scrutinized in detail to see whether further optimization was possible. The function routine, for example, spends most of its time computing a matrix-vector (Scheme I) or matrix-matrix product (Scheme II). In both cases, the Cray code is compiled as a sequence of vector-vector products (i.e. only the inner loop is vectorized). Scheme I affords limited potential for further improvement. In Scheme II, since the solution phase involves multiple right-hand sides, the solve subroutine could be reformulated to take advantage of the assembly language matrix-vector multiplication subroutine, in which higher levels of loops are vectorized. Similarly, the function subroutine could be optimized. The effects of successive implementation of these changes are summarized in Table 4. As a result, these operations no longer consume the preponderance of processor time, and other modules in the integrator become candidates for further optimization. The subroutine which applies the implicit mid-point rule was speeded up by a factor of about 10 by a minor rewrite of a portion of the code in vectorizable form. The error test routine similarly was improved by more than a factor of 2. The remaining routines consuming substantial processor time were associated with the extrapolation and the error estimation in the ODE solver. These could not be further improved without a major restructuring, and were not modified at this time.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch005

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch005

TABLE 2. TIMING OF SENSITIVITY ANALYSIS COMPUTATION

38 Species, 98 Reaction Pyrolysis Mechanism

SCHEME I

	HONEYWELL 6080			CRAY-1				
					non-vector		vector	
	No. Calls	av. ms.	%	No. Calls	av. ms.	%	av. ms.	%
Function	52658	29.00	25.87	57446	0.96	32.28	0.19	15.04
LU Decomp.	7538	158.88	20.29	8345	3.43	16.68	2.06	23.15
Solve	54339	34.50	31.76	59302	.80	27.70	.29	23.48
Total, sec.		5902.79			171.82		74.32	

In Supercomputers in Chemistry; Lykos, P., el al.;
ACS Symposium Series; American Chemical Society: Washington, DC, 1981.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch005

TABLE 3. TIMING OF SENSITIVITY ANALYSIS COMPUTATION

SCHEME II

	HONEYWELL 6080			CRAY-1 (vector)		
	No. Calls	av. ms.	%	No. Calls	av. ms.	%
Function	3679	1015.49	43.59	4200	5.40	19.68
LU Decomp.	477	71.41	0.40	570	1.95	0.96
Solve	3679	932.99	40.05	4200	9.35	34.06
Total, sec.		8571.65			115.35	

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch005

TABLE 4. TIMING OF SENSITIVITY ANALYSIS COMPUTATION

38 Species, 98 Reaction Pyrolysis Mechanism

SCHEME II, CRAY-1, Optimized

	+ Optimized Function			+ Optimized Solve			+ Optimized Step, Error Test		
	Calls	av. ms.	%	Calls	av. ms.	%	Calls	av. ms.	%
Function	4456	1.00	4.32	4728	1.02	6.57	4728	1.02	10.20
LU Decomp.	628	1.95	1.19	671	1.96	1.79	671	1.96	2.78
Solve	4456	9.35	40.55	4728	1.57	10.12	4728	1.57	15.73
Mid-point Step				2276	8.29	25.69	2276	0.78	3.78
Error Test				5079	3.11	21.50	5079	1.31	14.13
Total, sec.		102.81			73.48			47.26	

In Supercomputers in Chemistry; Lykos, P., el al.;
ACS Symposium Series; American Chemical Society: Washington, DC, 1981.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch005

4. DISCUSSION

As a result of the transfer of the GF Method of sensitivity analysis to the vector machine, an improvement of more than a factor of 100 in running time has been achieved, with an associated cost effectiveness of about 60 from Scheme I on the Honeywell to Scheme II on the Cray. This has been accomplished not only by virtue of the use of a higher speed machine and the vector processor, but also by making the proper choice among alternative algorithms and paying close attention to coding details.

These experiments provide a number of highly instructive lessons. Chief among them is the necessity to explore all programming alternatives for the optimum. In the present example Scheme II appeared at the outset to offer substantial economies over Scheme I. Indeed it is more efficient per step, and it was not until the initial trials were completed on the Honeywell that it became apparent that this gain was offset by its need of more than double the number of steps to complete the problem. However, the ability to vectorize more effectively offset this loss and Scheme II ultimately turned out to be superior. In effect, the increase by a factor of 2 in the number of operations required has been overwhelmed by the gain in computational speed afforded by extensive vectorization, *e.g.*, a factor of 7 for a single-precision multiply.

It must be pointed out these considerations are highly dependent on the nature of the computation. In the particular problem reported here the extent of array processing is the overriding feature that makes these economies possible. The size of the arrays in relation to the hardware is also important. Experience has shown that it takes the equivalent of about four executions of a DO-loop to set up the array processor, and that the maximum number of array elements which can be processed without incurring additional overhead is 64.

Program optimization is facilitated by breaking up the code into small subroutines, and using a timer to pinpoint those modules in which most of the processor time is spent. Fortunately the Cray CFT Fortran system is already equipped with a flow trace and timing system, and the PORT subroutine library has been written in a highly modular form in keeping with the "software tools" concept. [*14*] In a large program the subroutine linkage overhead is trivial (less than 0.1% in this problem) and the few seconds consumed by the timing measurement is well worth the expenditure.

In summary, sensitivity analysis, which has heretofore been considered a very large and costly calculation, has been reduced to the point where it may be done as a matter of routine, and even incorporated into other computations that rely on sensitivity values, as for example non-linear parameter estimation.

LITERATURE CITED

1. Edelson, D., *J. Chem. Ed.* 1975, **52,** 642
2. Edelson, D., *J. Phys. Chem.*, 1977, **81,** 2309
3. Atherton, R. W., Schainker, R. B., and Ducot, E. R., *AIChE Journ.* 1975, **21,** 441
4. Dickinson, R. P., and Gelinas, R. J., *J. Comp. Phys.* 1976, **21,** 123
5. Cukier, R. I., Levine, H. B., and Shuler, K. E., *J. Comp. Phys.* 1978, **26,** 1, and references cited therein.

6. Hwang, J.-T., Dougherty, E. P., Rabitz, S., and Rabitz, H., *J. Chem. Phys.* 1978, **69,** 5180
7. Dougherty, E., Hwang, J.-T., and Rabitz, H., *J. Chem. Phys.* 1979, **71,** 1794
8. Dougherty, E., and Rabitz, H., *Int. J. Chem. Kinet.* 1979, **11,** 1237
9. Koda, M., Dogru, A. H., and Seinfeld, J. H., *J. Comp. Phys.*, 1979, **30,** 259
10. Edelson, D., and Allara, D. L., *Int. J. Chem. Kinet.*, 1980, **12,** 605
11. Edelson, D., *Computers & Chemistry,* 1976, **1,** 29
12. Lindberg, B., in *Stiff Differential Systems* (R. Willoughby, ed.), Plenum Press, 1974, pp 201-215.
13. Fox, P. A., Hall, A. D., and Schryer, N. L., *ACM Trans. Math. Software* 1978, **4,** 104-126
14. Kernighan, B. W., and Plauger, P. J., *Software Tools,* Addison, New York, 1976.

RECEIVED July 27, 1981.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch005

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch006

The Effect of Buoyancy on the Thermal Ignition of Carbon Monoxide

WALTER W. JONES[1]

Naval Research Laboratory, Washington, DC 20375

In this paper, we report on a calculation which shows the effect of buoyancy on thermal ignition of a homogeneous mixture. The intent of this was to start with a useful calculation, which could not be done using brute force techniques, and demonstrate the importance of optimizing the numerical implementation of a reactive flow model to run on a vector computer. As similar problems in combustion become more extensive and intricate, it behooves us to utilize computers in the most efficient manner possible. It is no longer feasible to continue to "ask the computer" to do more and more work, without thought as to how a particular problem is to be implemented. The number of problems for which one would like to use a computer, as well as the complexity of these problems, is increasing at an astronomical rate. The other side of the coin, of course is that computers, and especially central processors (CPU's) are becoming cheaper.

Two big advances over the last fifteen years have been the ability of CPU's to pipeline instructions, that is work on several instructions simultaneously, and the ability to use one instruction to operate on a large collection of data. This latter feature is referred to as the "vector mode" and will be the primary emphasis of this paper.

The vector instruction is useful since it allows a CPU to do one operation on a large set of data; previously each separate set of data (two operands) required decoding an instruction and then fetching the data finally performing the required operation. Even in the pipeline mode, it has not been possible to achieve the theoretical efficiency of a computer, namely, one result per machine cycle, without the vector capability. Basically, the vector instruction tells the CPU where itwill have to go to get data, and how much data will be required.

---

[1] Current address: National Bureau of Standards, Washington, DC 20234.

This chapter not subject to U.S. copyright.
Published 1981 American Chemical Society

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch006

Traditionally the vector feature has been the sole property of large scale supercomputers. With the advent of array processors, however, this important feature is becoming available to minicomputer users also.

The general problem has been to extend the usefulness of the induction parameter model proposed by Oran et al. (1). This induction parameter model (IPM) is proposed as a means to enable one to estimate, relatively easily, the energy necessary to achieve ignition when using a thermal heating source. Much of the calibration of this model, for example the effect of deposition volume (quench volume), can be done with one-dimensional models, and shock tube experiments. There are phenomena, however, which must be studied in two or three dimensions. Examples are turbulence and buoyancy. This paper discusses the effect of buoyancy and possible extensions to the IPM.

The IPM is a simple application of the slow-flow approximation to the pressure equation (2)

$$\frac{dp}{dt} \approx 0 = - \gamma P \nabla . \bar{V} + \nabla . \gamma N k_B \kappa \nabla T + (\gamma - 1)\frac{\partial \varepsilon}{\partial t} \qquad (1)$$

and yields an algebraic equation for the divergence of velocity. This combined with the mass conservation equation

$$\frac{1}{\rho}\frac{d\rho}{dt} = - \nabla . \vec{V} \qquad (2)$$

and an equation of state, in this case we assume an ideal gas, yields

$$\frac{1}{T}\frac{dT}{dt} = \frac{(\gamma-1)\ \partial\varepsilon/\partial t}{P_\infty} + \frac{\nabla \cdot K \nabla T}{T} \qquad (3)$$

where "P_∞" is the back ground pressure. "T" is the temperature, "γ" the ratio of specific heats of the gas C_p/C_v, and $K \equiv \frac{\gamma-1}{\lambda_m N k_B}$ where λ_m is the mixture thermal conductivity. An assumption that $\partial\varepsilon/\partial t$ is deposited according to a Gaussian profile allows for a closed form for the solution of this equation. If

$$\frac{\partial \varepsilon}{\partial t} = A(t)\ e^{-k^2(t)\ r^2}, \qquad (4)$$

then we have two simultaneous equations for "A" and "k"

$$\frac{dk}{dt} = - kV - 2\ \kappa\ k^3 \qquad (5)$$

$$\frac{dA}{dt} = \frac{S}{\gamma P_\infty} - 6\ \kappa\ k^2 A\ . \qquad (6)$$

Equation (3) then yields for the temperature

$$T\ (r,t) = T_\infty \exp\ (-A(t) \exp\ [-k^2(t)r^2]). \tag{7}$$

This allows us to define an induction parameter

$$I = \int_0^t \frac{dt'}{\tau(T(r,t'))} \tag{8}$$

where τ is the induction time for the kinetics scheme being used. A plot of this induction time for our carbon-monoxide scheme (discussed in the next section) is shown in figure (1). A simple fit to the data can be made using

$$\tau(T) = \tau_* \exp\left(\frac{19775}{T-T_*}\right)$$

In the figure, the circles represent the measured data, and the circles which are not filled in were the points used to find τ_* and T_*. The IMP then predicts the energy necessary for ignition by finding the energy to be deposited such that $I \rightarrow 1$.

As can be seen from equation (3 - 7) there is no mechanism to allow for the effects of buoyancy. In section (V) we will discuss a possible extension.

Technical Aspects of the Two Dimensional Calculation.

The equations to be solved are (2):

$$\frac{dp}{dt} = -\ \nabla \cdot \vec{V} \tag{9}$$

$$\frac{d\vec{\S}}{dt} = \nabla x\ (\frac{\nabla P}{\rho}) - \vec{\S}\nabla\cdot\vec{V} + \nabla x\ (\frac{\nabla\cdot\nu\nabla\vec{V}}{\rho}) \tag{10}$$

$$\begin{aligned}\frac{d}{dt} D = -\ \nabla\cdot(\frac{\nabla P}{\rho}) + (\S^2 + \vec{V}\cdot\nabla^2\vec{V} - 1/2\nabla^2\vec{V}^2)\\ + \nabla\cdot\vec{g} + \nabla\cdot(\frac{\nabla\cdot\nu\nabla\vec{V}}{\rho})\end{aligned} \tag{11}$$

$$\begin{aligned}&\frac{d\varepsilon}{dt} + \varepsilon D = -\ \nabla\cdot P\vec{V} + \nabla\cdot\kappa\nabla T + \frac{\partial\varepsilon}{dt}\Big|_{chem} + \sum_i h_i\rho_i\vec{V}_i'\\ &\text{where } \varepsilon \equiv 1/2\rho\vec{V}^2 + \frac{P}{\gamma-1} + \rho\phi \equiv \text{total energy}\end{aligned} \tag{12}$$

$$\frac{d\rho_i}{dt} = -\rho_i\nabla\cdot\vec{V} - \nabla\cdot\rho_i\vec{V}_i + \frac{\partial\rho_i}{\partial t}\Big|_{chem} \tag{13}$$

and

$$D \equiv \nabla \cdot \vec{V}$$

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch006

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch006

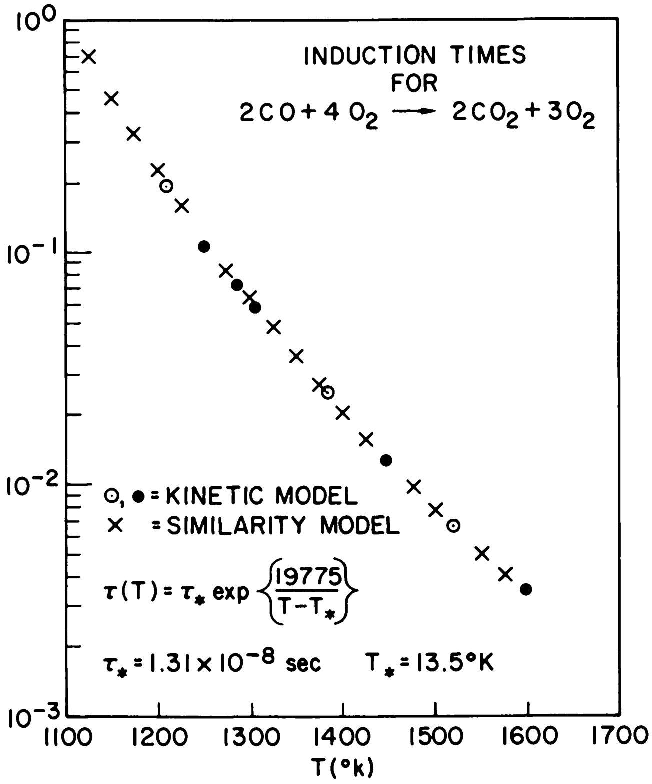


Figure 1. Induction time for the CO + 2O₂ mixture as a function of initial temperature.

This is a very general formulation. The symbols used are:

ρ = total mass density (gm/cm^3)

n = total particle density (#/cm^3)

ρ_i = mass density of the ith species (gm/cm^3)

n_i = number density of the ith species (#/cm^3)

$\vec{V}$ = bulk fluid velocity (cm/sec)

$\vec{\xi}$ = $\nabla \times \vec{V}$ vorticity (sec^{-1})

D = $\nabla . \vec{V}$ divergence (sec^{-1})

g = gravity (- 980 cm/sec^2) - negative gravity points downward

γ = C_p/C_v = ratio of specific heats of the fluid

κ = thermal conductivity coefficient (erg/cm - sec - K)

h_i = enthalpy of the ith species (ergs/molecule)

$\vec{V}'_i$ = diffusion velocity of the ith species (cm/sec)

ν = viscosity coefficient (gm/cm/sec)

P = pressure (dynes/cm^2)

V = volume of the cell (cm^3)

D_{ij} = binary diffusion coefficient (i $\leftrightarrow$ j, cm^{2}sec)

S_i = diffusion source term for the ith species (cm^{-1})

We need two additional equations to close this set. We use the ideal gas law $P = nkT$ to give us a relation between pressure, density and temperature, and

$$S_i = \sum_j \frac{n_i n_j}{n^2 D_{ij}} (\vec{V}'_j - \vec{V}'_i) \qquad (14)$$

to relate the diffusion velocities to the other physical parameters. The quantities S_i and D_{ij} are discussed in detail later.

Buoyancy is a major effect in most diffusion flames and even in premixed systems it can be responsible for much of the flame dynamics. The computational problem introduced here is one of

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch006

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch006

widely disparate fluid dynamic timescales. The slow flow algorithm used for the temporal integration of the fluid dynamics equation is a means to filter sound waves out of the equations so that timesteps much longer than Courant step $\delta t \sim \delta x/C_s$ can be taken without numerical instability. The standard approach of formulating an implicit pressure equation requires at least a formal substitution of one equation, in finite difference form, into another and usually some form of additional numerical smoothing. The resulting algorithms do not resolve step gradients well during convection. Thus we use the flux-corrected-transport technique (3) for the convection. However, this technique is inherently nonlinear and hence does not lend itself to an implicit formulation. Therefore we use the slow flow algorithm which is asymptotic rather than implicit in concept, and which allows the unlimited use of the FCT algorithm.

Equation (9), (10) and (13) are solved as is using the FCT technique. However, equations (11) and (12) are combined, together with the assumption $\frac{dP}{dt} \approx 0$, to form

$$\gamma P \nabla\cdot\vec{V} = -\vec{V}\cdot\nabla P + \left(\frac{\partial P}{\partial t}\Big|_{chem} - \left\langle \frac{\partial P}{\partial t} \right\rangle_{av}\right)$$

$$+ (\gamma-1)\ (\nabla\cdot\kappa\nabla T - \vec{V}\cdot(\nabla\cdot\nu\nabla\vec{V}) + \sum_i h_i p_i V_i' \qquad (15)$$

where $$\left\langle \frac{\partial P}{\partial t} \right\rangle_{av} \equiv \frac{1}{V}\int dv\ \{\gamma-1\}\ \frac{\partial \varepsilon}{\partial t}\Big|_{chem} \qquad (16)$$

and finally

$$P_{new} = P_{old} + \delta t \left\langle \frac{\partial P}{\partial t} \right\rangle_{av} \qquad (17)$$

Such an asymptotic formulation embodies the assumption that waves traveling at the sound speed do not influence the overall hydrodynamics motion, nor change the chemical kinetics. This, of course, restricts us to flows where $V_f < C_s$. These equations are solved on the staggered grid shown in figure (2). The reasons for using a staggered grid is that it allows one to implement the differential equations in a conservative finite difference form (2), while maintaining simple, physical boundary conditions. In addition, such a choice of grid system allows one to specify open or closed boundary conditions using either the equation for the divergence of velocity (11) or the vorticity (10). In either case, the fluid flux is the quantity specified, and it is important to insure that the boundary conditions are self-consistent. If the velocity is written in terms of a stream function and a velocity potential

$$V = \nabla\phi + \nabla x\vec{\Psi} \qquad (18)$$

then one obtains Poisson equations for these two variables,

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch006

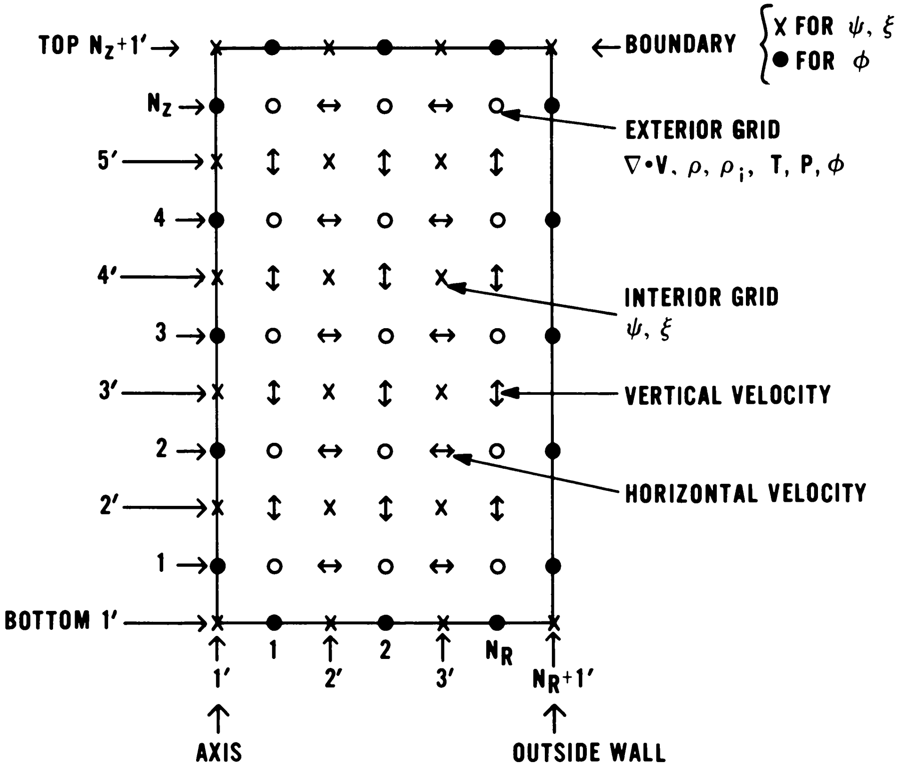


Figure 2. Grid system used in the two-dimensional model.

$$\nabla \times V = \vec{\xi} = \nabla \times \nabla \times \vec{\psi} \tag{19a}$$

and

$$\nabla . V = D = \nabla^2 \phi \tag{19b}$$

Inflow or outflow can be specified using appropriate boundary conditions, although some characteristic problems are more easily implemented with one than the other. In any case, it is important to insure that the conditions specified are self-consistent. For example, inflow can be specified along the top boundary, by specifying the inflow velocity at V_2, as in figure (3). This in turn specifies a relative change in vorticity from ξ_1 to ξ_2. If this is not to introduce net vorticity into the system (which is implicit in equation (10) unless there is also a net divergence present here) then we must require

$$\int_s \vec{\xi} \cdot dA = 0 = \oint \nabla \times \vec{\Psi} \cdot dl$$

The above example of specifying V_2, and consequently a change from ξ_1 to ξ_2, requires a concomitant reversal in some other region of the ξ boundary and thus outflow. As can be seen, such boundary conditions must be implemented with some care if the numerical scheme is to be self-consistent.

Techniques for Vectorization

The class of problem which we are presenting here can not be done with a reasonable amount of computer time, using brute force techniques. It is necessary, for example, to solve a two-dimensional problem rather than a three dimensional problem. Another example is the problem of molecular diffusion (4,5,6). The usual binary diffusion approximation is inadequate. At the very least total mass is transported in this approximation. The primary difficulty in a general treatment is that of inverting a large ($N_{species} \times N_{species}$) matrix at each time step and for each grid point. A technique (4) has been developed which avoids this problem. Essentially, the technique hinges on a perturbation expansion for the diffusion velocity in terms of the forcing function S, equation (14). The form of the solution for the velocities is

$$\vec{V}'_i = - \left(\frac{\rho - \rho_i}{\rho}\right) \frac{n^2 D_i}{(n-n_i)n_i} \quad (1 + \bar{A} + \overline{AA} + \dots.) \tag{20}$$

where D_i is defined in terms of the binary diffusion velocities (2,5,6)

$$D_i \equiv (n-n_i)/\sum_{j \neq i} \left(n_i/D_{ij}\right) \tag{21}$$

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch006

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch006

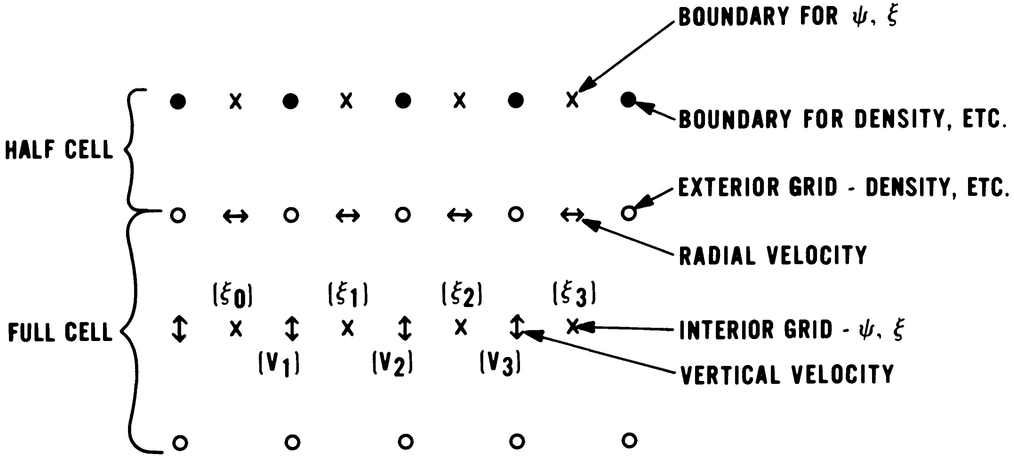


Figure 3. Boundary grid used in the two-dimensional model. This illustrates the interaction between velocity and vorticity in setting boundary conditions.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch006

and
$$A_{jk} \equiv \frac{\rho i}{\rho} \delta_{jk} + \frac{n_i}{D_{jk}} \frac{(\rho-\rho_k)}{\rho} \frac{D_k}{(n-n_k)} (1-\delta_{jk}) \qquad (22)$$

In addition to reducing the computing time by order ($N_{species}$), this form lends itself to explicit vectorization over the grid. Also, of course, fewer calculation will reduce the numerical error. Figure (4) shows the time required to compute the diffusion fluxes for various size grids. In all cases, this is the time from entering the subroutine to exit. The results are given for a five species carbon-monoxide chemistry scheme. The abcissa reflects the number of terms used the perturbation expansion. We have found that in most pratical cases, only three terms (n = 2) are required. This usually reduces the error to less than 5%, which makes the numerical results as accurate as the physical quantities (D_{ij}) used in computing the diffusion fluxes, and in the original derivation of this form of the diffusion equation (5,7). In addition, for vector lengths of ten (10) or larger and for five or more species, this form of finding the diffusion flux term ($n_i\vec{V}_i'$) is as fast as doing the usual differencing for the binary approximation ($\nabla \cdot D_i \nabla \rho_i$).

To show the effect of using long vectors (which reduces the required vector overhead) we have compared the timings for a calculation involving four hundred grid points (20 x 20) and five species. As can be seen from figure (5) even vectors of length ten (ten grid points by five species) yields a great deal of improvement over the same code run in scaler mode. The curve shown as (+) is the ratio of time required for the calculation with the vector length shown on the abcissa (n = 10, 20...) divided by the corresponding time for a scaler code (n = 0). The curve indicated by (0) compares the time for vectors of various lengths with that for a calculation using a vector of length ten (10). In all cases, the vector length refers to the number of grid points and the five species are folded into the calculation. For lengths of 100 or larger, the calculation runs at about 20 MFLOPS/sec which is only slightly slower than the machine speed of 25 MFLOPS/SEC and is due to some scaler code inherent in "IF" tests.

Results

The calculations presented here are intended to show the effect of buoyancy on the ignition properties of a homogeneous fuel-oxydizer mixture, in this case carbon monoxide-oxygen. The experimental cell chosen was 1.2 cm in height, and 1.6 cm in diameter. The grid system was 40 x 40 for the main grid and 39 x 39 for the grid used to carry the time histories $\vec{\psi}$ and $\vec{\xi}$. The initial species composition was $CO + 2O_2$.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch006

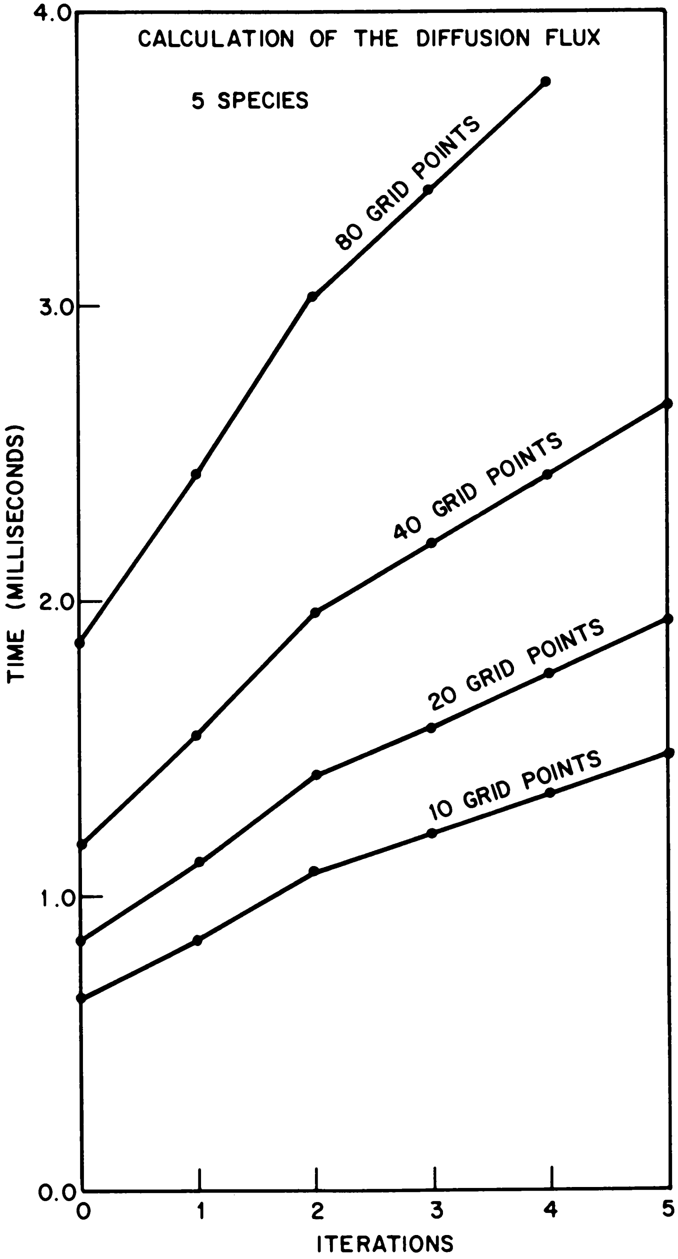


Figure 4. Total time required to compute the diffusion fluxes for grids of 10, 20, 40, and 80 points. These times are obtained using a vectorized multispecies diffusion algorithm (4).

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch006

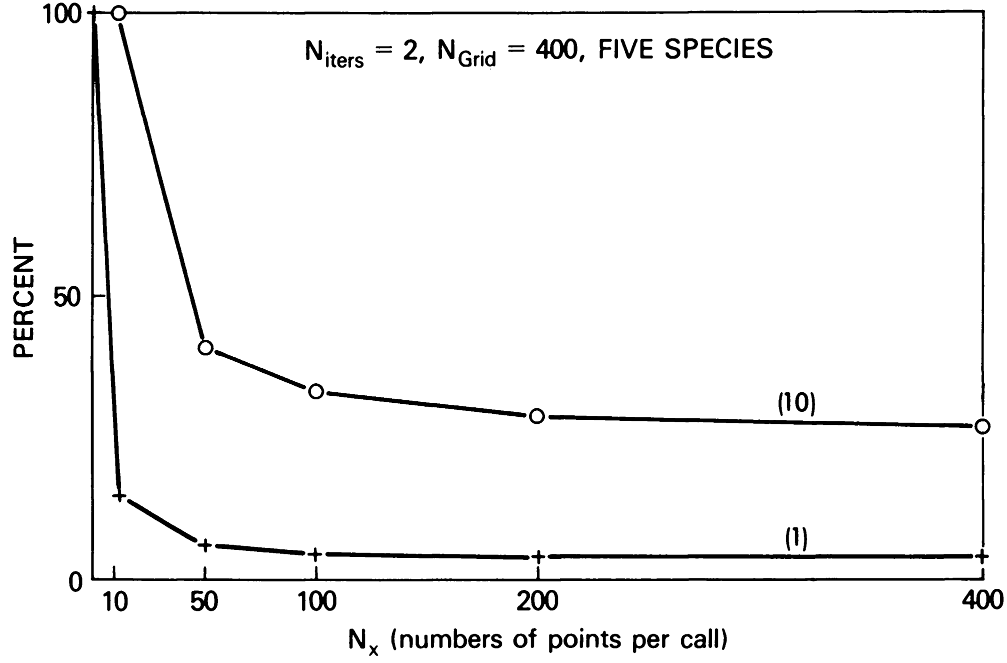


Figure 5. Percentage of time for a vector of length n *vs. the scaler mode.*

The energy (E) was added to the gas uniformly over the time τ and spatially in a Gaussian format shown in figure (6). The center of the point of energy disposition was 0.4 cm from the bottom of the cell (1/3 of the height) and on the cylinder axis. Each of the calculations was run long enough to determine whether or not ignition would occur. The only parameters varied for the runs were the magnitude of the gravitational acceleration, g, and the total energy deposited, E. In principle only six runs would be required for the six cases ($g = 0$, 1 and 10 g_o and $E = E_>$ and $E_<$), but in order to calibrate the IPM, it was necessary in each case, to start with some energy which did not ignite the mixture and increase (E) until ignition would just occur. The results of these calculations are shown in figures (7a, 7b, 7c). In each case, the percent increase over E_o is shown. A summary of these results are shown in figure (8). In order to plot the data on a "log" plot it was necessary to verify that some small acceleration did not influence the ignition properties. Thus we redid the case $E = E_o$ with $g = 0.01g_o$ to verify that indeed there was ignition and no discernable buoyancy effects showed. These results suggest an acceleration scaling of the form (1 + log (1 + 0.0175 g/g_o)), with g_o = 980 xm/sec^2.

Conclusions

Using the techniques discussed in section III and IV we have been able to study the effect of acceleration on ignition of a homogeneous fuel oxydizer mixture. The ability to study multidimensional effects (buoyancy, turbulence etc.) hinges on the use of numerical methods (slow-flow, asymptotic chemistry etc.) which circumvent the time constraints encountered in brute force techniques. These methods go hand in hand with modern fast computers, especially vector machines where judicious programming allows us to attain the actual memory or CPU cycle time.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch006

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch006

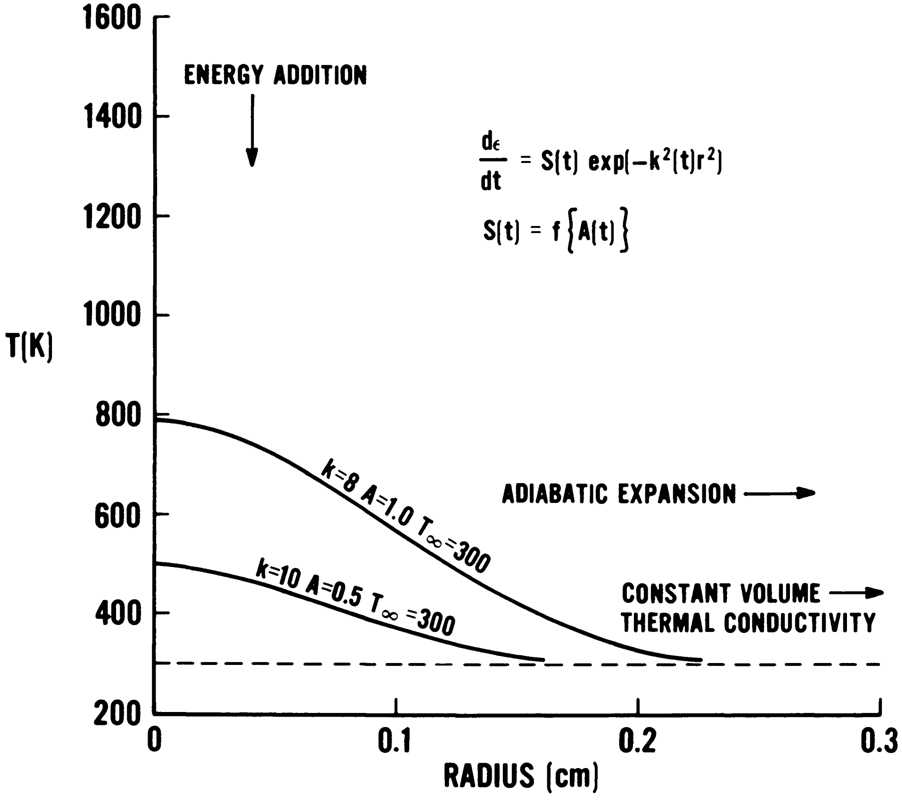


Figure 6. Energy deposition curve showing the various physical effects.

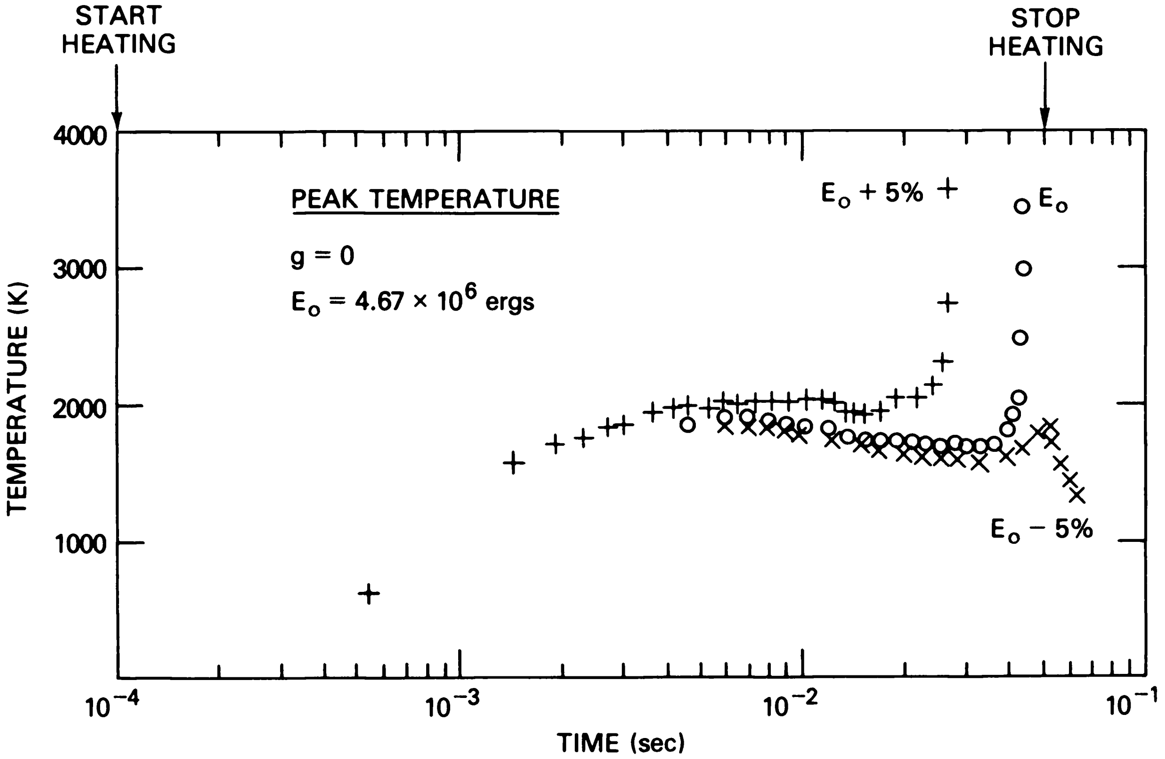


Figure 7a. Ignition energy curve. The peak temperature is shown as a function of time. The change is the magnitude of the gravitational acceleration, 0 g_0. The excess energy required (above that predicted by the IPM) is shown as a percentage of $E_o = 4.67 \times 10^6$.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch006

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch006

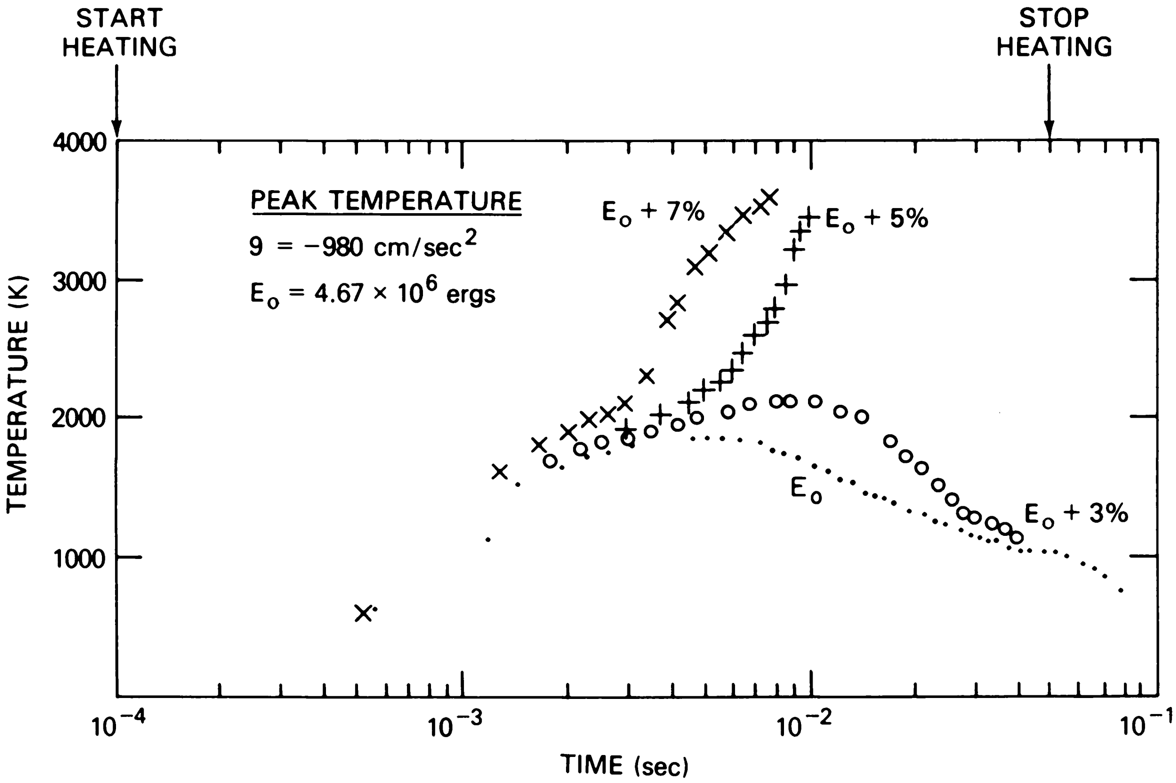


Figure 7b. Ignition energy curve. The peak temperature is shown as a function of time. The change is the magnitude of the gravitational acceleration, 1 g_0. The excess energy required (above that predicted by the IPM) is shown as a percentage of $E_o = 4.67 \times 10^6$.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch006

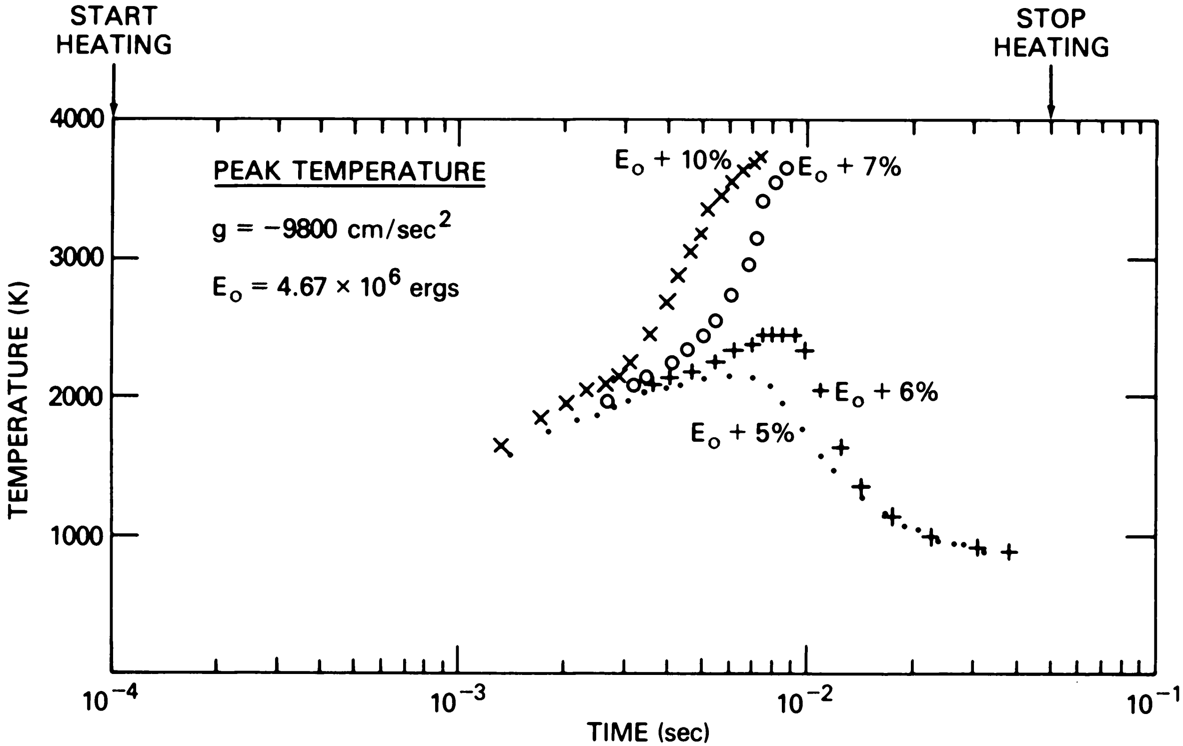


Figure 7c. Ignition energy curve. The peak temperature is shown as a function of time. The change is the magnitude of the gravitational acceleration, 10 g_0. The excess energy required (above that predicted by the IPM) is shown as a percentage of $E_0 = 4.67 \times 10^6$.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch006

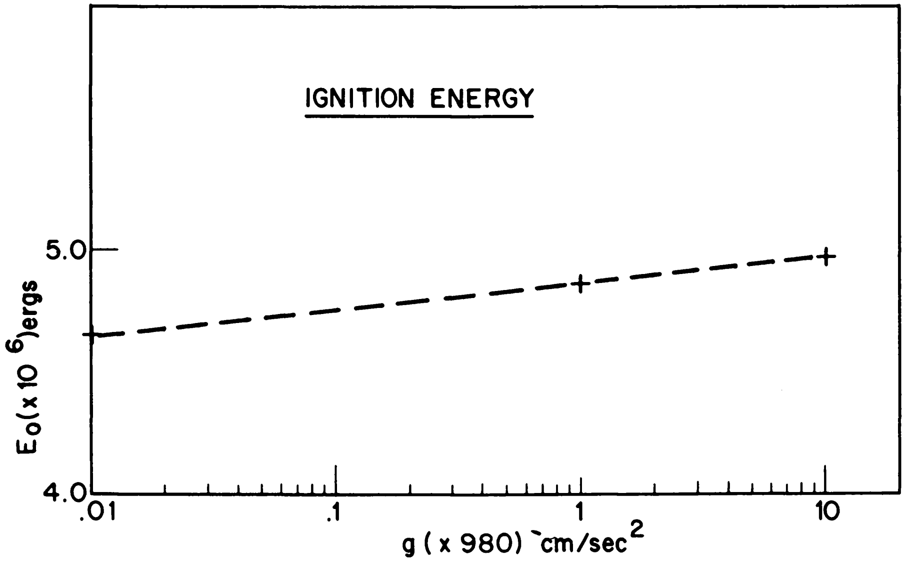


Figure 8. Summary of the ignition energy required for each value of acceleration. The modification used for g = 0 is discussed in section V.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch006

Literature Cited

1. E.S. Oran, J.P. Boris, T.R. Young, M. Flanigan, M. Picone and T. Burks, "Simulation of Gas Phase Detonation: Introduction of an Induction Parameter Model," NRL Memorandum Report #4255 (1980).

2. W.W. Jones and J.P. Boris, "Flame A Slow Flow Combustion Model," NRL Memorandum Report #3970(1979); J. Phys. Chem. 81, 2532(1977).

3. D.L. Book, J.P. Boris and K. Hain, J. Comp. Phys. 18 248 (1975); J.P. Boris and D.L. Book, J. Comp. Phys. 20, 397(1976); J.P. Boris and D.L. Book, J. Comp. Phys. 11, 38(1973).

4. W.W. Jones and J.P. Boris, "A MultiSpecies Diffusion Algorithm," submitted to Comp and Chem. (1980).

5. J.O. Hirschfelder, C.F. Curtis and R.B. Bird, Molecular Theory of Gases and Liquids, Wiley, New York (1954).

6. H.R. Buddenberg and C.R. Wilke, Industrial and Engineering Chem., 41, 1345(1949).

7. F.A. Williams, "Combustion Theory," "Addison Wesley, Reading, Mass. (1964).

RECEIVED July 27, 1981.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch007

7

Supercomputers and the Problem of Organic Synthesis

MALCOLM BERSOHN

University of Toronto, Toronto, Canada M5S 1A1

Summary

There are myriad possibilities for the synthesis of a complex organic substance, Typically, with present computers, we can examine only one route out of about 10,000 reasonable routes. On a supercomputer, concurrent parallel explorations of the branches at the top of the decision tree would allow us to eliminate poorer routes near the top of the tree and discover, at a relatively early stage, the crucial last few steps of most or all of the really promising pathways. This would be an enormous advance toward optimality of the solutions.

Introduction: The Nature of the Problem

The planning of a synthesis of a complex molecule is one of those grand problems like weather forecasting or chess playing which are clearly too big for the unaided human mind and in a certain sense are too big for any computer. Solutions are always suboptimal; there is never enough time to consider all the possibilities.

The problem to which a synthesis program addresses itself is the devising of an economic synthesis for a given input molecule. Each step or reaction of a synthesis costs money to perform and the cost depends also on the yield of each step. Hence we are looking for syntheses with as few steps as possible and with as high an overall yield as possible. The need for economic syntheses is obvious in the case of agricultural chemicals which are manufactured by the ton. It is less obvious in the case of pharmaceutical chemicals, in which case the chemical manufacturing cost is usually a small fraction of the total cost of a drug. But in fact efficient syntheses are crucial for pharmaceutical research which is always concerned with new molecules. Some of these new molecules are isolated from nature but most are synthesized in the laboratory as a result of some researcher's idea written on a blackboard. A

0097-6156/81/0173-0109$05.00/0
© 1981 American Chemical Society

projected synthesis that will take two weeks of the researcher's time is acceptable. If the best synthesis devised will take a man year then the synthesis will probably not be attempted. Evidently the rate of production of possible new drugs depends on, among other things, the quality of the synthesis planning.

The Need for a Computer

Historically, synthesis planning has not been done with the use of a computer, partly because the necessary program did not exist and partly because the need was not fully visible. During the 1940's the great synthetic chemists who had spent decades at this work were familiar with most of the reactions available. At the present time the number of new reactions is increasing by about 200 per month which is about 10% per year of the roughly 25,000 available. There is now no one who claims to know most of the synthetic reactions described in the chemical literature. Everybody performs literature search as needed. Devising good organic syntheses has been compared to playing a chess-like game in which new pieces and new legal moves are constantly being introduced.(1) There is a general awareness that a computer would be useful at the simple retrieval level. A commercial facility has originated to fill this need(2) and various companies have their private systems, the most outstanding of which is that of Fugmann and his collaborators.(3) Corey and Wipke went a step further and wrote a program which retrieves the reactions and generates the reactants for display to the chemist as an aid to his decision-making about a suitable synthetic route.(4) This work has even reached the textbooks.(5) It is fair to say that there is general concession that organic chemists have a serious information retrieval problem, which is steadily worsening and which can be resolved with a computer. There is no general recognition that the problem space is too huge for the human mind. In truth, however, the arithmetic in regard to the search of the synthetic organic chemical problem space is most discouraging for the prospects of the unaided human mind. If we wish to synthesize molecule A, then, let us say there are ten substances B, i.e. B,B',B'',B''' etc which can give rise to A in a single step. Some of these B symbols, may actually represent a pair of molecules, but the point is the same, i.e. that there are a number of ways of producing A in one step, with one chemical reaction. Now suppose that each of these substances B cannot be purchased at low cost and therefore present a synthetic problem. For each B there may be some ten molecules C, i.e. C,C',C'',C''' etc. which can give rise to B in a single step. In this manner we can work back to some substance or pair of substances which are cheap or readily available. The path from the readily available substance(s) to the goal molecule is our suggested synthesis. Suppose that eight steps are involved;

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch007

this is a typical number, which can be obtained from perusal of works on pharmaceutical synthesis(6). Then we have 10**8 or 100,000,00 different compounds potentially to be considered, and correspondingly 100,000,000 reactions. Some common reactions may be used over and over but the number 100,000,000 suggests that some of the exploration may take us through exotic chemistry and the number of literature searches would exceed that possible in many lifetimes. Incidentally the figure ten as a rough estimate for the branching factor in this decision tree is convenient for calculation but is unrealistically low for any difficult, hence economically important problem. The figure 20 would be better to use as an average in such cases. For an important problem, then, we have at least 20**8 reactions to simulate, which is 25,600,000,000.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch007

The Need for a Faster Computer

Currently the synthesis program at the University of Toronto, running on an IBM 3033 computer, requires about four milliseconds to generate the reactant B', let us say that will give rise to a product A. If B' consists of two coreactants the time will be slightly more. If the molecules have simple structures, such as those seen in elementary organic chemistry textbooks, then the time could be as little as half a millisecond.

What fraction of the possible number of eight step pathways can a program examine? In ten minutes, for our investment of $250 at commercial rates we will simulate only 150,000 reactions. To ask for a computer that provides the computations at 10,000 times the current speed is impossible, so we are forced back onto judgement, experience, statistical generalizations, in short -- heuristics, to guide the search for an efficient pathway and to tell us when to abandon an incomplete pathway as unpromising. We can shrink the size of the problem space by imposing the restriction of a particular starting material. We can also restrict the number of sequential functional group modifications that take place, as this is potentially the most time wasting and pointless part of the decision tree. There are a host of other ways in which we can build in "intelligence" into the program. But the heuristics necessarily prune out some brilliant possibilities because of the limitations of computer time. Accordingly an increase in the power of the computer would multiply the effectiveness of this program by almost a corresponding factor.

The effect of a faster computer can be looked at in another way. At present the synthesis program of the author is excellent for four step problems and markedly poorer as the problem gets more difficult and the number of steps absolutely required for the synthesis increases. Consider that 20**4 is only 160,000 which means that for a synthesis that can be accomplished in four steps the program examines substantially all of the reasonable routes. To do this for a synthesis which requires five steps would need twenty times the computation. The same kind of reasoning applies even if we do not look at all the possibilities but use heuristics to weed out a sizeable fraction of the routes without generating them. If the fraction of discarded routes remains the same, then a twenty fold increase of computer power is required for the program to be as effective for a synthesis of n+1 steps as it is on an IBM 3033 computer for a synthesis of n steps. The exponential increase of required computer time is reminiscent of graph enumeration problems.(7)

The Usefulness of a Supercomputer

First we observe that this is a "structure crunching" rather than a number crunching problem and in agreement with this the program hardly ever multiplies or divides. On the machine instruction level, the program spends most of its time fetching, comparing and branching. Some 50% of the time spent by the program is used in canonicalizing the connection table that represents the molecular structure, in other words in deciding on the proper numbering for the atoms of the molecule.(8) The routine that performs this task accumulates sums of certain atomic properties for the environments of each atom and compares them, to determine which atoms "outrank" which other atoms. In any case, it appears that array processing is inapplicable to this problem. Parallel processing might be of some small help to the low level operations of this program, concerned with inspecting the molecules, in the following way. The routine which discovers the interesting product substructures and the functional groups could perform its task on a representation of the molecule in which the atoms are arbitrarily numbered. The substructure recognition then could be done at the same time as the canonicalizing routine was deciding on the numbering. This however would speed up the program only by a factor of about 30% as the substructure recognition routine does not use up more than about 30% of the time. Sheer speed of the machine instructions with a suitable pipeline seems to be the only thing that could accelerate the elementary operations of this kind of structure manipulating program.

While the elementary, lower level operations of the program seem inapplicable for vectorization or parallelism, there are enormous possibilities for parallel algorithms at the highest level, if there were indeed a highly concurrent system of multiprocessors available. Search of the synthetic decision tree could be done in parallel in various ways, and the programming would present little difficulty. Supppose for example that the goal molecule is A and that there are twenty molecules B, B(1)... B(20) which can give rise to the goal molecule in one synthetic step. The present program generates

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch007

the structure of B(1), based on or triggered by some very promising product substructure present in goal molecule A. It then tries to solve the problem of how to synthesize B(1) without knowing anything about the other B(i) substances, which in fact have not yet been examined. If synthetic trees could be built concurrently from each of the B(i) substances then it should be possible after a short race to pick out the one approach which has the most promise in the last few steps. For example suppose the paths developed downward from B(1) toward simple molecules have not after, let us say five steps, produced decidedly smaller molecules of decidedly simpler structure than A. This may also be true of the paths developed by another processor performing the task of deriving a synthesis of B(2). But the processor addressing itself to the derivation of a synthesis for B(9) may have after five steps reached molecules of decidedly smaller size and simpler structure than A. In such case this processor could signal the others that they should abandon their tasks and help it with the pathways down from its current position towards available molecules. The rest of the problem solving would then be almost instantaneous. Furthermore the last few steps of a synthetic pathway are the most important since all material lost in the last few steps has been carried at considerable expense through all the first several steps, and therefore the last few steps critically affect the cost. Consequently the parallel search would result in a rapid optimization of the last few steps.

In the above we are presuming an evaluation function, to determine the "promise" of an incomplete route. This function takes into account the complexity of the molecule at hand in the middle of the route, i.e. the number of its chiral centers, functional groups and rings, as well as the accumulated cost (or overall yield from the molecule at hand to the goal molecule.

In the language of the program, PL/I, there exists the TASK facility,(9) which allocates concurrent functions, hence what remains of the software aspects is to write a PL/I compiler for the supercomputer of the future.

Limitations Imposed by Computer Memory

The memory capacity of a computer is also taxed by such an organic synthesis program. At present there is no attempt to retain the whole of the synthesis tree. This means we cannot do a breadth first search. If we could do a breadth first search the efficiency of the search would be improved as we would be guaranteed the shortest possible synthetic pathway. (In a depth first search we move from A down to B' down to C''', let us say and so forth. In a breadth first search one generates all the B's, then all the C's and so forth. Each of the searches terminates when an available substance is found. The depth first search is constrained in its depth by an instruction from the

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch007

user at input time. The breadth first search automatically finds the shortest synthesis.)

Large molecules occupy some 2k bytes of storage. This means that if we retain previous thinking we must have some 300 megabytes of auxiliary storage that can quickly be accessed. Again, this is prohibitively expensive on current systems.

In considering the limitions of disk storage we must further consider the data base of chemical reactions which must be available to the program. We have mentioned the number 25,000 as the number of synthetic reactions now existing. This is the present size of the file of Derwent.(2) Naturally, of these 25,000 the most popular 2000 are used more than all of the rest put together. Nonetheless, many molecules of interest have something special about them so that their synthesis may require an unusual reaction. Finally we observe that there is an enormous store of literature observations about reactions which show us that such and such a reaction is inappropriate when such and such a substructure is present because the desired reaction proceeds at a slower rate than that of a side reaction. These must in principle and sooner or later in fact be incorporated into any useful reaction data base.

If we allow 200 bytes to store a reaction, and the pertinent references for performing it in the laboratory, then we need twelve megabytes of disk storage for the database of synthetic reactions. This is not yet realistic at Toronto; we have only a little more than 1000 reactions available. However the number is being rapidly added to. Certainly the twelve megabytes of storage will be used in the ultimate synthesis program that will emerge in the 1990's.

Appendix: Description of the Program

Assuming that a reader of this book is more interested in vector processors, decision trees and differential equations than in organic chemistry, I will provide only some examples which are intended to clarify the foregoing discussion.(10) Figures 1 and 2 show some typical results of the program.

Input to the Program

The user specifies a desired upper limit on the number of steps in an acceptable synthesis. The user also specifies his lower limit on the overall yield. In addition there must be some kind of definition of what is an acceptably simple molecule, i.e. one whose structure is so simple that it can be assumed to be readily available or cheaply prepared. This definition is conveyed by the user in the form of the minimum number of chiral centres, rings and functional groups that are acceptable in a "readily available" material. In addition the structure of a preferred starting material can be input. Finally the structure of the goal molecule is provided by the user. Structures of molecules are communicated to the program in the form of

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch007

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch007

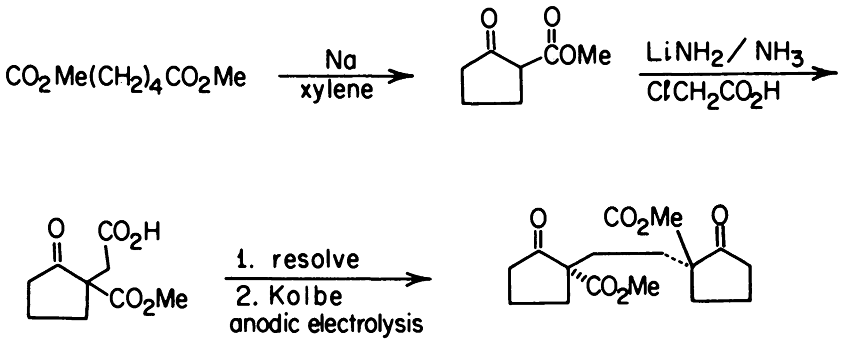


Figure 1. A synthesis suggested by the program, illustrating its understanding of molecular symmetry.

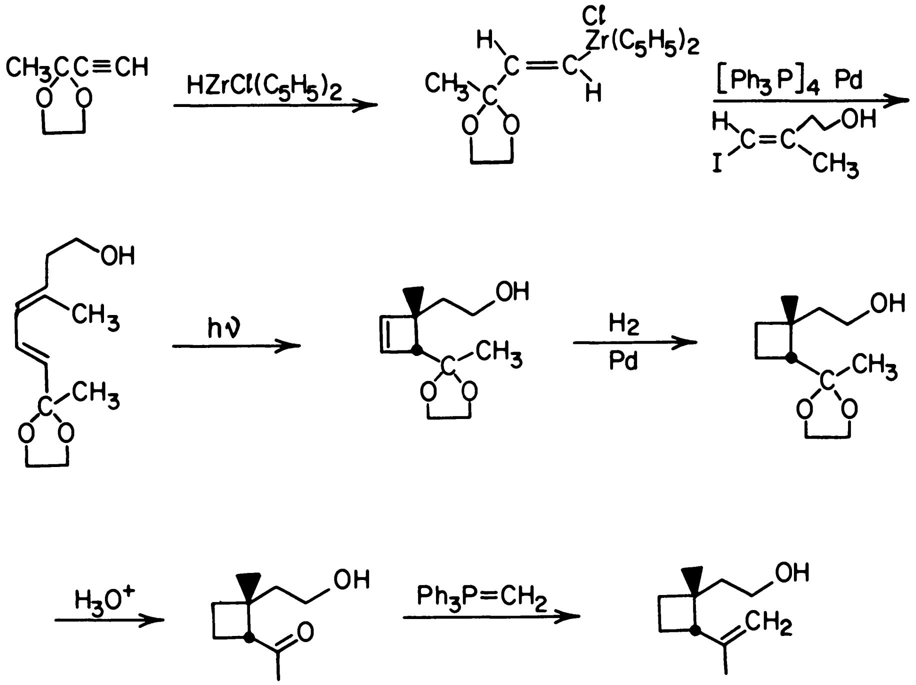


Figure 2. A suggested synthesis, illustrating the broad scope of the reactions available in the program.

connection tables together with separate lists of chirality and double bond stereochemical information.

Literature Cited

1. W.T.Wipke in "Computers in Chemical Education and Research", E.V. Ludena,N.H.Sabelli and A.C.Wahl,Eds.,Plenum Press, N.Y., 1976,p 381
2. Derwent Publications Ltd., Rochdale House,128 Theobalds Rd., London WC1, England
3. R.Fugmann "The IDC System" in "Chemical Information Systems", J.E.Ash and E.Hyde, Editors, John Wiley & Son, New York(1975), pp 195 et seq.
4. E.J.Corey and W.T.Wipke,Science 166,178(1969); W.T.Wipke, D.Dolata,M.Huber and C.Buse, chapter in ACS Symposium Volume No. 112, "Computer Assisted Drug Design", E.C.Olson and R.C. Christoffersen, Editors(1979)
5. F.A.Carey and R.J.Sundberg, Advanced Organic Chemistry", Plenum Publishing Corp., New York(1977)
6. D.Lednicer and L.A.Mitscher, "The Organic Chemistry of Drug Synthesis",John Wiley & Sons, New York, 1977
7. J.L.Martin,"Computer Techniques for Evaluating Lattices", in "Phase Transitions and Critical Phenomena",vol. 3, Academic Press,London,1974
8. M. Bersohn, Computers & Chemistry, 2,118(1988)
9. PL/I Language Reference Manual, GC33-0009, IBM Data Processing Division, White Plains, New York(1976)
10. M. Bersohn, Bull. Chem. Soc. Japan, 45, 1897-1903(1972); M. Bersohn,"Syntheses of Drugs Proposed by a Computer Program," American Chemical Society Symposium Volume on Computer Assisted Drug Design, E.C. Olson & R.E. Christofferson, Eds., 1979 pp 341-352; M.Bersohn, A. Esack and J. Luchini, Computers & Chemistry, 2,105(1978)

RECEIVED April 29, 1981.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch007

Computer Methods of Molecular Structure Elucidation from Unknown Mass Spectra

IN KI MUN and FRED W. McLAFFERTY

Cornell University, Department of Chemistry, Ithaca, NY 14853

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch008

Mass spectrometry (MS) is now a well-accepted tool for the identification as well as quantitation of unknown compounds. The combination of MS with powerful separation methods such as gas chromatography (GC) or high-performance liquid chromatography (LC) provides a technique which is widely accepted for the identification of unknown components in complex mixtures from a wide variety of problems such as environmental pollutants, biological fluids, insect pheromones, chemotaxonomy, and synthetic fuels. The importance of such analyses has grown exponentially in the last few years; there are now well over a thousand GC/MS instruments in use around the world, most with dedicated computer systems which make possible the collection from each of hundreds of unknown mass spectra per day (1).

One of the most serious limitations in the application of these powerful GC/MS and LC/MS systems is the accurate and efficient identification of this flood of unknown mass spectra. A variety of computer-assisted techniques have been proposed (2, 3, 4), which can be classified generally as "retrieval" or "interpretive programs (2). The former matches the unknown mass spectrum against a data base of reference spectra; the ultimate limitation of this approach is the size of the data base, which currently contains the mass spectra of 33,000 different compounds (5, 6), less than 1% of the number listed by Chemical Abstracts. If a satisfactorily-matching reference spectrum cannot be found by the retrieval program, an interpretive algorithm can be used to obtain partial or complete structure information, or to aid the human interpreter in this task (7-10).

This paper will focus on interpretive algorithms; the types of programs proposed and in use will be compared in terms of their functions and applications. Priorities for possible improvements to these algorithms will be proposed with particular reference to the most imperative needs which we perceive. Also of importance are the new opportunities arising from the rapid improvements in capacity, availability, and cost of powerful computer resources.

0097-6156/81/0173-0117$05.00/0
© 1981 American Chemical Society

Functions of an Interpretive System

Interpretation should generally follow the paradigm of heuristic search, "plan-generate-test" (11). Proposed algorithms differ in the emphasis placed on each of these steps; often one or more steps are left to the human interpreter. In the context of mass spectral interpretation the "planning" phase can be defined (12) as translation of the available information (unknown mass spectrum, etc.) into structural features such as molecular weight, elemental composition, and specific substructures. "Generation" involves constructing all possible molecules consistent with these data (13). "Testing" utilizes methods for ranking these postulates, such as by predicting their mass spectra and comparing these to the unknown (12-16). Thus, in "testing" one attempts to convert structural data into spectral data, the opposite of "planning", in which the unknown spectral data is used to postulate structural information.

Structural Information from Spectral Data. The kinds of information that can be derived from an unknown mass spectrum by either human or computer examination include the identities of substructural parts of the molecule (parts that both should, and should not, be present), data concerning the size of the molecule (molecular weight, elemental composition), and the reliability of each of these postulations. In our opinion, the latter is much more critical for mass-spectral interpretive algorithms than those for techniques such as NMR and IR; the effect of a particular substructure on the mass spectrum is often dependent on other parts of the molecule, and a thorough understanding of these effects can only be achieved by studying the spectra of closely related molecules.

Several algorithms have been proposed which attempt to duplicate the human interpretive process (2, 3, 4, 17, 18), coding known fragmentation pathways so that the program can deduce structural features. However, complex programs are required to interpret the spectra of simple molecules. Because synergistic effects of functional groups are ubiquitous in mass spectral decompositions, we see little hope that useful programs can be designed for other than spectra of narrowly-defined structural classes. For similar reasons present programs of this type should not be significantly helpful for total unknowns not in the current reference file (5, 6).

However, almost any unknown will contain some substructures which are well-represented in the reference file. Because of this, useful substructural information can be obtained from retrieval program results, as the best-matching compounds often show structural features similar to those of the unknown. For example, SISCOM (which recognizes structural SImilarity COding Multiple matching factors) provides such substructural information as well as a matching capability (4). "Pattern recognition" (19) is a

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch008

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch008

powerful technique for such correlations, but cannot take advantage of known mass spectral behavior without pretraining of the algorithm, a disadvantage similar to the interpretive programs coding fragmentation rules (20). The Self-Training Interpretive and Retrieval System (STIRS) (7) does not require pretraining; it has been optimized for such feature recognition by defining 17 classes of mass spectral data particularly suited for the identification of different types of substructures. The data of the unknown mass spectrum corresponding to each of the pre-selected classes is matched against the corresponding data for all compounds in the reference file; if a significant proportion of the 15 best matching compounds contain a specific structural feature, this is indicative of the presence of that structural feature in the unknown (7). In a recent study (8) a list of 589 substructures was selected on the basis of STIRS performance using 899 random unknowns; knowing this performance, the number of the 15 best-matching compounds containing each substructure can be used to calculate the probability that it is present in the unknown. Table I shows an example of such results.

A modification of the STIRS algorithm (9) makes possible the prediction of the molecular weight from an unknown mass spectrum with 91% reliability (95% for the first and second choices). This program has been extended recently to predict the elemental composition (10) with somewhat lower accuracy. An example of the information supplied by STIRS is given in Table I.

Generation of Possible Molecular Structures. The structural information derived from the spectral data can then be used to define all possible combinations which give logical molecular structures. By far the most comprehensive program available for this purpose is CONGEN, developed by the Stanford group (12, 13). The program is given the elemental composition of the unknown and lists of structural features which are, and are not, present; from these it generates all chemically-logical molecular structures consistent with these restrictions. Note that the program assumes that all these structures are correct, while the information from mass spectra is usually of <100% reliability (see Table I).

Testing of Postulated Structures. For a complex molecule the generation program usually finds a multiplicity of possible structures. Ranking the reliability of these involves obtaining further information to narrow the choices, or predicting their spectra (12-18) for comparison to the unknown spectrum. Probably the most sophisticated prediction programs are those developed by the Stanford group (12, 13, 15, 16). Their experience shows that distinction between closely related compounds requires developing fragmentation rules based on the spectra of many such compounds. Subtle spectral differences are often important; for these it is necessary to utilize class-specific rule-based approaches (15). Impressive results can be obtained for unknowns known to be in the

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch008

Table I. STIRS Predictions from the Mass Spectrum of *n*-Propyl *p*-Hydroxybenzoate

Mol. wt.	K[a]	Elem. comp.
180	90	$C_{10}H_{12}O_3$
181	37	
210	-45	

Substructures[b]		Reliability
27.	$HO-C_6H_4-CO-$	98%
443.	C_6H_5-CO-, $-C_6H_4-CO-$	98%
450.	$HO-C_6H_4-$	97%
54.	$-CO-O-$	84%
468.	$HO-C_6H_4-CO-O-$	77%
101.	$-O-CH_2-$	75%

Best matches[b]	27	443	450	54	468	101
p-$HOC_6H_4COOCH(CH_3)_2$	+	+	+	+	+	+
m-HOC_6H_4COOH	+	+	+	+	+	
o-$HOC_6H_4COOCH(CH_3)_2$	+	+	+	+	+	
m-$HOC_6H_4CONHCH_2COOH_3$	+	+	+			
p-$HOC_6H_4COOC_2H_5$	+	+	+	+	+	+
p-HOC_6H_4COOH	+	+	+	+	+	
p-$HOC_6H_4CONHNH_2$	+	+	+			
m-$C_2H_5OC_6H_4COOH$		+		+		+
m-$HOC_6H_5COCH_3$	+	+	+			
p-$HOC_6H_5COCH_3$	+	+	+			
p-$HOC_6H_4COC_6H_4$-*o*-OH	+	+	+			
p-$HOC_6H_4COC_6H_4$-*o*-$COOH$	+	+	+	+		
p-$HOC_6H_4COO(CH_2)_3CH_3$	+	+	+	+	+	+
o-$HOC_2H_4COOCH_2CH(CH_3)_2$	+	+	+	+	+	+
p-$HOC_6H_4COOCH_3$	+	+	+	+	+	

---

[a] K is indicative of the prediction confidence on an arbitrary scale (9)

[b] Predicted by the overall match factor MF11.3 (9)

limited number of compound classes for which such algorithms are presently available, such as estrogens and marine sterols (16).

Unknown Spectra with Minimal Additional Information

For the complex mixtures to which GC/MS and LC/MS are most commonly applied, the unknown components more often are only known to have specific chromatographic-retention behavior, or to belong to very broad classes of compounds, such as basic or non-polar. For example, for a synthetic fuels component many structures might be highly improbable (nucleotides, fluorocarbons); however, the component could also be most any type of hydrocarbon (alkane, aromatic) or heterocyclic compound and could contain a variety of functionalities such as hydroxyl, ether, thiol, and amino. Although these may not be truly "total" unknowns, programs for interpreting or predicting specific classes of compounds will be of little use unless it can be shown which one is applicable. Conversely, the results from an algorithm applicable to "total" unknowns can be used by the interpreter in combination with whatever additional information is available. For example, dramatic recent improvements in Fourier-transform infrared instrumentation give promise that infrared spectral data may soon also be available from GC/IR/MS.

For the identification of such complex mixtures there will also be a wide range in the interpreter's needs for specificity, reliability, and speed. For example, if a component very probably is a tetrachlorodibenzodioxin it may be highly important to see if this is the 2,3,7,8-isomer; if a butyl phthalate is identified by the computer, the interpreter may not care to spend further time in evaluating the isomeric possibilities.

Molecular Structure Postulation with STIRS and CONGEN. For the spectrum of an unknown component which is not known to belong to a compound class for which an efficient interpretive algorithm is available, STIRS at present appears to provide the most extensive structural information without human interpretation. For the prediction of 589 substructures STIRS also provides a reliability value automatically (8). However, our efforts to use the STIRS predictions of the elemental compositions and substructures with CONGEN to predict possible structures for the unknown have to date given unsatisfactory results. One reason is that CONGEN assumes that the substructures do not overlap; the GENOA program under development by the Stanford group (21) will not have this restriction. However, the most serious limitation is that substructure predictions by STIRS are not 100% reliable (Table I). Increasing the number of STIRS-predicted substructures used by CONGEN reduces the number of structural possibilities it generates, but increases the probability that part of the STIRS information is wrong; with one wrong substructure as input, all of the CONGEN output must be wrong. Thus in the final "test" phase, which for such unknowns

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch008

must be done by the human interpreter, prediction of spectra for postulated structures is made even more difficult. Development of a computer method for spectral prediction applicable to most types of compounds would obviously be helpful; a "self-training" approach similar to that of STIRS is a possibility.

The Supercomputer Approach. The computer time required for generation of all possible structures by CONGEN is highly dependent on the input data, but this can require minutes of CPU time on an IBM-370/168. If computer systems such as those discussed in this symposium could reduce this dramatically, one could visualize trying various combinations of the STIRS-generated data to find those leading to only a few possibilities. The reliability of each predicted set could then be estimated by combining the reliabilities of the STIRS data used to obtain the set. However, our initial experience suggests that for more complex unknowns this system will often generate a large number of possibilities of similar reliability.

Possible Improvements to STIRS

Even if improved hardware and/or software is feasible for the "generate" and/or "test" phases of structure elucidation, it would appear that improvements in the first phase for derivation of structure information could substantially reduce the effort required for these latter two phases, and similarly increase the proportion of unknowns for which the interpreter can derive a satisfactory answer only aided by STIRS. (The availability of infrared absorption data from GC/IR/MS would be particularly helpful.)

STIRS Prediction of Maximal Substructures. As outlined above, STIRS predicts the presence and reliability of 589 substructures (8) from the proportion of the 15 best-matching spectra which represent compounds containing the substructure. The interpreter can infer the qualitative presence of substructures not on this list of 589 in a similar fashion, but this can be a tedious procedure.

The supercomputer could also offer a solution to this problem. An algorithm has been devised which can generate the largest substructures which two compounds have in common (22). It thus can intercompare all of the 15 best-matching compounds to find substructures predicted by STIRS which are not in the 589. This maximal-substructure algorithm is not used routinely, however, because of the extensive computer time required; again, the supercomputer could make this feasible.

Automatic identification of larger substructures with simpler algorithms can be visualized. Substructures from the 589 list identified by STIRS could be used to restrict, and thus speed, the maximal substructure search. Those compounds of the 15 best-matches containing the substructure with the highest reliability

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch008

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch008

can be checked for which of the other identified substructures they contain most often; lists of such correlations of the identified "589" could easily be generated (the bottom of Table I shows a correlation of predicted substructures and best-matching compounds which should aid the interpreter in this analysis). Any commonality of relationship to each other in these molecules (e.g., overlapping carbonyl, phenyl of one attached to sulfur of the other) could also be determined from their respective connection tables by the computer (or by the interpreter).

STIRS Prediction of Substructures Not Present. CONGEN can utilize a BADLIST of features known not to be present in the molecule. Negative information is more difficult to determine from an unknown mass spectrum because other parts of the molecule may suppress a fragmentation reaction characteristic of a specific function. However, this is less true of substructures such as amino which strongly influence ion decomposition pathways. Thus it might be useful to carry out an evaluation of the ability of STIRS to identify the absence of specific substructures in the same way used to find the 589 substructures whose presence is best found by STIRS (8).

Aids to Interpreter Generation of Possible Structures. Simpler structure-generation programs could be helpful to the interpreter, even though they would be less thorough than CONGEN (12, 13). An algorithm could generate the possible combinations of the STIRS-identified substructures (not the actual molecular possibilities as generated by CONGEN) consistent with the predicted elemental composition (or molecular weight). This would use the elemental compositions of the substructures, and should consider possible overlaps; the substructures CH_3CO- and -CO-OCH(CH_3)- could be present in the unknown as CH_3CO-OCH(CH_3)-. For example, the $C_{10}H_{12}O_3$ composition and substructures predicted in Table I, if all correct, are only consistent with the isomeric molecules HO-C_6H_4-CO-O-CH_2-C_2H_5, despite the fact that the final C_3H_7 is not present in any of the identified substructures.

Acknowledgment. We are indebted to R. E. Carhart, R. G. Dromey, K. S. Haraki, S. Russo, D. H. Smith, D. B. Stauffer, and R. Venkataraghavan for stimulating discussions and to the National Science Foundation (Grant CHE-79-10400) for generous financial support.

Literature Cited

1. Burlingame, A. L.; Baillie, T. A.; Derrick, P. J.; Chizhov, O. F., Anal. Chem. (1980), 52, 214R .

2. Pesyna, G. M.; McLafferty, F. W., "Determination of Organic Structures by Physical Methods", Vol. 6 pp. 91-155, Academic Press, New York, 1976.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch008

3. Chapman, J. R., "Computers in Mass Spectrometry", Academic Press, New York, 1978.

4. Henneberg, D., *Adv. Mass Spectrom.* (1980) *8*, 1511.

5. Heller, S. R.; Milne, G. W. A., "EPA/NIH Mass Spectral Data Base", NSRDS-NBS 63, U.S. Government Printing Office, Washington, D.C., 1978.

6. Stenhagen, E.; Abrahamsson, S.; McLafferty, F. W., "Registry of Mass Spectral Data", extended version on magnetic tape, John Wiley, New York, 1978.

7. Kwok, K.-S.; Venkataraghavan, R.; McLafferty, F. W., *J. Am. Chem. Soc.* (1973) *95*, 4185.

8. Haraki, K. S.; Venkataraghavan, R.; McLafferty, F. W., *Anal. Chem.* (1981) *53*, 386-392.

9. Mun, I. K.; Venkataraghavan, R.; McLafferty, F. W., *Anal. Chem.* (1981) *53*, 179-182.

10. Mun, I. K.; McLafferty, F. W. In preparation.

11. Feigenbaum, E. A., "Information Processing 68", North Holland, Amsterdam, 1968.

12. Carhart, R. E.; Varkony, T. H.; Smith, D. H., "Computer-Assisted Structure Elucidation", p. 126, American Chemical Society, Washington, D.C., 1977.

13. Smith, D. H.; Buchanan, B. G.; Englemore, R. S.; Duffield, A. M.; Yeo, A.; Feigenbaum, E. A.; Lederberg, J.; Djerassi, C., *J. Am. Chem. Soc.* (1972) *94*, 5962.

14. Zander, G. F.; Jurs, P. C., *Anal. Chem.* (1975) *47*, 1562.

15. Gray, N. A. B.; Carhart, R. E.; Lavanchy, A.; Smith, D. H.; Varkony, T.; Buchanan, B. G.; White, W. C.; Creary, L., *Anal. Chem.* (1980) *52*, 1095.

16. Lavanchy, A.; Varkony, T.; Smith, D. H.; Gray, N. A. B.; White, W. C.; Carhart, R. E.; Buchanan, B. G.; Djerassi, C., *Org. Mass Spectrom.* (1980) *15*, 355.

17. Yamasaki, T.; Abe, H.; Kudo, Y.; Sasaki, S.-I., "Computer-Assisted Structure Elucidation", p. 108, American Chemical Society, Washington, D.C., 1977.

18. McKeen, L. W.; Taylor, J. W., *Anal. Chem.* (1979) *51*, 1368.

19. Jurs, P. C.; Isenhour, T. L., "Chemical Applications of Pattern Recognition", Wiley-Interscience, New York, 1975.

20. Isenhour, T. L.; Lowry, S. R.; Justice, Jr., J. B.; McLafferty, F. W.; Dayringer, H. E.; Venkataraghavan, R., *Anal. Chem.* (1977) *49*, 1720.

21. Carhart, R. E.; Smith, D. H.; Gray, N. A. B.; Nourse, J. G.; Djerassi, C., *J. Org. Chem.* (1981) *46*, in press.

22. Cone, M. M.; Venkataraghavan, R.; McLafferty, F. W., *J. Am. Chem. Soc.* (1977) *99*, 7668.

RECEIVED May 25, 1981.

9

The Relative Performances of Several Scientific Computers for a Liquid Molecular Dynamics Simulation

D. M. CEPERLEY

National Resource for Computation in Chemistry, Lawrence Berkeley Laboratory, Berkeley, CA 94720

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch009

In the last decade, the computer modeling of matter by simulations has become a very important area of theoretical chemistry.(1) One goal of the simulations is to understand the properties of macroscopic systems starting from the Coulomb potential and Schroedinger equation. Although it is feasible that simulation methods can treat the complete many-body quantum problem,(2) most simulations today assume a classical model with some effective interparticle potential. There are two common methods employed, Metropolis Monte Carlo (MC)(3) is an effective algorithm used for calculating static properties of many-body systems. Molecular Dynamics (MD)(4) is the term employed when Newton's equations of motion are solved to find equilibrium and non-equilibrium, dynamic and static properties of many-body systems. What all of these simulations methods have in common and their limitation is a processor fast enough to move hundreds of atoms, hundreds and thousands of times. What I want to discuss in this short note, are some of the computational characteristics of simulations and my experience in using a standard simulation program on several scientific computers.

While at the National Resource for Computation in Chemistry, I have developed a general classical simulation program, called, CLAMPS (for classical many particle simulator)(5) capable of performing MC and MD simulations of arbitrary mixtures of single atoms. The potential energy of a configuration of N atoms at positions $R = \{r_1, \ldots, r_N\}$ and with chemical species $\{\alpha_1, \ldots, \alpha_N\}$ is assumed to be a pairwise sum of spherically symmetric functions.

$$U = \sum_{i<j} \phi_{\alpha_i\alpha_j}(|r_i - r_j|_M) \qquad (1)$$

Where $\phi_{\alpha\beta}(r)$ is the interaction between two atoms of type α and β and $|r|_M$ means the minimum image distance consistent with periodic boundary conditions. In addition, there can be

0097-6156/81/0173-0125$05.00/0
© 1981 American Chemical Society

bonding potentials between certain pairs of atoms. If some of the atoms are charged there is another term in the potential energy arising from the interaction of a charge with the charges outside the simulation box: the Ewald image potential.(6) This can be conveniently written as

$$U_E = \sum_k v_k |\rho_k|^2 \qquad (2)$$

Where k is a vector in the reciprocal lattice of the simulation cell, ρ_k is the Fourier transform of the charge density,

$$\rho_k = \sum_i q_i \exp(ik \cdot r_i) \qquad (3)$$

q_i is the charge of particle i, and v_k is the Fourier component of the long range potential.(6)

In simulations, computation of the potential energy and forces takes the vast majority of the computer time. The other operations, such as moving the particles, usually are much quicker. Shown in Table I is the FORTRAN coding needed to compute the pair sum in eq. (1). In a general purpose program such as CLAMPS one cannot assume that the pairwise interactions are simple enough to compute at each step, whereas, a table lookup is equally efficient for all systems. In CLAMPS the potentials and the derivative $-r^{-1}d\phi/dr$ are computed on a grid linear in r^2 at the beginning of the program and stored. Tables with the order of 10^4 entries usually give sufficient accuracy for most problems without any interpolation because of the statistical nature of the computation.

The coding in Table I illustrates the central problem of simulations. The number of pairs is N(N-1)/2. The number of floating point operations (FLOPS) per pair is about 25, assuming the branches are executed 50% of the time. Thus for 100 atoms (a minimal simulation) we will need 1.2×10^5 FLOPS for a single time step. The number of memory and indexing operations is similarly large. Typically one needs to execute between 10^3 and 10^5 time steps. Thus the simulations are limited by the number of floating point operations one can afford.

For systems which can be modeled with particles interacting with only short ranged forces (that is the potential can be neglected beyond several neighbor shells), the number of operations per time step will be proportional to the number of particles times the average number of neighbors of a given particle. For such models, simulations of 10^4 atoms are possible today on available mainframe as well as minicomputers. For many chemical systems, such as those containing macromolecules, one would like to work with still larger systems over many time steps. Even with today's computers, most chemical systems cannot be simulated without

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch009

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch009

Table I

```
C LOOP OVER ALL PAIRS OF ATOMS I, J
      DO 1 I=1, NATOMS-1
      DO 2 J=I+1, NATOMS
C CALCULATE PERIODIC DISTANCES
      R2=0.0
      DO 3 L=1, NDIM
      DX(L)=X(I,L)-X (J,L )
C ELL AND EL2 ARE THE BOX AND HALF THE BOX LENGTHS
      IF (DX(L).GT.EL2(L)) DX(L)=DX(L)-ELL(L)
      IF (DX(L).LT.-EL2(L)) DX(L)=DX(L)+ELL (L)
3     R2=R2+DX( L )**2
C IT AND JT ARE THE CHEMICAL TYPES
      IT=ITYPE( I )
      JT=ITYPE( J )
C CONVERT DISTANCE TO A TABLE ENTRY
      LI=CSI ( IT,JT)*R2
C IF OUTSIDE TABLE POTENTIAL IS ZERO
      IF(LI.GE.LMAX) GO TO 2
C LT IS THE TABLE FOR THIS INTERACTION
      LT=LTABLE( IT,JT)
C LOOK UP POTENTIAL AND DERIVATIVE
      V=V+EPS(IT,JT)*PTABLE(LI,LT)
      FT=EPSF(IT,JT)*FTABLE(LI,LT)
C NOW ADD TO FORCES
      DO 4 L=1, NDIM
      F=FT*DX(L)
      FORCE( I,L )=FORCE(I,L)+F
4     FORCE( J,L )=FORCE(J,L)-F
2     CONTINUE
1     CONTINUE
```

FORTRAN Code for the pairwise sum of eq. (1).

making many simplifying assumptions. Both a supercomputer as well as better algorithms are necessary to tackle these problems.

For the purpose of comparing performance on different computers, I have used the Stillinger-Lemberg(7) model for water. This model contains central force interactions between charged oxygen and hydrogen atoms. The three different potential functions between OO, OH and HH, are tailored to give the correct geometry and dipole moment for an isolated molecule and some of the pair bonding properties of two molecules

Simulations of charged systems are very important. Common examples are plasmas, ionic solutions, dipole system and electronic systems. Because all pairs are included in the sum of eq. (2), the computer time only depends on the number of atoms and the number of time steps. For this reason my results should be applicable to all similar systems. I will discuss here only results for molecular dynamics simulations. The situation for Monte Carlo is completely parallel, although the actual coding is different since atoms are moved singly rather than all together.

Because the atoms are charged, the Ewald image potential from eq. (2) must be used to account for the long-range Coulomb potential. In the following benchmarks, I have included all terms in the sum in eq. (2) for which $k \leq 6\pi/L$; this comprises 123 terms, and is adequate to represent the potential to one part in 10^4. As long as the number of terms is held fixed, the computer time to evaluate eq. (2) will be proportional to the number of atoms while the pairwise sum in eq. (1) will take time proportional to the square of the number of atoms. Thus for large enough systems, it is the pairwise sum which dominates the calculation. The sines and cosines needed for ρ_k are computed recursively. I will not discuss the computation of the Ewald sum in detail, because it is relatively specialized.

Computer Comparisons

In this section, I will discuss the programming considerations and timing results for the four computers on which I have tested CLAMPS. In all cases the code was not substantially changed. Essentially only the routines which performed the sums in eqs. (1) and (2) were modified. All changes were in FORTRAN or with FORTRAN callable routines. The timing results are not optimal, but rather typical of what could be achieved by a user in FORTRAN. The timing results are given in Table II for systems containing 27 and 216 molecules (81 and 648 atoms). MFLOPS refers to the number of million floating point operations per second in executing the pairwise sum of Table I assuming each pass through consists of 25 floating point operations.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch009

Table II

T_p is the time in seconds to execute the pairwise sum in equation (1); T is the total time in seconds per molecular dynamic step. MFLOPS is the number of million floating point operations per second of the code in Table I, assuming that it contains 25 FLOPS (i.e., MFLOPS = 1.25 x 10^5 x $N(N-1)/T_p$) where N is the total number of atoms, 81 or 648. The asterisk on CRAY-1 indicates a vectorized version of CLAMPS was used.

Computer	81 Atoms			648 Atoms		
	Tp	MFLOPS	T	Tp	MFLOPS	T
VAX 11/70	0.35	0.23	1.63	22.2	0.24	32.5
CDC 7600	0.033	2.5	0.125	2.1	2.5	2.85
CRAY-1	0.0182	4.5	0.100	1.1	4.8	1.77
CRAY-1*	0.0070	11.6	0.0157	0.257	20.4	0.311
VAX-FPSAP	-	-	-	25.0	0.21	-

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch009

DEC VAX 11/70

The VAX used, is located at NRCC in Berkeley, has a floating point accelerator, 2.5 M Bytes of memory, and was running version 1.3 of the operating system. The code was run in single precision (32 bits/word) and that was found adequate to conserve energy and give satisfactory equilibrium properties. The code used to perform the pairwise sum is essentially that of Table I.

CDC 7600

The 7600 used is located at Lawrence Berkeley Laboratory, is approximately ten years old and has 65 K of 60 bit word fast memory (small core). Because CLAMPS has dynamic memory allocation, it is possible to fit a simulation in fast memory of up to about 2000 atoms as long as the potential tables are not too extensive. The compiler used was the standard CDC FTN 4.8, OPT=2. The only difference between the CDC coding of the pairwise sum and that in Table I is that the periodic boundary conditions (loop 3) are handled by Boolean and shift operations instead of branches. Branches on the 7600 causes all parallel processing to halt.

CRAY-1

The CRAY used is located at Lawrence Livermore Laboratory. Characteristics of the CRAY are described elsewhere in this volume. There is a large advantage in achieving vector rather then scalar code. This can be seen in Table II. Initially, the CDC version of CLAMPS was run on the CRAY with the time results showing it only slightly faster than the 7600. Several subroutines of CLAMPS were then vectorized and the simulation executed in approximately 1/5 the time. The vectorized version of the pairwise sum appears in Table III. The problems encountered in vectorizing this routine were:

1) The periodic boundary conditions in loop 3 contain 2 branches. Vectorization was achieved by using the FORTRAN callable vector merge function.

2) The branch for the case when the squared pair separation is outside the table will inhibit vectorization. The last element of the table has been changed to zero and all occurrence outside the table are truncated to LMAX. The rest of the code, which is not executed on the VAX or CDC 7600, is executed here. It is often necessary on a vector machine to increase the total number of floating point operations to achieve vector rather than scalar processing. The MFLOP rates reported here are computed on the basis of the original number of floating point operations. The extra ones added to achieve vectorization are not included.

3) The table look-ups for the force and potential can be done with the GATHER function. GATHER (N, A, B, INDEX) is equivalent to the FORTRAN statements.

```
                    DO 2 I = 1, N
           2        A(I) = B(INDEX(I))
```

Although GATHER is a scalar operation and rather slow, 12 machine cycles/element, (a machine cycle is 12.5 ns), GATHER is faster than computing all but the simplest inverse power potentials. By comparison a square root takes 14 machine cycles/element and the entire pairwise sum takes an average of 98 machine cycles/pair. Note that temporaries are set up for the scaling factors of the potential, as well as the addresses for the start of the tables. These temporaries are changed only rarely and so do not affect the timing. They would be unnecessary if there were only one type of particle.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch009

Table III

```
C LOOP OVER ALL PAIRS OF ATOMS I,J
      ITL=0
      DO 1 I=1,NATOMS-1
      I1=I+1
      NC=NATOMS-I
C CHECK TO SEE IF WE NEED TO REFRESH OUR TEMPORARIES
      IF( ITYPE(I).EQ.ITL) GO TO 20
      ITL=ITYPE( I )
C GATHER MAKES EPST(J )=EPS(ITYPE(J),ITYPE(I))
      CALL GATHER(NC,EPSF (I1),EPS(1,ITL),ITYPE(I1))
C GATHER ALSO EPSF,  CSI AND LTABLE
      CALL GATHER(NC,EPSFT(I1),EPSF(1,ITL),ITYPE(I1))
      CALL GATHER(NC,CSIT(I1),CSI(1,ITL),ITYPE(I1))
      CALL GATHER(NC,LTABT(I1),LTABLE(1,ITL),ITYPE(I1))
2     DO 3 J=I1,NATOMS
3     R2(J)=0.
C CALCULATE PERIODIC DISTANCES
      DO 4 L=1,NDIM
      T=SIGN( ELL(L),X(I,L)
      DO 5 J=I1,NATOMS
C CVMGP(X,V,Z)=X If Z.GT.0 AND Y OTHERWISE
5     DX(J,L )=CVMGP((X(I,L)-X(J,L))-T,X(I,L)-X(J,L)
      +,ABS(X( I,L )-X( J,L ))-EL2( L ))
      DO 4 J=I1, NATOMS
4     R2(J)=R2(J)+DX(J,L)**2
C MAKE R2 INTO TABLE ENTRIES WITH LMAX BEING MAXIMUM
      DO 6 J=I1,NATOMS
6     INDEX(J)=LTABT(J)+MIN0 (LMAX,INT(CSIT(J)*R2(J)))
C GATHER V(R( J ) ) INTO A VECTOR
      CALL GATHER(NC,R2(I1),PTABLE, INDEX(I1)
C SUM THEM UP
      VTOTAL=VTOTAL+SDOT(NC,R2(I1),1,EPST(I1),1)
C GATHER DERIVATIVES FROM FTABLE INTO R2
      GALL GATHER(NC,R2(I1),FTABLE,INDEX(I1),1)
      DO 7 J=I1, NATOMS
7     R2(J)=R2(J)*EPSFT(J)
C MULTIPLY BY DISPLACEMENTS AND ADD INTO FORCE VECTORS
      DO 8 L=1, NDIM
      FORCE (I,L)=FORCE(I,L)+SDOT(NC,R2(I1),1,DX(I1,L),1)
      DO 8 J=I1,NATOMS
8     FORCE(J,L)=FORCE(J,L)-R2(J)*DX(J,L)
1     CONTINUE
```

FORTRAN code, optimized for the CRAY, which performs the pairwise sum of eq. (1).

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch009

4) The summing of the force on particle I can be performed by the BLAS (8)(Basic Linear Algebra Subroutine) SDOT, which is quite efficient (about 3.5 machine cycles/element).

VAX 11-70 with attached Floating Point Systems Array Processor (120B)

The system used is that located in the Chemistry Department at Columbia University. The architecture and characteristics of the Array Processor (120B) are described elsewhere in this volume. A very careful investigation of the use of an array processor to perform Monte Carlo simulations has been done by Chester, et. al.(9) There they demonstrated that with assembly language hand coding (APAL) a simulation on the AP would run about twice as long as an CDC 7600. Here we report the timing results for the pairwise sum using APTRAN, the FORTRAN Cornell computer (Level 3.5). Because of the size of CLAMPS (3500 lines), and the intermix between calculations and I/O operations, the entire code cannot be compiled in the AP. Instead a single subroutine, which contains the coding in Table I was written, with all data passed through a common block. This routine then executes on the AP with the rest of the program executing on the host (VAX 11-70). Before each calculation of the pairwise sum, the coordinates need to be passed to the AP. After it is finished the potential and forces are then passed back to the host. The transmittal time is small (less then 10 ms), so that this mode of operation is satisfactory if there are at least 100 particles. However, the code generated by the compiler is not nearly as efficient as assembly language coding would be, the pairwise sum executes in roughly the same time as in the VAX alone. Undoubtedly restructuring of the FORTRAN would improve the execution time. The accuracy (38 bits) of the AP is sufficient for molecular dynamics. A general review of array processors and their usefulness for chemical computations is contained in Ref. (10).

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch009

Simulation Needs

Finally I would like to summarize the computational characteristics of simulations and what makes a good simulation computer.

1) A simulation is almost always cpu bound. Hence, fast floating point speed is essential.

2) Memory size can be made quite small, 20K to 200K.

3) Accuracy demands are less than in many areas of computational chemistry. Usually one needs accuracy only to one part in 10^4 in energy for Monte Carlo and one part in 10^6 for the forces in Molecular Dynamics.

4) As we have seen on the CRAY the ability to gather data together is essential. Memory speed must be commensurate with floating point speed. When nearest neighbor tables are used fast scatter operations are also needed. The two essential random memory operations needed are:

```
          B(I) = A(INDEX(I))
B(INDEX(I)) = B(INDEX(I)) + A(I)
```

5) Except for the above gather-scatter operations, simulations are easily vectorizable as defined by the CRAY FORTRAN or the CYBER 200 FORTRAN. Typical vectors have 50 to 500 elements each.

6) The Basic Linear Algebra Subroutines (BLAS) are a convenient way of maintaining efficiency and portability. They should be extended to include such things as GATHER, and vectorized EXP, SQRT, SIN and COS.

Acknowledgment

This research was supported in part by the Office of Basic Energy of Science of the U. S. Department of Energy under Contract No. W-7405-ENG-48 and by the National Science Foundation under Grant No. CHE 7721305. Special thanks to Dr. M. Rao and Prof. B. Berne for making available and assisting me in the use of the computing facilities in the Chemistry Department at Columbia University.

Literature Cited

1. Lykos, P., Ed. "Computer Modeling of Matter"; ACS Symposium Series 86: Washington, D. C., 1978.

2. Ceperley, D.; Alder, B. J.; Phys. Rev. Letts. 1980, 45, 566.3. Metropolis, M.; Rosenbluth, A. W.; Teller, M. N.; Teller, E.; J. Chem. Phys. 1953, 21, 1087.

4. Hansen, J. P.; McDonald, I. R.; "Theory of Simple Liquids"; Academic Press: New York, 1976; Chapter 3.

5. Contact the Quantum Chemistry Program Exchange, Department of Chemistry, Room 204, Indiana University, Bloomington, Indiana, 47405 for CLAMPS.

6. Valleau, J. P.; Whittington, S. G.; "Modern Theoretical Chemistry 5A"; Ed. B. Berne; Plenum: New York, 1977.

7. Rahman, A.; Stillinger, F.; Lemberg, H.; J. Chem. Phys. 1975, 63, 5225.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch009

8. The BLAS are a collection of 38 FORTRAN callable subroutines that peform many of the basic operations of numerical linear algebra. Contact International Mathematical and Statistical Libraries, (IMSL) for more information.

9. Chester, G.; Gann, R.; Gallagher, R.; Grimson, A.; "Computer Modeling of Matter" Ed. P. Lykos; ACS Symposium Series 86: Washington, D. C., 1978.

10. Ostlund, N. S.; "Attached Scientific Processors for Chemical Computations: A report to the Chemistry Community", NRCC report (1980).

RECEIVED March 18, 1981.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch009

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch010

10

The Need for Supercomputers in Time-Dependent Polymer Simulations

MARVIN BISHOP—Fordham University at Lincoln Center, New York, NY 10023

DAVID CEPERLEY—National Resource in Computational Chemistry, Berkeley, CA 94720

H. L. FRISCH—State University of New York at Albany, Department of Chemistry and Center for Biological Macromolecules, Albany, NY 12222

M. H. KALOS—Courant Institute of Mathematical Sciences, New York, NY 10012

The development of polymer science has had a profound effect on technology. Polymer materials have important applications because of their mechanical, dielectric, flow, and thermal properties. These physical properties are determined by the molecular configurations of the polymer. For example, a polymer in solution which is tightly coiled will have a lower viscosity than when it is extended because the tightly coiled structure presents a smaller cross-sectional area to the solvent molecules. Clearly, a fundamental area of polymer research is the investigation of the determining factors of molecular configurations and the mechanisms by which the polymer chain moves from one configuration to another.

A number of theoretical models have been developed. In early models (1)a polymer chain was represented by a random walk between points in space. The points represent the polymer atoms and the steps the interatomic bonds. In a pure random walk intersections are allowed. The simplification of Gaussian statistics enabled workers to calculate analytically many properties. However, the repulsive forces between atoms prevent such intersections in real polymers. The addition of excluded volume to the random walk makes the model too complicated for exact calculation. Computer Monte Carlo methods have been employed (2) to examine model polymer systems. Many models assume a lattice and hard potentials which prevent more than one chain segment from occupying the same site. The lattice models gain computational simplicity at the expense of making space discrete. The hard potentials allow one to study the effects of excluded volume on polymer properties. In this manner, it has been well established that the dependence of the average mean square end-to-end distance, $<R^2>$, and the average mean square radius of gyration, $<S^2>$, on the number of units in a polymer chain, ℓ, changes from $\ell^{1.0}$ for a chain without excluded volume to $\ell^{1.2}$ for a chain with excluded volume. These results hold for infinitely dilute systems, i.e., one chain,

0097-6156/81/0173-0135$05.00/0
© 1981 American Chemical Society

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch010

and are in agreement with the early theoretical treatment of Flory. (1) A few studies have also been done for continuous polymers and for more realistic potentials (3). All of these simulations have employed the standard Metropolis Monte Carlo (4) procedure and, as such, have provided no information about chain dynamics. Time dependent data are crucial for understanding the transport and dielectric behavior of polymers.

Dynamic Monte Carlo simulations were first used by Verdier and Stockmayer (5) for lattice polymers. An alternative dynamical Monte Carlo method has been developed by Ceperley, Kalos and Lebowitz (6) and applied to the study of single, three dimensional polymers. In addition to the dynamic Monte Carlo studies, molecular dynamics methods have been used. Ryckaert and Bellemans (7) and Weber (8) have studied liquid n-butane. Solvent effects have been probed by Bishop, Kalos and Frisch (9), Rapaport (10), and Rebertus, Berne and Chandler (11). Multichain systems have been simulated by Curro (12), De Vos and Bellemans (13), Wall *et al* (14), Okamoto (15), Kranbuehl and Schardt (16), and Mandel (17). Curro's study was the only one without a lattice but no dynamic properties were calculated because the standard Metropolis method was employed. De Vos and Belleman, Okamoto, and Kranbuehl and Schardt studies included dynamics by using the technique of Verdier and Stockmayer. Wall *et al* and Mandel introduced a novel mechanism for speeding relaxation to equilibrium but no dynamical properties were studied. These investigations indicated that the chain contracted and the chain dynamic processes slowed down in the presence of other polymers.

The nature of the static and dynamic properties of concentrated polymer systems is the focus of our work. We have already examined some static properties and are presently starting our study of the system dynamics.

Model

The potential of our model consists of two parts: a shifted Lennard-Jones potential (18), U_{LJ}, which operates between all N 'beads' in the system and a modified harmonic potential (19), U_H^P, which links ℓ beads into N chains ($N_p=\ell N$). Each 'bead' represents a statistical segment of the polymer which includes many monomers.

$$U_{LJ}(R) = \left\{\begin{array}{ll} 4\varepsilon\left[(\frac{1}{R})^{12} - (\frac{1}{R})^{6} + \frac{1}{4}\right] & R \le 2^{1/6} \\ 0 & R > 2^{1/6} \end{array}\right\} \qquad (1)$$

$$U_H(R) = \left\{\begin{array}{ll} -0.5kR_0^2\ \ell n(1-(R/R_0)^2) & 0 \le R \le R_0 \\ 0 & R > R_0 \end{array}\right\} \qquad (2)$$

The Lennard-Jones potential takes account of the excluded volume in a convenient manner in that it is repulsive, short ranged, and continuous. The harmonic potential modifies the simple harmonic potential so that the spring connectors have finite extensibility. We have set $R_0 = 1.95$ and $k = 20$. This value of R_0 makes chain crossing impossible and the value of k makes the time scale of the internal vibrational motion not much shorter than that of the translational motions so that a very small time step, Δt, is not required in the numerical integration routines. Since the relaxation of the polymer chain is a long time scale process, a small Δt implies correspondingly large computer times. In all simulations the polymers are initially placed in a lattice configuration of edge length L consistent with the pre-selected number density Np/L^3. Periodic boundaries are used in all three directions. A 'box scheme' is used in order to eliminate unnecessary calculations of pair interactions (20).

The difficulties in simulating polymer systems stem from the long relaxation times these systems display. Long runs are needed in order to ensure adequate equilibration. We have employed the method of Wall and Mandel (21) as modified for continuum three dimensional polymers by Webman, Ceperley, Kalos and Lebowitz (22). Each chain is considered in order and one end is chosen randomly as a 'bead'. Suppose the initial chain coordinates are $C = \{\vec{X}_1, \ldots \vec{X}_n\}$. A new position of that bead, $\vec{X}'$, is selected such that $\vec{X}' = \vec{X}_n + \Delta\vec{X}$ where $\vec{X}_n$ is the initial head position and $\Delta\vec{X}$ is a vector randomly chosen via a rejection technique from the probability distribution $\exp(-\beta U_H(\Delta\vec{X}))$ ($\beta = 1/k_BT$, k_B Boltzmann's constant, T the temperature) and U_H is given in Eq. (2). The trial chain has coordinates $C' = \vec{X}_2, \vec{X}_3 \ldots \vec{X}_n, \vec{X}'$ after the chain units are shifted along the chain contour. This configuration is accepted with probability equal to $\min(1, \exp(-\beta\Sigma_2^{\ell} (U_{LJ}(X_j - X') - U_{LJ}(X_j - X_1)))$ where $U_{LJ}(r)$ is non-bonding potential (Eq.1) of two beads separated by a distance r. If the move is accepted then the new chain configuration becomes C'. Otherwise it remains at C. In any case, the next chain is considered and the selection processes repeated. However, the rejected move is still considered as a configuration in subsequent averaging. This 'reptation' Monte Carlo algorithm has been incorporated into the NRCC program CLAMPS. Significant chain motions are effectuated by these 'single' moves (17). We have employed this technique to sample the Boltzmann distribution of our polymer systems.

Chain lengths of 5,10,20,32,50 and 70 beads have been studied (23). Studies at densities of 0.1,0.3 and 0.5 demonstrate that chain dimensions are compressed as the concentration is increased. Both the mean square end-to-end distance, $\langle R^2 \rangle$, and the mean square radius of gyration, $\langle S^2 \rangle$, have a power law dependence upon $\ell-1$, the number of bonds, with exponent approximately 1.16 for $\rho = 0.1$ and 1.07 for $\rho = 0.3$ and 0.5. $\langle R^2 \rangle$ and $\langle S^2 \rangle$ scale with density as $\rho^{-\gamma}$ where $\gamma \sim -0.22 \pm 0.02$ for long chains, in reasonable agreement with the scaling prediction of -0.25. The

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch010

asphericity ratios, the pair correlation functions of the center of masses, and the extent of chain overlaps indicate the nonideal behavior of these systems. These investigations give information only about static properties. However, they also provide equilibrated systems for initializing dynamics calculations.

Polymer Dynamics

We are in the process of examining time dependent properties such as the diffusion coefficient and the relaxation times of various correlation functions as a function of both chain length and density. Following the work of Kirkwood (24) and Rouse (24) we assume that the velocity of the polymer is proportional to the forces acting on it at any time; this is the high-viscosity limit in which inertial terms are neglected. Neglecting also hydrodynamic forces, we then have for the velocity of the *j*th bead at time t

$$\vec{v}_j(t) = -\beta D \nabla_j U(R) + \vec{W}(t) \qquad (3)$$

Here β is the reciprocal temperature of the solvent, D is the diffusion constant of a monomer, and W is a Gaussian fluctuating "Langevin force" (due to the solvent) with mean $\langle \vec{W}(t) \rangle = 0$ and covariance $\langle \vec{W}(t_1)\, \vec{W}(t_2) \rangle = 6D\delta(t_1-t_2)$. This leads to the Smoluchowski equation for the time evolution of the polymer probability density f(R,t),

$$\frac{\partial f(R,t)}{\partial t} = D \sum_{j=1}^{N} \nabla_j \cdot [\nabla_j f(R,t) + \beta f(R,t) \nabla_j U(R)] \qquad (4)$$

The solution of (3) approaches equilibrium as $t \to \infty$; $f(R,t) \to Z^{-1} e^{-U(R)}$.

The solution of the diffusion equation (4) was generated by a Monte Carlo random walk: Full details will be given elsewhere (25). We mention here only that our basic time step was chosen to be $\tau = 0.01$ in units of σ^2/D; distances were measured in units of σ. These calculations are being done using the NRCC program CLAMPS (26).

The dynamical trajectory generated by such a process can be visualized as a movie showing the polymer motion. Figure 1 presents a frame of an $\ell = 20$, $N = 5$ system. The duplicated chains are a consequence of the periodic boundary conditions. When part of a chain sticks out of the fundamental box, its image enters the opposite face. Rather than showing pieces of chains we have drawn complete chains for both the original and image chain. In figure 2, 50 frames are superimposed. One can see the enve-

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch010

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch010

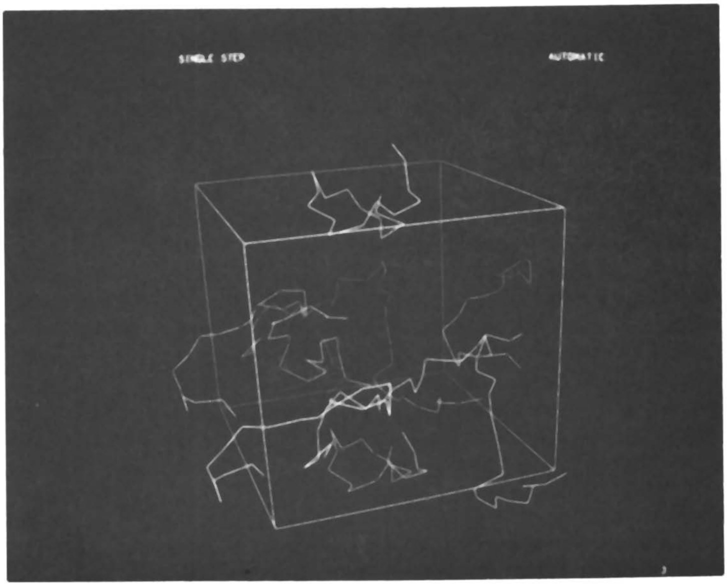


Figure 1. A frame of the $l = 20$, $N = 5$ *system. Note the periodic images.*

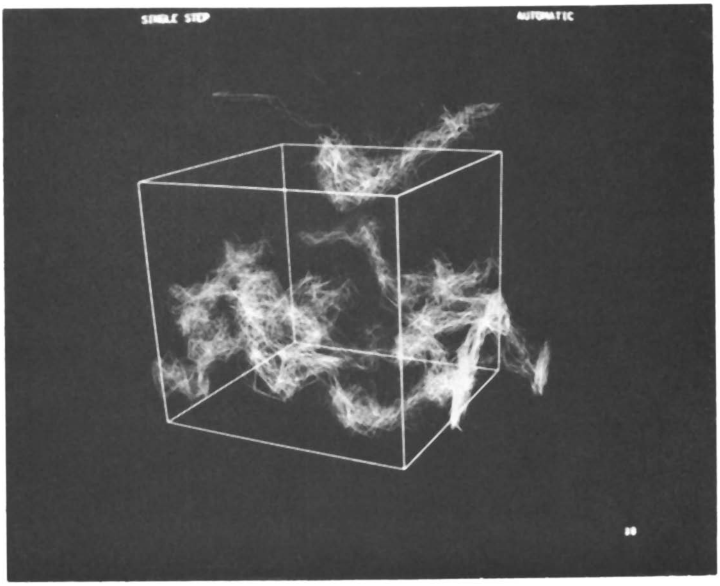


Figure 2. The superposition of 50 frames.

lope of the polymer motion. On the time scale displayed the chains exhibit mostly wagging motions and there is little large scale rearrangement.

These are the beginning of our study but already it is clear that many important polymer problems are beyond the capability of even a CDC7600. It is only with long polymers that certain properties appear. For example, it has been experimentally determined that the viscosity of almost all polymers varies as a power of polymer length and that the power changes from a value of 1 for small polymers to a value of 3.4 for long polymers (27). The change takes place at a critical length of about 10^3 monomer units. The beads in our model represent segments of the polymer chain and hence the number of beads per chain necessary to observe this transition will be much less than 10^3. However, the computer time needed to undertake this calculation is huge. The employment of a "box scheme" to eliminate unnecessary calculations of pair interactions means that the calculation time scales asymptotically as N_p. One step of a calculation with $\ell = 5$, $N = 25$ took 0.06 CPU secs on a CDC7600. Assume that we would see the onset of the viscosity law change for $\ell = 200$, $N = 25$ so that one step would require 2.4 sec. However, longer chains require many more steps for relaxation. In fact, the relaxation times scales (6) as $\ell^{2.2}$. The small system required about 5,000 steps for adequate relaxation. Thus, one would need 1.7×10^7 steps or 10^4 hours of CPU time for the larger system. Clearly such computations require the use of super computers in the gigaflop range or better. We await their development.

Conclusions

Even though great progress has been made in the application of computers to polymer science there are still problems which are unapproachable with the present generation of computers.

Acknowledgements

This research was supported in part by the National Science Foundation, the Donors of the Petroleum Research Fund, administered by the American Chemical Society, the U. S. Department of Energy under Contract EY-76-C-02-3077*000, and the National Resource for Computation in Chemistry under a grant from the National Science Foundation (NSF CHE 7721305) and the basic Energy Science Division of the U. S. Department of Energy under contract W-7405-ENG-48. We wish to thank Dr. Arthur J. Olson and Dr. T. J. O'Donnell of the NRCC graphics facility for generation of the graphics pictures.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch010

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch010

Literature Cited

1. Flory, P.J. Principles of Polymer Chemistry (Cornell University, New York, 1953).
2. Wall, F.T.; Windwer, S.; Gans, P.J. Methods in Computational Physics, (Academic, New York, 1963), Vol.I; Windwer, S. Markov Chains and Monte Carlo Calculations in Polymer Science, ed. G.G. Lowry (Dekker, New York, 1970).
3. See for example Stellman, S.D. and Gans, P.J., Macromolecules, 5, 516(1972), Grishman, R. J. Chem. Phys., 58, 220 (1973), and McCrackin, F.L.; Mazur, J. and Guttman, C.M. Macromolecules, 6, 859 (1973).
4. Metropolis, N.; Rosenbluth, A.W.; Rosenbluth, M.N.; Teller, A.H. and Teller, E. J. Chem. Phys, 21, 1087 (1953).
5. Verdier, P.H.; and Stockmayer, W.H. J. Chem. Phys., 36, 227 (1962).
6. Ceperley, D.; Kalos, M.H.; and Lebowitz, J.L. Phys. Rev. Lett., 41, 313 (1978).
7. Rychaert, J.P.; and Bellemans, A. Chem. Phys. Lett., 30, 123 (1975).
8. Weber, T.A. J. Chem. Phys., 69, 2347 (1978).
9. Bishop, M.; Kalos, M.H.; and Frisch, H.L., J. Chem. Phys. 70, 1299 (1979).
10. Rapaport, D.C. J. Chem. Phys., 71, 3299 (1979).
11. Rebertus, D.W.; Berne, B.J.; and Chandler, D. J. Chem. Phys., 70, 3395 (1979).
12. Curro, J.G., J. Chem. Phys., 61, 1203 (1974).
    ——————, J. Chem. Phys., 64, 2496 (1976).
13. De Vos, E.; and Bellemans, A. Macromolecules, 7, 812 (1974).
    ——————————————— Macromolecules, 8, 651 (1975).
14. Wall, F.T.; Chin, J.C.; and Mandel, F. J. Chem. Phys., 66, 3143 (1977).
    Wall, F.T.; and Seitz, W.A. J. Chem. Phys., 67, 3722 (1977).
15. Okamoto, H. J. Chem. Phys., 70, 1690 (1979).
16. Kranbuehl, D.E.; and Schardt, B. Computer Modeling of Matter, ed. P. Lykos, ACS Symposium Series 86, 1978.
17. Mandel, F. J. Chem. Phys., 70, 3984 (1979).
18. Weeks, J.D.; Chandler, D.; and Anderson, J.C. J. Chem. Phys., 54, 5237 (1971).
19. Armstrong, R.C., J. Chem. Phys., 60, 724 (1974).
20. Quentrec, B.; and Brot, C., J. Comput. Phys. 13, 430 (1973).
21. Wall, F.T.; and Mandel, F., J. Chem. Phys., 63, 4592 (1975).
22. Webman, I.; Ceperley, D.; Kalos, M.H., and Lebowitz, J.L.; (to be published).
23. Bishop, M.; Ceperley, D.; Frisch, H.L.; and Kalos, M.H., J. Chem. Phys., 72, 3228 (1980).
24. Rouse, P.E., J. Chem. Phys., 21, 1272 (1953).
    Zimm, B.H. J. Chem. Phys., 24, 269 (1955).
25. Ceperley, D.; Kalos, M.H.; and Lebowitz, J.L. to be published, Macromolecules.

26. CLAMPS was written with support from the National Resource for Computation in Chemistry and is available from the NRCC for distribution.
27. Porter, R.S. and Johnson, J.F. Chem. Rev., 66, 1 (1966).

RECEIVED April 14, 1981.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch010

11

Applications of Large-Scale Computers and Computer Graphics

Investigations of Biological Macromolecular Structure, Function, and Evolution

ARTHUR M. LESK
Fairleigh Dickinson University, Teaneck, NJ 07666

KARL D. HARDMAN
IBM Corporation, Thomas J. Watson Research Center, Yorktown Heights, NY 10598

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch011

Crystal-structure determinations provide atomic coordinates of proteins, nucleic acids, and viruses. Computational studies of these data -- using both purely-numerical techniques and interactive graphics -- seek the principles of structure, dynamics, function and evolution of living systems at the molecular level.

How can the new generation of computers and the new generation of molecular biologists interact most effectively? Certain algorithms and software now in use are mature, and will be applicable to a library of structures that is increasing progressively in scope and quality. We anticipate two effects of the introduction of very large, fast computers: Certain tasks, in which simple computational power is the limiting resource, will become feasible. Other tasks, which must now be run in "batch" mode, will achieve sufficiently fast execution times to make it possible to run them interactively.

To achieve the optimal division of labor between human and computer, it will be necessary to improve the channels of communication between them. This will require careful design of:
(1) The structure of the data base. It must have the flexibility to assimilate the results of investigations in progress.
(2) Interactive graphics systems. The increased power of host computers and display devices can easily overwhelm the human participant in the interactive execution of a program.

Current computational investigations of proteins treat:
(1) Structures. The identification of paradigms of conformation and the study of their evolution. (2) Thermodynamic stability. A protein as a three-dimensional jig-saw puzzle; its sidechains fit together snugly, excluding water. (3) The pathway by which proteins fold spontaneously. (4) Flexibilities of conformations. (5) Interactions with small molecules, other proteins, and other macromolecules.

We shall discuss the effects of increasing size and power of computers on our ability to address these problems.

0097-6156/81/0173-0143$05.00/0
© 1981 American Chemical Society

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch011

Background

X-ray crystallographers have now determined the structures of approximately one hundred biological macromolecules -- proteins, nucleic acids, and viruses -- to atomic resolution. These investigations have demonstrated that, unlike synthetic polymers, the biological molecules have specific three-dimensional conformations. Indeed, all information required to specify the structure of a protein is contained in the sequence of amino acids, and therefore the structure is also implicit in the sequence of nucleotides in the DNA or RNA genome. Analysis of the structures has provided explanations of their biological functions, and has revealed that there are recurrent architectural themes in their design (1, 2).

It is worth emphasizing the thermodynamic dilemma that nature has faced in generating, in the proteins, a set of molecules such that (1) each one will take up a specific conformation (under appropriate conditions of solvent and temperature), so that it will have reliable and reproducible functional properties, but (2) that the same basic chemical structure must be compatible with the spontaneous formation of a great variety of molecular structures and functions, so that the molecules can evolve by means of small changes. The potential for variety requires a flexible chain that can fold in many possible patterns. Each individual molecule is thereby forced to pay a high thermodynamic price for the fixation of its degrees of internal rotational freedom in the active structure. The thermodynamic stability of the functional states of biopolymers is achieved through subtle interactions among subunits and between the polymer and the solvent.

Much of the succeeding discussion will emphasize the case of globular proteins, because many more structures of this class are available, and because the quantity and variety of computational studies of proteins is especially large.

The sets of atomic coordinates of protein structures provide the raw material for a number of investigations aimed at elucidating the principles of protein architecture, the mechanism of folding, the dynamics of the structures (including the mechanism of function, which may be thought of as the dynamics of the interaction among proteins, substrates and cofactors) and the mechanism of protein evolution. The purpose of this article is to analyze and classify the kinds of studies now in progress in our own and other laboratories, and the kinds of questions that people would like to ask but are currently unable to answer.

Our basic question will be: What computational tools will produce the most effective progress of molecular biology? We feel that what is required is not only increased power, but increased sophistication in the design of the channels of access to this increased power.

Most current studies are largely descriptive: What do proteins look like? In which respects and to what extent do two structures resemble each other? Other studies are analytic or predictive: Can we calculate the forces that stabilize the structure? If so, can we use our knowledge of these forces to predict the thermodynamically stable state of a protein from its amino acid sequence? (The spontaneity of the formation of the native state proves that nature has an algorithm for this process.)

All these investigations depend on both high computer power and sophisticated software. For example, conformational energy calculations have the [ultimate and as yet unrealizable] goal of determining the global minima of nonlinear functions of large numbers of variables: the values of the thousands of atomic coordinates that correspond to the minimum conformational free energy (3, 4). It is unsurprising that no general solution is achievable even with the expenditure of large amounts of computer time. More restricted simulations, in which the molecule stays in a local region of its phase space, have produced interesting results, but even these require heavy calculations (5, 6).

The descriptive, classificatory and comparative approach to analysis of structures also depends on computing. It is no longer feasible to pursue these studies with physical models. Beyond the obvious mechanical problems, there is the logistical catastrophe: the amount of space and materiel increases linearly with the number of structures to be examined. In addition, there is no way to save and restore structures. It is therefore necessary to apply computer graphics to draw representations of structures. The importance of supercomputers in studies of this kind will lie not only in the feasibility of large calculations that are not possible at present, but in the conversion of many tasks from batch mode to interactive.

Although the distinction between analytic and predictive studies is useful, we do not suggest that it is possible to divide the field into graphics problems on the one hand and "dark" number-crunching problems on the other. Graphics is necessary to report the results of dynamics calculations: the computer-generated movies of R. Feldmann, in collaboration with M. Levitt and M. Karplus, show how difficult it would be to extract the information they contain from a program in any other way. And, of course, graphics is useful in preparing and checking the initial state of a system prior to such a calculation, and in monitoring its progress. Conversely, the analysis of a static structure can involve extensive numerical calculations, especially those involving the description of the occlusion of internal surfaces and the extent of accessibility of different residues to solvent (7, 8).

There emerges from these considerations a kind of triangular structure of the activities: at one corner is the scientist, at a second the "brute" force of a large and powerful computer, and at the third are graphics devices. One theme of this article is that a properly designed system must pay attention to the special talents of each member of this partnership, dividing the labor in an optimal way, and must provide channels of communication of adequate capacity among them.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch011

Supercomputers offer potential for progress in the field of computational molecular biology, in that they can permit more extensive experiments in the compute-bound areas such as molecular dynamics, and quicker response times for more complicated tasks in interactive work. But the increased power they promise creates challenges for the system designer, to enable the scientist to utilize the power most effectively. With many tasks, the larger computational power generates a larger quantity of results. If these are to remain comprehensible, they must be presented to the scientist in an intelligible form, and in a managable size. This will generally require pictorial output -- drawings of structures, or graphs or charts -- rather than pages of numbers.

Plan of This Presentation

After a brief review of some basic vocabulary useful in describing protein structure, we should like to discuss the following:

(1) Computational aspects of structure determination of biological macromolecules. This has important implications about the expected quality of the final results.

(2) What kinds of questions do we and others want to study? What kinds of tools do we need to help us answer them?

(3) A survey of existing software for analysis of structure, function and dynamics.

(4) Design considerations for systems based on supercomputers.

Protein Conformation

Chemically, proteins contain linear chains of amino acids, linked by peptide bonds. The twenty side chains that can occur in globular proteins differ in size, shape, charge, and polarity. A protein is in some respects like a jigsaw puzzle, in which the pieces of the molecule fit together in specific ways to create the native structure.

The peptide linkages between amino acids form the primary structure of the protein. The primary structure is all that the nucleotide sequence of the genetic material determines, and therefore the primary structure contains all information necessary to specify the complete three-dimensional conformation.

Because the peptide group tends to be planar, the backbone of the protein has two angles of internal rotation per monomer unit. Side chains contribute other degrees of conformational freedom. Steric interactions limit the allowed ranges of the conformational angles of the backbone. Within the allowed regions of conformation space, certain structures are stabilized by hydrogen bonding and other interactions. These include the α-helix and β-sheet. These types of structures are therefore utilizable as standard

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch011

subunits in protein structures. In some proteins, disulfide bonds between cysteine residues hold two regions of the chain together.

The hydrogen-bonded structural units, plus the disulfide bonds, constitute the secondary structure of the protein.

In the complete structure, secondary-structured regions are assembled into compact units. The interactions between secondary structural units seem to be stereochemically specific. This is called the tertiary structure of the protein. (Certain common patterns of interaction between secondary structural units are known as supersecondary structures.)

The assembly of two or more independently-folding polypeptide chains into a complete protein is known as its quaternary structure.

There is some reason to suspect -- although this assumption has not been proved -- that proteins fold through the formation of secondary structural units, which assemble by accretion, each combination of structured units lending additional stability to the nascent globule (9, 10). This suggests that a possible approach to the prediction of a protein structure might be to attempt to predict secondary structure first, and then search for favorable interactions between units (11, 12). It should be emphasized, however, that any failure of such calculations does not disprove the model for folding, and any success would not confirm it.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch011

Sources of the Stability of Proteins Under Physiological Conditions of Solvent and Temperature

Several factors compensate the protein-solvent system for the loss of chain entropy required by folding:

(1) Hydrogen bonding. Because peptide units form hydrogen bonds to water in the unfolded state, peptide-peptide hydrogen bonds must form in the folded state to avoid an otherwise intolerable loss of enthalpy (13).

(2) The globular structures of proteins reduce the amount of surface area of the residues that can come into contact with solvent. Although water itself is a relatively highly-ordered liquid water molecules surrounding a solute molecule can be even more highly-ordered than they are in pure water. Therefore the release of water upon burying residues into the interior of proteins contributes a positive entropy change to the folding process (14).

(3) A fairly dense packing of the atoms in the protein interior enhances the contribution of van der Waals interactions to the stability (15).

Structure Determination

We are concerned primarily with investigations that begin where structure determination leaves off. However, we must consider the quality of the results of structure determinations, which prescribes the kinds of computations we can credibly undertake.

The basic steps in a contemporary protein structure determination are these (16):

(1) The material is isolated in its "native" form.

(2) Crystals of the material are grown, and isomorphous derivatives are prepared. (The derivatives differ from the parent structure by the addition of a small number of heavy atoms at fixed positions in each -- or at least most -- unit cells. The size and shape of the unit cells of the parent crystal and the derivatives must be the same, and the derivatization must not appreciably disturb the structure of the protein.) The relationship between the X-ray diffraction patterns of the native crystal and its derivatives provides information used to solve the phase problem.

(3) Sets of X-ray diffraction data are collected on native crystals and derivatives.

(4) By combining the intensity patterns of diffraction from native crystals and derivatives, it is possible to generate a rough map of the electron density distribution. This will not be expected to have well-resolved atomic peaks. However, in favorable cases it will be possible to trace most of the chain; and certain prominent sidechains should be visible, for example, tyrosine "lollypops".

(5) A model of the structure must be fit to the map. This used to be done with the "Richards box", a device containing a half-silvered mirror to give the illusion of superposition of a physical model and the electron density map (conventionally contoured in serial sections, traced onto transparent film, and stacked (17).) Recently, some protein structures have been determined using interactive computer graphics to fit a stick model of a structure to an electron density map (18, 19).

(6) When a correspondence has been established between peaks in the electron density and atomic positions of the model, with the r.m.s. deviation of the α-carbon atoms from their true positions below about 0.5 Å, it is feasible to begin some kind of refinement procedure. A refinement procedure is a method for calculating and checking adjustments in the atomic coordinates in the model, to minimize some measure of its inaccuracy (20). The classic measure of accuracy is the residual, or "R-factor":

$$R = \frac{\Sigma \left| \, |F_o| - |F_c| \, \right|}{\Sigma |F_o|}$$

in which $|F_o|$ and $|F_c|$ are corresponding observed structure factor magnitudes and those calculated from the model, and the summation estends over that portion of Fourier space for which data were collected. (The structure factors are the Fourier coefficients of the electron density distribution. Although both the magnitude and the phase of the structure factors F_c of the model are calcul-

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch011

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch011

able, only the magnitudes of the observed structure factors F_o are measurable. Therefore the comparison between the model and the experimental data can involve only the structure factor magnitudes.)

Some refinement procedures include, at certain stages, criteria for stereochemical regularity in addition to the agreement between observed and calculated structure factor magnitudes.

Recent Advances in Refinement Techniques for Macromolecules

One of the primary goals of crystallographic studies of biological macromolecules is to extend the methods far enough for reliable determinations of atomic bond lengths, particularly in active site regions. In the past, for the determination of models of proteins as large as 100 amino acids, it has been necessary to retain standard stereochemistry (fixed bond lengths and angles) throughout the entire procedure (21). It is now forseeable that in the final stages of structure determination these geometric constraints may be relaxed or even eliminated entirely, to give an objective determination of structure. Some important recent advances are experimental: low-temperature methods, and the development of areadetectors for diffraction methods. Others are computational: in particular, advances in the power of crystallographic refinement techniques for macromolecules.

For example, Agarwal has developed a novel approach to least-squares refinement which provides extremely fast computation times and remains practical for larger macromolecules (over 2500 nonhydrogen atoms) at very high resolution (better than 1.2 Å) (22). The method utilizes algorithms developed for digital signal processing and optimization techniques, with fast-Fourier transform procedures for calculation of structure factors and gradients for predicting the three-dimensional atomic shifts. Computaton times are approximately proportional to n log n per cycle, where n is the number of structure factors. This is an important advance over the n^2 dependence of computation time on number of parameters that characterizes older methods. The 2-Zn form of insulin has been refined to 1.5 Å resolution (more than 1100 nonhydrogen atoms and approximately 12000 structure factors) to an R factor of 0.113 (23). Other computational experiments demonstrate the utility of this method at both higher resolutions (the antibiotic beauvericin at 1.2 Å (24)) and lower ones (sperm whale myoglobin at 2.0 Å (25)).

It is evident that improved methods of data collection will make very high resolution studies of biological macromolecules possible. The speed and size of current computers are already adequate to refine atomic models of medium-sized proteins to higher resolution than most protein crystals diffract, without the need to impose stereochemical constraints at the final stages of the refinement. For example, with the Agarwal refinement program, myoglobin can be refined using a complete set of crystallographic

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch011

data at 1.0 Å nominal resolution (approximately 70,000 independent structure factors) in less than 4 minutes of computer time per cycle, on an IBM 370/3033 (25). Improvements currently being implemented by R.C. Agarwal and others will reduce this time to between 1 and 2 minutes (26). Looking to the next generation of computer hardware, it is apparent that even these calculations will require less than 10 seconds per cycle. Thus the accuracy of the models for macromolecules obtainable will depend only on the quality of the crystals and of the measurements of their diffraction patterns.

These results imply that it is already possible to provide a variety of statistics for evaluation of the precision of the atomic coordinates. These statistics can be used to judge whether the structure determination has achieved an acceptable level for credible interpretation of function. Realistic goals for precision of individual bond lengths are overall r.m.s. deviations of approximately 0.1 Å for proteins of moderate size with data to 1.5 Å (and better for data nearer 1.2 Å). In some cases, the "interesting" part of the protein is more rigid in structure than the average -- for example, organometallic complexes within proteins, or tightly-bound substrates -- and in these cases the precision of individual bond lengths and angles might well be higher than the average.

The accuracy of the model is a more difficult question, and criteria for accuracy are yet to be established. At some point, at very high resolution, it will be necessary to prove that atypical stereochemical features are indeed real, by refining the models with relaxed geometrical constraints or none at all. This must be a major goal of crystallographic studies if correlations of structure and function are to be meaningful.

The results of structure determinations provide the core of a data base of biological macromolecular structure applicable to further investigations. Currently, the Protein Data Bank at the Chemistry Department of Brookhaven National Laboratories has responsibility for archival storage and distribution of these rssults in the United States of America (27). Later in this paper we shall discuss howsystems might be designed to facilitate access to such a data base.

What kinds of questions are we asking? What computational tools do we have? (To be followed by: What kinds of questions would we like to ask? What kinds of tools do we need?)

It is obvious that we cannot review exhaustively a field as active as computational protein chemistry is today. All that we can hope to do is to classify some of the approaches to problems currently recognized as (or at least deemed by consensus to be) interesting ones.

One possible approach to such a classification is based on the range of conformations treated in the investigation. Thus we might be led to distinguish:

(1) Studies of static conformations, including all studies of protein architecture per se, and some studies of the interactions of small molecules with proteins.

(2) Studies of local conformational deformations, including the response of the macromolecule to the binding of substrate in descriptions of function, or the interconversion of conformations in allosteric proteins.

(3) Studies of transitions from the unfolded to the folded state, involving global conformational rearrangements.

Investigations of static structures include architectural descriptions, comparisons and classifications, and identifications of recurrent patterns such as supersecondary structures. Examples include the classification of types of proteins by Levitt and Chothia (28), the hierarchical analysis of protein structures by Rose (29), or the comparison of globin structures by Lesk and Chothia (30).

Other computational studies of static structures seek to analyze the contributions to the thermodynamic stability of the native state. These studies include the detection of hydrogen bonds, packing patterns, and analysis of buried surface area.

Examples of studies of local conformational dynamics include the films made by Richard Feldmann, in collaboration with M. Levitt and with M. Karplus, which show the dynamics of pancreatic trypsin inhibitor and its interaction with solvent, and the study by Case and Karplus of the pathway by which an oxygen molecule can enter and leave the binding pocket of myoglobin (31). (In the static structure, there is no stereochemically feasible path for binding oxygen -- the process requires a distortion of the protein structure.)

No attempt to determine the folding pathway of a protein from the denatured to the native state either by energy minimization or by molecular dynamics has yet been successful. Many people believe that it would be more effective to approach the problem through a study of the formation and interaction of secondary structural units. A conservative appraisal of the situation is that this problem is not one that can be solved by increase in computer power alone.

Software tools for analysis of static structures include:
(1) Graphics programs that permit scrutiny, dissection and manipulation of structures.

Computer-generated representations of macromolecules fall into three broad categories: (a) line drawings (line segments represent chemical bonds), including the familiar "ball-and-stick" models, and OR-TEP drawings. (b) Drawings of space-filling models such as simulations of CPK models, or representations of molecular surfaces (Figure 1). (c) Schematic diagrams, or cartoons, in the most common variation of which cylinders represent helices and arrows represent strands of sheet (Figure 2). Originated by A. Rossmann, as artists drawings, there now exist computer programs to create them.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch011

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch011

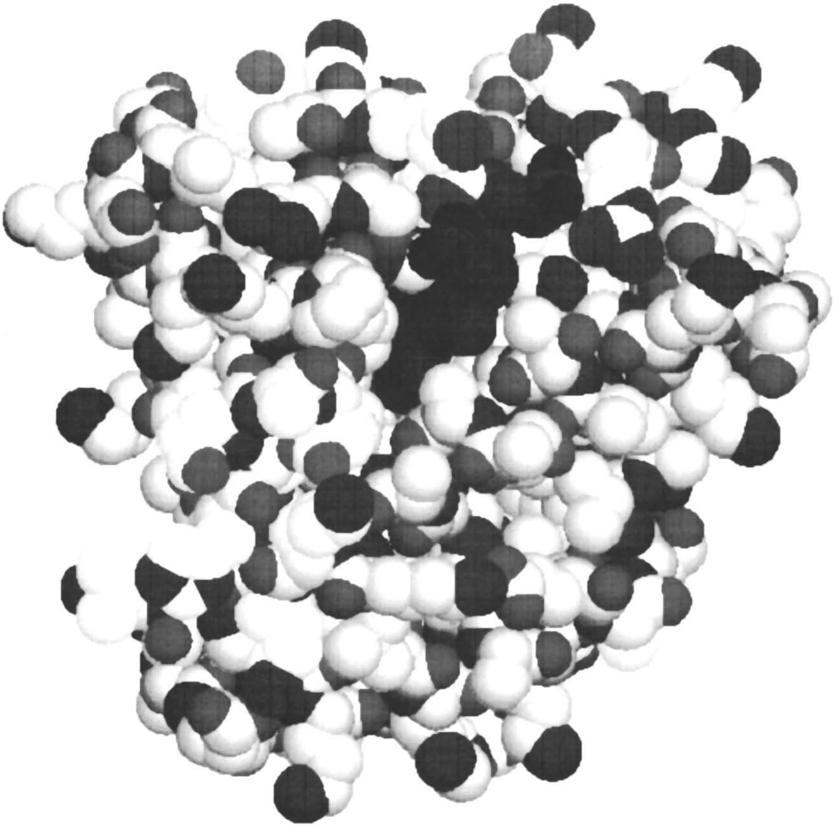
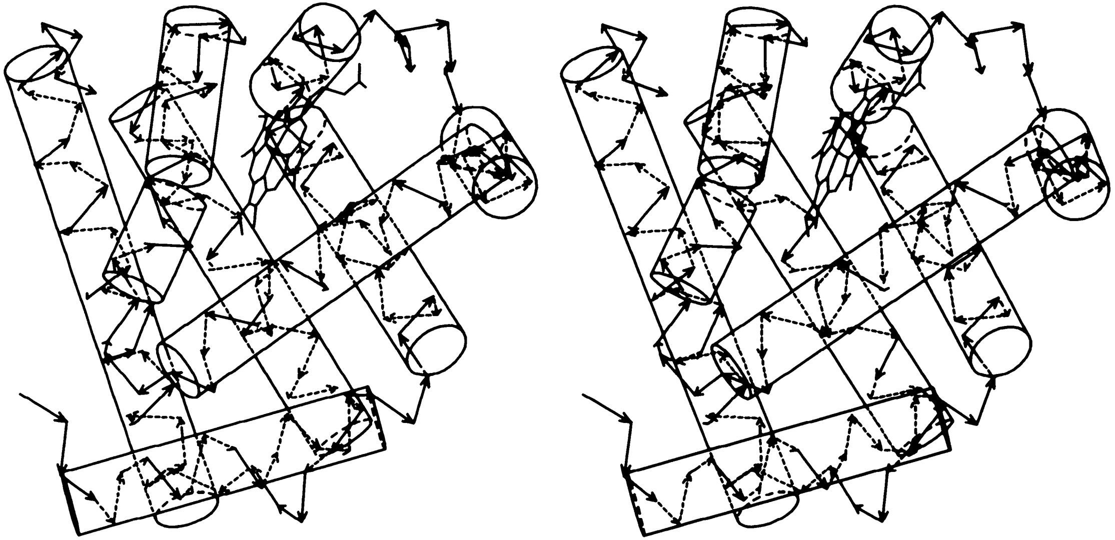


Figure 1. A space-filling model of sperm whale myoglobin.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch011



SPERM WHALE MYOGLOBIN **SPERM WHALE MYOGLOBIN**

Figure 2. A schematic diagram of sperm whale myoglobin. Cylinders represent α-helices. Figures 1 and 2 show the molecule in the same orientation, looking into the heme pocket.

Each representation of a protein or nucleic acid conveys to the viewer different aspects of its structure: line drawings give the bones, space-filling models the flesh, and schematic diagrams the gestalt of the design. No single representation of a protein or nucleic acid is adequate for all purposes, but the combination of several is more powerful than the total of all taken independently.

Line drawings are effective at showing the chain folding, and can indicate spatial relationships between a few selected groups such as the sidechains interacting with a prosthetic group or substrate. The entire molecule is visible. Space-filling representations give a better idea of the packing of the atoms, but special techniques are required to delve beneath the surface, such as "cheese-wire" sections, or rendering of the atoms on surface layers as translucent rather than opaque. The main disadvantage of simulated CPK models is the difficulty of analyzing a picture of an entire protein. It is useful to have a schematic diagram alongside a space-filling representation, to aid in its interpretation.

(2) Analysis of structural patterns

Given the standard geometries of hydrogen-bonded atoms, or of helices and sheets, it is possible to apply pattern-recognition techniques to analyze the conformation of a protein. This has been done (many times) and is a useful approach to the extent that the problem is well-defined. However, in many cases helices deviate from standard geometry -- especially at the ends where they can tighten up or unravel. Any investigation that requires analysis of this kind of detail is better pursued interactively.

(3) Estimates of stabilization energies

Conformational energies are analyzable both in terms of pairwise interactions among atoms (the Coulomb and van der Waals contributions) and by the calculation of surface area accessible to solvent, based on methods pioneered by Lee and Richards (32). Wodak and Janin have recently improved the computational technique for estimating accessible surface areas (33).

Programs that explore conformation space -- either locally or globally -- require some estimate of conformational energy. Generally this is done by deriving parameters for atom-atom interactions which reproduce the thermodynamic and structural parameters of crystals of small organic molecules. These parameters define the conformational energy surface in the vicinity of any state, if interactions with solvent are neglected (34).

In molecular dynamics calculations, a program determines solutions of the classical equations of motion, starting from some set of initial velocities (that define the temperature of the system, assuming it to be isolated). By following the motion of the system over a period of time, starting from a nonequilibrium state,

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch011

it is possible to study the approach to equilibrium. By following the motion of the system in an equilibrium state, it is possible to calculate equilibrium properties, and to study fluctuations. Simple energy minimization can be thought of as molecular dynamics at 0 K, in a medium of infinite viscosity.

We have described some of the software that already exists and which already is applicable to investigations of protein structure. But if these calculations can already be accomplished, why do we need supercomputers? In some cases, such as molecular dynamics, the question is one of brute number-crunching power -- a faster computer will permit simulations of the motions of larger molecules over longer time intervals. But in other cases the question is not one of feasibility vs. infeasibility of the calculation, but in the speed with which the calculation can be completed relative to the time scale of interactive computing. Supercomputers may, in some cases, be able to convert certain tasks that must currently be run in batch mode to tasks that can be run interactively. This opens new doors.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch011

What kinds of questions would we like to be able to ask? What kinds of tools do we need?

The increased power of the next generation of computers will permit many tasks that must now be executed as batch runs to be performed interactively. This implies that it will be necessary to design an interface between a human operator and the set of atomic coordinates that permits the interaction to be effective. Although many algorithms currently in use are adequate for this purpose, much of the actual software is not supple enough to be easily adaptable to a variety of tasks. We envisage a system in which the point of view of the user has shifted to the idea of retrieving information from the data base, rather than performing calculations on one or more proteins.

We realize that if we could suggest a complete set of "primitive questions", we would be performing a useful service. However, we feel that we cannot anticipate all the questions that we or anyone else will want to ask. (Conversely, we will not concede that anyone else can anticipate all the questions that we will want to ask.) Any generally useful system will therefore have to be adaptable to the needs of different users.

Certain categories of questions are fairly obvious, however. These include:

(1) Identification of structural units. This involves the search for secondary structural units (more precisely, for units that are superposable on standard units to within a specified error), or for combinations of them. The operator will want to specify the range of the search -- within a certain molecule, or over a class of proteins, or over the entire available set of coordinates. He or she will also need to be able to qualify the object of the search, specifying perhaps a range of lengths of

helix, or perhaps to ask for two helices in contact, with interaxial distances and angles in specified ranges.

(2) Analysis of any selected unit. What is close to what? (Here what = atom or residue or secondary-structural unit.) Where are the hydrogen bonds? What surface is accessible to solvent? (both in an excised fragment of a molecule per se, and also the value that would characterize the fragment within the entire protein.)

(3) Conformational energy estimates. What, approximately, is the cohesive energy of a selected unit?

(4) The geometry of interfaces -- including the binding of substrates, effectors and drugs. What is the nature of the packing at an interaface? Is there room for another methyl group? How does a substrate fit into a pocket? How would a known drug, or a molecule that someone is trying to fashion into a drug, fit into a pocket?

(5) Manipulation of conformations. This must include combinations of (a) manual changes to individual sidechains and large-scale movements of helices (for example) and (b) energy minimization and molecular dynamics.

Clearly this kind of question can easily go beyond the limits of interactive computing even with a supercomputer. Two examples of difficult problems that have not been solved yet because of computational limitations are:

(a) Given a protein, such as human hemoglobin, that undergoes an allosteric change upon binding of a small molecule. Take the molecule in the unliganded form, insert the ligand without changing the protein conformation, and follow the natural course of the allosteric change.

(b) Given two secondary-structural units such as helices or sheets. Determine all possible complexes with estimated cohesive energies greater than some threshold.

Design Considerations For Data Bases

It is clear from these examples that both numerical/textual processing, and versatile and high-quality interactive graphics, both for input and for output, must be a component of any system for interactive studies of biological macromolecules that can take advantage of the power of supercomputers. It is also clear that it would be futile to attempt to design a system that would satisfy every investigator. It is our hope that the structure of the data base could be designed so that the system is adaptive -- moldable by any user to his or her individual needs at the moment. The data base should have the potential for assimilating the results of working sessions, to facilitate continuing the study. Indeed, ideally one could provide the user with a "virtual data base" -- one that is configurable within a work session to satisfy the special needs of any job of any user.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch011

The goal is to help supercomputers to permit the achievement of excellent results by ordinary scientists.

The Human Component (35)

We have emphasized the role of a human being as a participant in the execution of a computer program (as distinct from his or her other roles as programmer or operator of a computer or a terminal (36).) There are a number of considerations governing the optimal design of an interactive system involving a supercomputer which relate specifically to the needs of the human being. Although these problems occur to some extent in all interactive computing, we mention them here because (1) in general they have received inadequate attention and (b) in particular there is reason to fear that they will become more severe with a supercomputer.

It is not uncommon to find that a human being working at a computer terminal is under physiological and psychological stress. Certain factors contributing to the stress are obvious (these may include: crowded and noisy conditions, extremes of temperature, uncertainty over response time, limited duration of access to equipment, deadline for finishing work, lengthy travel required to reach the site of special facilities). Other factors are more subtle, and are not well-understood. [A peculiar type of impatience afflicts some scientists when attached to computers, preventing optimal response -- by the scientist -- to even simple problems when they arise unexpectedly. There seems to be a difficulty in switching from the implementation of solutions to problems (which can be performed effectively in partnership with a computer), to the planning of the solutions (for which the computer often not only fails to be a help but is a serious distraction.) The analogy to "... two spent swimmers, who do cling together and choke their art", is apt and will be recognized, ruefully, by numerous readers.]

Some of the factors causing stress arise from "managerial decisions" concerning allocation of space, choice of peripherals, operation of the monitor system, priorities; the usual justification being the cost of improvements. Other stress factors arise from sloppy programming techniques (for example, rigid and overly complicated input conventions, or unclear or absent error diagnostic messages.) Within the community of scientist-programmers, these are the factors over which we can exercise the greatest control.

Stress degrades the performance of tasks involving creative thought. If a human being is to contribute effectively to a partnership with a computer, he or she should be allowed to function without handicaps. Therefore the causes of stress should be identified and, as far as possible, removed. Effective partnership with a supercomputer will require even higher levels of intellectual performance from the human. Therefore it will more-imperatively require the reduction of stress.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch011

To the extent that pleasant and relaxed working conditions are luxuries, simplistic financial arguments justify refusing them (or reserving them as rewards), and complaints of their absence are dismissable as hedonistic. But what if they are not luxuries? This is the question we wish to raise.

Work supported in part by National Science Foundation grant PCM80-12007.

Literature Cited

1. Schulz, G.E.; Schirmer, R.H. "Principles of Protein Structure"; Springer-Verlag: New York, 1979.
2. Dickerson, R.E.; Geis, I. "The Structure and Action of Proteins"; Harper & Row, New York, 1969.
3. Némethy, G; Scheraga, H.A. Quart. Revs. Biophysics 1977, 10, 239.
4. Levitt, M.; Warshel, A. Nature 1975, 253, 694.
5. McCammon, J.A.; Gelin, B.R.; Karplus, M. Nature 1977, 267, 585.
6. Levitt, M. in "Protein Folding"; R. Jaenicke, ed.; Elsevier/North-Holland Biomedical Press, Amsterdam, 1980; p. 17.
7. Richards, F.M. Ann. Rev. Biophys. Bioeng. 1977, 6, 151.
8. Richards, F.M. Carlsberg Res. Commun. 1979, 44, 47.
9. Karplus, M.; Weaver, D.L. Nature 1976, 260, 404.
10. Baldwin, R.L. Trends Biochem. Sci. 1978 3, 66.
11. Richmond, T.J.; Richards, F.M. J. Mol. Biol. 1978, 119, 537.
12. Lesk, A.M.; Chothia, C. Biophys. J. 1980, 32, 35.
13. Pauling, L.; Corey, R.B.; Branson, H.R. Proc. Nat. Acad. Sci. 1951, 37, 205.
14. Kauzmann, W. Adv. Prot. Chem. 1959, XIV, 1.
15. Chothia, C. Nature 1975, 254, 304.
16. Blundell, T.L.; Johnson, L.N. "Molecular Biology: Protein Crystallography"; Academic Press: London, 1976.
17. Richards, F.M. J. Mol. Biol. 1968, 37, 225.
18. Diamond, R. "Bilder User's Manual"; Laboratory of Molecular Biology, Cambridge, England, 1978.
19. Jones, T.A. J. Appl. Cryst. 1978, 11, 268.
20. Ahmed, F.R., ed. "Crystallographic Computing", Munksgaard, Copenhagen, 1976, section B4.
21. Diamond, R. Acta Cryst. 1966, 21, 253.
22. Agarwal, R.C. Acta Cryst. 1978, A43,791.
23. Isaacs, N.W.; Agarwal, R.C. Acta Cryst. 1978, A34, 782.
24. Geddes, A.; Hardman, K. MS. in preparation.
25. Hardman, K. in "Interaction Between Iron and Proteins in Oxygen and Electron Transport"; Chien Ho, ed.; Elsevier/North-Holland Biomedical Press, 1981.
26. Agarwal, R.C. Personal communication.
27. Bernstein, F.C.; Koetzle, T.S.; Williams, G.J.B.; Meyer, E.F. Jr.; Brice, M.D.; Rodgers, J.R.; Kennard, O.; Shimanouchi, T.; Tasumi, M. J. Mol. Biol. 1977, 112, 535.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch011

28. Levitt, M.; Chothia, C. Nature 1976, 261, 552.
29. Rose, G.D. J. Mol. Biol. 1979, 134, 447.
30. Lesk, A.M.; Chothia, C. J. Mol. Biol. 1980, 136, 225.
31. Case, D.A.; Karplus, M. J. Mol. Biol. 1979, 132, 343.
32. Lee, B.; Richards, F.M. J. Mol. Biol. 1971, 55, 379.
33. Janin, J.; Wodak, S. Proc. Nat. Acad. Sci. 1980, 77, 1736.
34. Levitt, M. J. Mol. Biol. 1974, 82, 393.
35. Brenner, A. Datamation 1977, 23, 283.
36. Lesk, A.M. Comp. Biol. Med. 1977, 7, 113.

RECEIVED July 20, 1981.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch011

Computer Simulation of Biomolecular Systems

P. DAUBER, MURRAY GOODMAN, and A. T. HAGLER
University of California, San Diego, Department of Chemistry, La Jolla, CA 92093
D. OSGUTHORPE, R. SHARON, and P. STERN
Weizmann Institute of Science, Department of Chemical Physics, Rehovot, Israel

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch012

While the study of biomolecular systems by computer simulation has been contributing to our understanding of the mechanics and energetics of these systems for fifteen years, we are now at the threshold of a new era in this exciting field. It has come about by the convergence of two important developments. The first is the advent of powerful current generation computers. The second is the evolution of and improvements in simulation techniques. The latter include adaptation of Monte Carlo and molecular dynamics techniques, long used in statistical physics and applied mathematics, to biomolecular systems. These techniques become viable options only with the high speed of modern computers, as they require immense amounts of computation.

Biological systems are characterized by extreme complexity, both structurally and dynamically. An example of the type of system we would like to understand is given in Figure 1. This is an α-carbon plot of the protein haptoglobin[1], a linear chain, as are most biological macromolecules, which contains 245 residues or ∿2500 atoms, and which as we can see from the figure, is folded in a complex and intricate way into a globular structure. The types of questions facing us now are: What are the forces that determine the structure? How do these complex biological macromolecules fold to attain their unique structures? What is the molecular basis of biological recognition and specificity, for example how do enzymes recognize and bind their substrates, how do antibodies recognize antigens? What role does the aqueous environment play? To answer these questions, and others like them, has been the goal towards which much of the work of the past fifteen years has been directed. We are still asking these questions, but hopefully are now much closer to achieving some of the answers. In this paper we will discuss some of the modern approaches being taken in studying these systems. We shall focus on peptides and proteins for the purpose of this discussion, but clearly the techniques do not depend on molecular type and could equally well be applied to other families such as nucleotides and carbohydrates.

0097-6156/81/0173-0161$08.00/0
© 1981 American Chemical Society

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch012

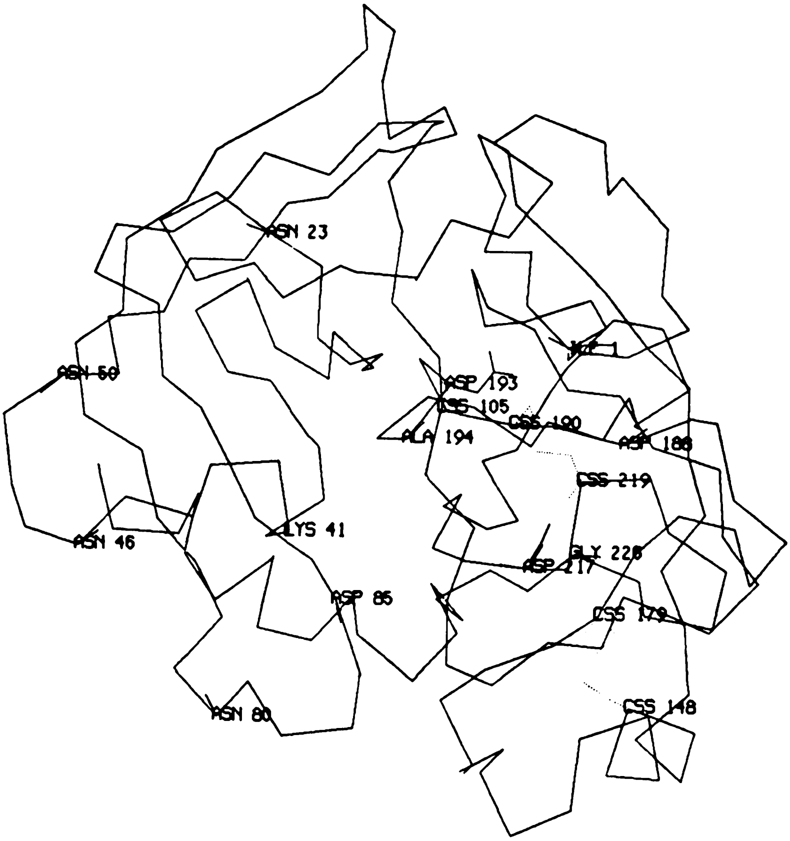
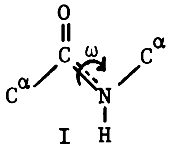


Figure 1. α-Carbon plot of the chain of the protein haptoglobin. Only the α-carbons are shown to allow the folding of the chain to be seen.

This structure was derived not by x-ray but rather in a study by Greer (1) using the structural homologies in the serine proteases, which are in turn homologous to haptoglobin. The outer loops are not defined well by this procedure (the structural homology breaks down in these regions) and thus it is especially relevant to the techniques discussed in this paper, as they are natural candidates for energy refinement.

In the initial stages of conformational analysis a large fraction of the work was devoted to small model compounds, in order to simplify the problem and develop the techniques[2a,b]. The dipeptide unit was the subject of much of this early work. It is the structural unit of which proteins are built and is shown in Figure 2. In a sense it is the "ethane" of the theoretical biophysicist. In the early work it was reasoned that the energy to stretch bonds and to distort angles is very large, relative to conformational variations, so these internal coordinates could be maintained at their equilibrium values. Furthermore the peptide bond



I

has partial double bond character, the barrier to rotation about this bond being about 20 Kcal/mole, thus it was assumed that this torsion angle, ω, could be frozen at 180°, and the six atoms in the peptide group kept rigid in the trans planar arrangement shown in schematic I. This leaves only two torsion angles ϕ, and ψ for each residue, which then completely determine the conformation of the backbone. This gives a tremendous reduction in the degrees of freedom, from 21 to 2 per residue, and made early studies tractable. The first approaches [2] to the problem of conformational energetics involved mapping the energy of the residue as a function of the angles ϕ and ψ. A function of the form

$$V(\Phi,\Psi) = \Sigma A/r_{ij}^{12} - \Sigma C/r_{ij}^{6} + \Sigma q_iq_i/r_{ij} + \Sigma K_\Phi(1 \pm \cos 3\Phi) + \Sigma K_\Psi(1 \pm \cos 3\Psi) + E_{hb} \tag{1}$$

was used. The first two terms represent the nonbonded interaction between all pairs of atoms, i and j, separated by 3 bonds or more-the distance between them being r_{ij}. This is a function of ϕ and ψ since the distance r_{ij} will vary with these angles if i and j are on either side of the angle. The next term represents the coulomb interaction between the partial charges carried by these atoms in the polar peptide group. The following two terms represent an intrinsic energy required to rotate about these angles in addition to that included in the previous terms. Finally the last term is an explicit functional form for hydrogen bonding, of which several types were used. The result of such a mapping is given in Figure 3. From this energy map we can see that much of the torsional space is excluded, basically due to steric interactions. Recent theoretical results have indicated some modifications in these terms. For example, ab-initio calculations and analysis of experimental data indicate that the intrinsic torsional potential

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch012

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch012

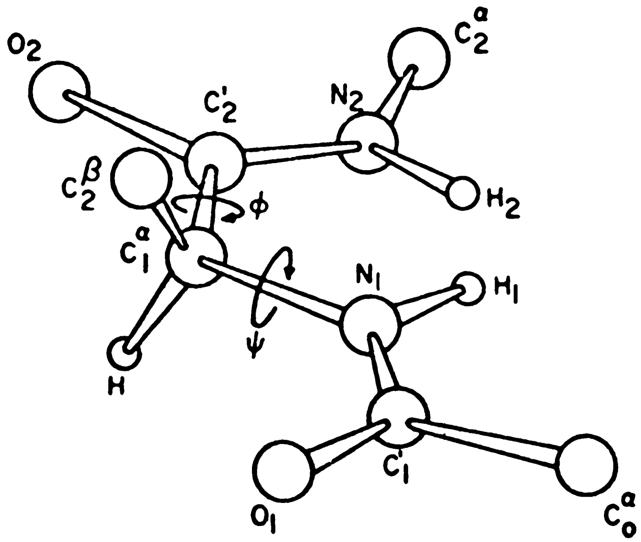


Figure 2. The structural unit of peptide. The torsion angles about the C′–N bond, φ; the Cα–C′ bond, ψ; and the peptide bond C′–N, ω; are indicated.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch012

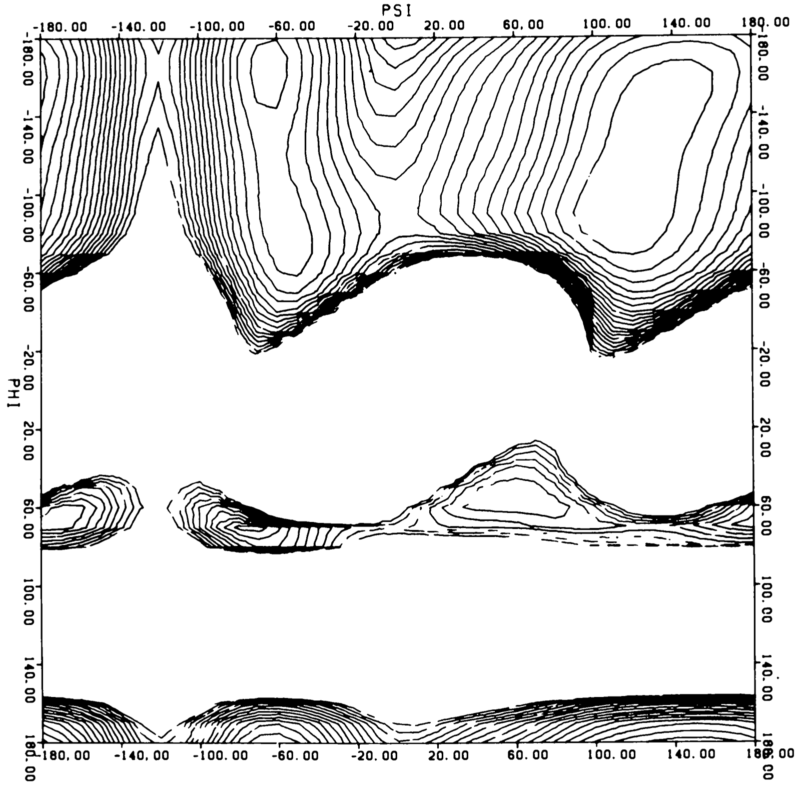


Figure 3. *The energy of* N-*acetyl*-N′-*methylalanine, the residue shown in Figure 2, as a function of the torsion angles* ϕ *and* ψ*. Equienergy lines are indicated by contours.*

about ϕ and ψ is negligible and may be omitted[3]. Furthermore, recent studies of crystal packing and sublimation energies indicate that if proper coulomb interactions are used they account for the major contribution to the hydrogen bonding energy and no explicit term is necessary[4-6]. Nevertheless, the basic features of the maps remain the same. The individual residue conformations commonly occuring in proteins such as extended chain, ε, and α-helix, α, are in allowed regions of the map, and residues in very high energy regions are rarely observed in proteins.

Following the initial mapping studies, minimization procedures were introduced. The conformational behavior of a great many oligopeptides, among them peptide hormones, was studied by minimizing the energies of these molecules with respect to the angles ϕ and ψ of each residue (and where appropriate the torsion angles χ_i of the side chains) [7]. In this way, a start was made in delineating the relation between primary structure (the covalent molecular connectivity) and the tertiary structure (the three dimensional conformation) of these important molecules.

However, as we know, molecules are in fact flexible and dynamic, rather than rigid and static. Furthermore, biomolecules exist and function not in vacuum but rather most often in an aqueous mileau. Thus it has become clear that if we are to make further progress in treating these exquisitely designed molecules we must investigate the importance of their flexibility, dynamics and environment. This is not a trivial task, as we alluded to above, and it has become possible only through the advent of current generation computers. Let us first consider the treatment of molecular flexibility and dynamics. We can represent the potential energy of a molecule in terms of all its internal degrees of freedom as shown in equation (2).

$$\begin{aligned} E = {} & \Sigma\left\{D_b[1 - e^{-\alpha(b-b_o)}]^2 - D_b\right\} + 1/2\ \Sigma H_\theta(\theta - \theta_o)^2 \\ & + 1/2\ \Sigma H_\phi(1 + s\cos n\phi) + 1/2\ \Sigma H_\chi \chi^2 \\ & + \Sigma\Sigma F_{bb'}(b - b_o)(b' - b_o') \\ & + \Sigma\Sigma F_{\theta\theta'}(\theta - \theta_o)(\theta' - \theta_o') + \Sigma\Sigma F_{b\theta}(b - b_o)(\theta - \theta_o) \qquad (2) \\ & + \Sigma\Sigma F_{\phi\theta\theta'}\cos\phi(\theta - \theta_o)(\theta' - \theta_o') + \Sigma\Sigma F_{\chi\chi'}\chi\chi' \\ & + \Sigma\varepsilon[2(r^*/r)^9 - 3(r^*/r)^6] + \Sigma q^2/r \end{aligned}$$

This type of representation of the potential energy in terms of the internal (valence) degrees of freedom is called a Valence Force Field. Valence force fields have long been used in vibrational spectroscopy in order to carry out normal mode analysis[8]. Basically what the terms in equation (2) express are the energies required to deform each internal coordinate from some unperturbed

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch012

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch012

standard value (denoted by the subscript "0"). In fact, however, equation (2) deviates somewhat from standard vibrational valence force fields by the inclusion of the last three terms, the non-bonded interaction, as we shall discuss further below. Looking at the individual terms we see that the first is a Morse potential yielding the energy required to stretch each bond from its relaxed value, b_0. The second term is quadratic and represents the energy stored in each angle when it is bent from its "standard" value, Θ_0. The third term represents the intrinsic energy required to twist the molecule about a bond by a torsion angle, ϕ. For example, as noted above, it takes approximately 20 Kcal/mole to twist the peptide bond in I by 90 degrees. The fourth term represents the energy required to distort intrinsically planar systems by χ from their planar conformation. (For example energy is required to move a carbon atom out of the plane of the other 5 atoms in the ethylene molecule, or similarly for the peptide group). The next terms represent the fact that various internal coordinates are coupled. That is the energy necessary to deform one internal depends on the current value of another, usually neighboring, internal. These coupling terms are known to be necessary from studies of vibrational spectra[8,9]. This concludes the standard spectroscopic valence force field.

We can, given the analytical representation in equation (2) (or one similar to it), minimize this energy with respect to all internal degrees of freedom, i.e. solve the equations $\partial E/\partial x_i=0$; $i=1, 3n$, where $\partial E/\partial x_i$ is the derivative of the energy with respect to cartesian coordinate x_i. Furthermore, at the minimum we can take the second derivatives of the energy, and by weighting these by the appropriate masses, construct the mass weighted second derivatives matrix, F. The dynamics in the form of the vibrational frequencies may be obtained from the eigenvalues, λ_i, of this matrix by

$$\begin{aligned} &\underset{\sim}{F}\, d\vec{q} = \lambda\, d\vec{q} \\ \text{where}\quad &d\vec{q} = \underset{\sim}{M}^{1/2}\, d\vec{x} \qquad (3) \\ &\underset{\sim}{F} = \underset{\sim}{M}^{-1/2}(\underset{\sim}{\partial}^{2}E/\partial x_i \partial x_j)\, \underset{\sim}{M}^{1/2} \end{aligned}$$

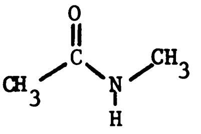
while the corresponding eigenvectors give the normal modes or characteristic dynamical motions associated with each frequency. Finally the vibrational spectra gives us information about the entropy and related thermodynamics of the conformation. Roughly speaking the "looser" the conformation, the higher its entropy, and lower its characteristic low frequency vibrational modes. This is expressed in the Einstein equations[11] relating the energy, (E_{vib}), the free energy, (A_{vib}), and entropy, (S_{vib}), to the vibrational frequencies.

$$E_{vib} = \sum_i \left[h\nu_i/2 + h\nu_i(e^{h\nu_i/kT} - 1)\right]$$

$$A_{vib} = \sum_i \left[h\nu_i/2 + kT \ln(1 - e^{-h\nu_i/kT})\right] \qquad (4)$$

$$S_{vib} = (E_{vib} - A_{vib})/T$$

In general, classical vibrational analysis has been concerned with small molecules. E.g. - a typical amide might be N-methylacetamide[12]:



The energy or even the relative energy of the molecule as a function of its conformation has not been of concern[8,12]. Thus, typically, nonbonded interactions have not been included. From our point of view this is a pity, since the nonbonded interactions are of crucial importance to the study of conformation and energetics. It means, unfortunately, that we cannot simply take valence force fields derived from vibrational spectroscopy and apply them directly to our problems. An advantage which would accrue to the field of vibrational spectroscopy from inclusion of the nonbonded interaction, is that frequency shifts which occur because of hydrogen bonding or strained interactions and their conformational dependence would be obtained in an entirely consistent way. The steric interactions can lead to significant frequency shifts and when included in a consistent way as in equation (2), can provide a powerful method for understanding intramolecular forces [13].

We have discussed the representation of the potential energy at some length because of its importance. The representation of the energy is fundamental and common to all conformational energy simulations, whether energy minimization, molecular dynamics, Monte-Carlo, or vibrational analysis. The properties calculated in any of these simulations are true to the biomolecular system being simulated to the extent that the energy expression faithfully represents the energy surface. It is of the utmost importance that we use increasingly sophisticated techniques to simulate these complicated systems, as well as continuing to improve and expand our library of energy functions[6].

Let us now turn to the results obtained in the simulation of various typical secondary structures found in peptides and proteins, using equation (2). We shall focus on the importance of allowing flexibility of the molecular geometry and the necessity for accounting for dynamic effects, especially low frequency deformation modes, in order to describe conformational equilibrium in

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch012

these molecules. For these purposes the constants in the representation of the potential energy were obtained by fitting the structure and vibrational spectra of N-methylacetamide (CH_3CO NH CH_3)[14] and the structure and sublimation energy of a large set of amide and acid crystals[4,5].

In Table I we report the thermodynamics describing the equilibrium between a series of conformations of N-acetyl-(alanyl)$_3$-N-methylamide. The potential energy of this molecule, as given by equation (2), was minimized with respect to the cartesian coordinates of all atoms in the molecule. Four initial "helical" conformations were minimized, C_{7eq}, C_{7ax}, α_R and α_L. (These conformations are shown in Figure 4 for the nonapeptide). As can be seen from the Table, in all cases but the last, local minima were reached close to the initial conformation. In the case of the α_L trimer a "folded" structure was obtained characterized by a CO (1) -- NH(3) hydrogen bond, forming a 10 membered ring, and a CO(3) -- NH(4) hydrogen bond forming a C_7 ring. That all structures correspond to true local minima can be seen by examining the value of the maximum derivatives. For all structures, all derivatives of the energy with respect to the cartesian coordinates are less than 10^{-7} Kcal/A°. The value of the derivatives is the only reliable measure of convergence in energy minimization (along with the requirement that the second derivative matrix be positive definite). For the trimer the lowest potential energy of these structures is achieved in the C_{7eq} conformation which is favored by 3.4 Kcal/mole over the α_R helix. The dynamics also contribute however. Although the vibrational energy is essentially constant here, the entropy of different conformations is significantly different and in this case also favors the C_{7eq} (by 1.6 Kcal/mole over the right handed α-helical conformation).

The corresponding quantities for the nonapeptide of alanine are represented in Table II. As the peptide chain grows longer the α_R helix becomes more stable relative to other conformations and this is reflected in the thermodynmaics of the chains. The potential energy of the helix is now over 21 Kcal/mole or 2 Kcal/residue more stable than the C_{7eq} conformation. This occurs because each additional peptide dipole now aligns in a favorable direction with respect to the existing chain dipole as well as hydrogen bonding to the residue preceeding it by 3 along the chain. The conformational entropy, however, opposes this trend and significantly favors the C_7 conformation, by 10 Kcal/mole (or$\sim$1 Kcal/residue) at 298°K. Thus we see that the conformational entropy contributes significantly to the free energy of these conformational equilibria. The effect can be even more dramatic when chains incorporating more than a single amino acid are considered. In a recent study[15] of a series of host-guest oligopeptides it was found that the contribution of the entropy to the free energy difference between C_{7eq} and α_R conformations could be as large or larger, at room temperature, than the potential energy, resulting in a reversal of the predicted stability (Table III). It is too

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch012

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch012

Table I

Entropy Contribution to Conformational Equilibria of Peptides

N-acetyl-(-Alanyl-)$_3$-N-methylamide

Conformation		Potential Energy (E_{conf})	Vibrational Energy (E_{vib})	T*Vibrational Entropy (TS)	Free Energy (A_{tot})	ΔA_{tot}
C_{7eq}	(-80,80)	0.34	113.77	24.21	89.90	-4.6
C_{7ax}	(80,-80)	2.54	113.80	22.62	93.72	-0.8
α_R	(-55,-60)	3.79	113.61	22.85	94.54	0.0
α_L	(*)	2.61	113.69	21.86	94.44	-0.1

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch012

Table II

Entropy Contribution to Conformational Equilibria of Peptides

N-acetyl-(-Alanyl-)$_9$-N-methylamide

Conformation		Potential Energy (E_{conf})	Vibrational Energy (E_{vib})	T*Vibrational Entropy (TS)	Free Energy (A_{tot})	ΔA_{tot}
C_{7eq}	(-80,80)	17.48	303.63	73.13	247.97	11.1
C_{7ax}	(80,-80)	22.57	303.73	68.64	257.66	20.8
α_R	(-55,-60)	-3.89	303.58	62.94	236.86	0.0
α_L	(55,60)	24.43	303.20	63.13	264. 50	25.6

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch012

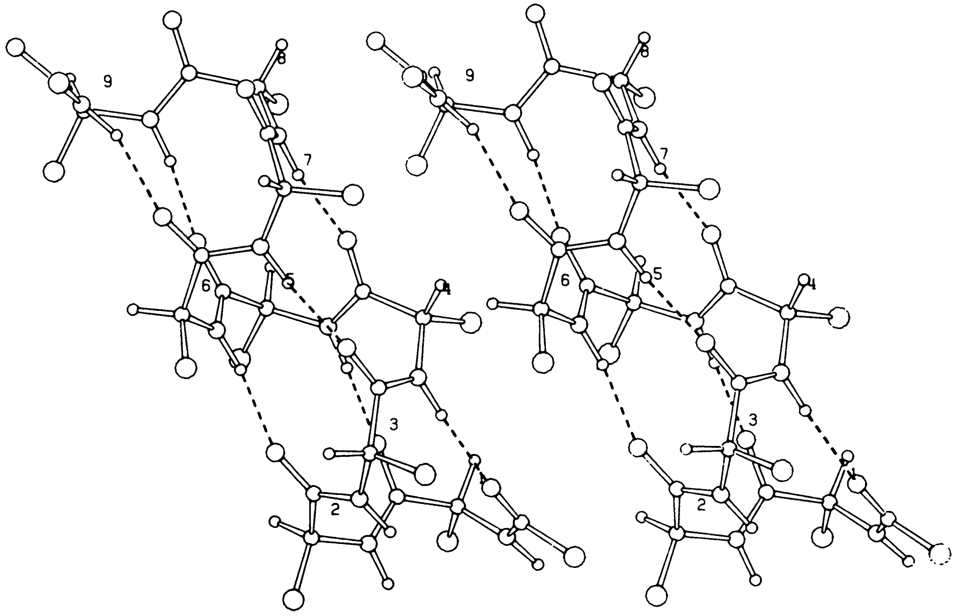


Figure 4a. Stereo figure of the peptide secondary structure of the right-handed α-helix (α_R). This and Figures 4b, 4c, and 4d are the final structures that result from the minimization of the energy as described ($dE/dx = 0$*). The first three N-H bonds do not form hydrogen bonds. The last hydrogen bond is slightly distorted.*

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch012

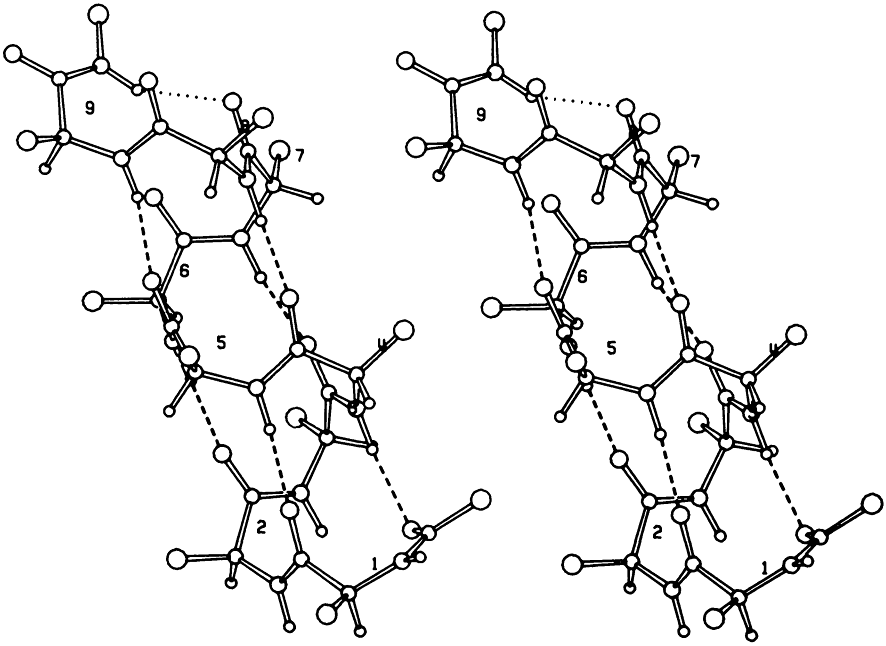


Figure 4b. Stereo figure of the peptide secondary structure of the left-handed α-helix (α_L). Minimization results in distortion at the ends. At the C-terminal end a ten-membered hydrogen bonded ring has been formed, and at the N-terminal end the peptide group has twisted out of the plane. (Compare with the structure of the right-handed helix).

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch012

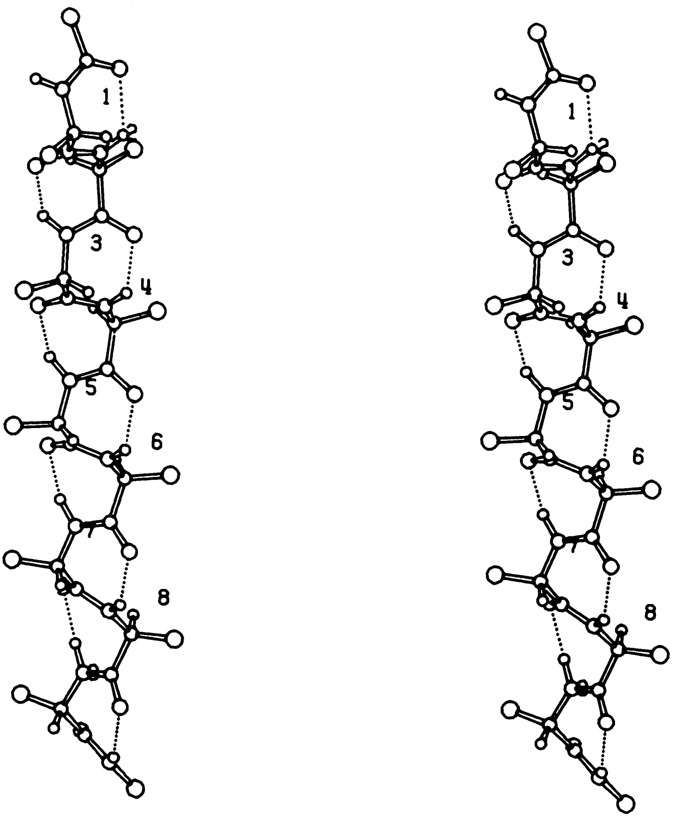


Figure 4c. Stereo figure of the peptide secondary structure of the extended C-7 equatorial helix (C_{7eq}). This structure is formed by repeating seven-membered rings.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch012

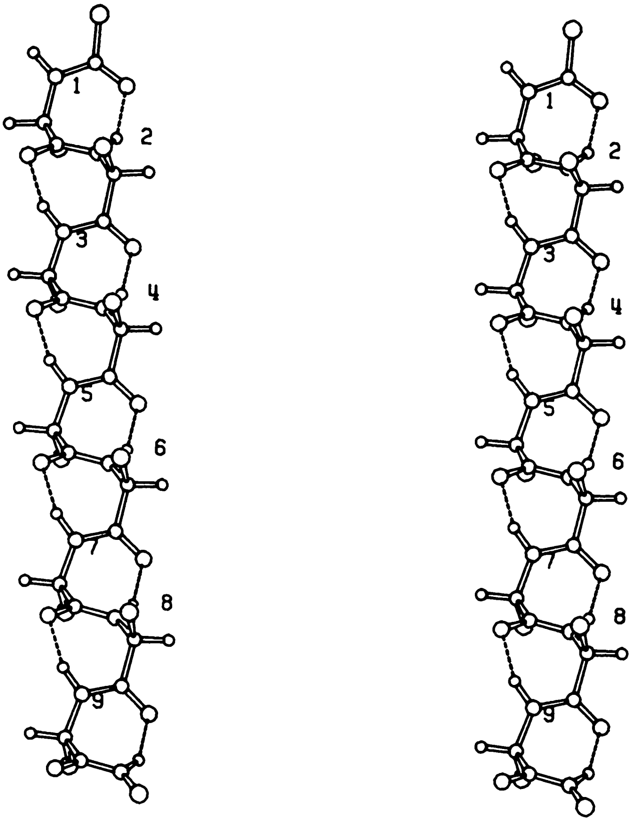


Figure 4d. Stereo figure of the peptide secondary structure of the extended C-7 axial chain (C_{7ax}). This is the same structure as the C-7 equatorial except that the C_β carbon (shaded atom) is axial to the seven-membered ring rather than in the equatorial position.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch012

Table III

Entropy Contribution to Conformational Equilibria of Peptides

Boc-Met_3-Gly-Met_2-O-Me

	Conformation	Potential Energy (E_{conf})	Vibrational Energy (E_{vib})	T*Vibrational Entropy (TS)	Free Energy (A_{tot})	ΔA_{tot}
C_{7eq}	(-80,80)	-41.22	258.72	87.25	130.25	1.83
α_R	(-55,-60)	-43.57	258.77	83.12	132.08	0.0

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch012

early to generalize these results but we clearly must consider the entropic contribution when considering the relative stability of various conformations, especially when considering a general sequence and when glycine is present in the sequence.

It is of interest to consider the molecular behavior underlying these entropic differences in an analogous manner to the way in which we understand the intramolecular forces underlying energetic variations. The conformational dependence of the entropy, reflects the conformational dependence of the vibrational spectra. We can represent this conformational dependence in the form of a "difference spectra" in which we plot the difference in the frequency of each normal mode as a function of the normal mode. (The normal modes are arranged in descending order of the frequencies). Such a "difference spectrum", with the α_R taken as the reference spectrum, is shown in Figure 5.

One of the features readily observed in Figure 5 is that the last 27 low frequencies are all of lower frequency in the extended, $C7_{eq}$ conformation than in the right-handed α-helix. Consideration of equation (4) shows that only the low frequencies contribute significantly to the entropy. That is when $h\nu >> kT$ the entropic contribution is essentially zero, while if $h\nu << kT$ the contribution to the entropy becomes proportional to $1/\nu$. In order to get some feeling for the numbers involved we note that the highest and lowest frequencies are ∿3450cm-1 and ∿10cm-1, which correspond to ∿10 and ∿0.03 Kcal respectively at room temperature. (kT is of course 0.6 Kcal). Thus it is clear that the lower frequencies which characterize the $C7_{eq}$ conformation will result in a higher entropy. The ten lowest frequency modes for the $C7_{eq}$ extended structure, and the α_R-helix are compared in Table IV, along with their corresponding energies and entropies.

We also observe from Figure 5 that much of the vibrational spectra is extremely sensitive to conformation. In many cases frequency differences of over 50 cm^{-1} occur. Such differences may be used as fingerprints of the different conformations and are a potentially powerful source of information for determining conformation in solution, especially non-aqueous solutions. Several studies have already been carried out using various regions of the IR spectra to determine conformation, using classical spectroscopic techniques [16]. The advantage of using the full representation of the energy as given in equation (2) is that, as noted above, frequency shifts due to differences in internal forces in the differences in internal forces in the different conformations are obtained.

A straightforward example of the type of information we can obtain from the vibrational spectra may be obtained from the behavior of the N-H stretching frequencies. In Table V the 10 N-H stretching frequencies in the α_R helical conformation and the $C7_{eq}$ conformation are given as well as the residue in which they occur. We note first of all that the vibrational frequency of the N-H is extremely sensitive to the location of the residue in the chain.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch012

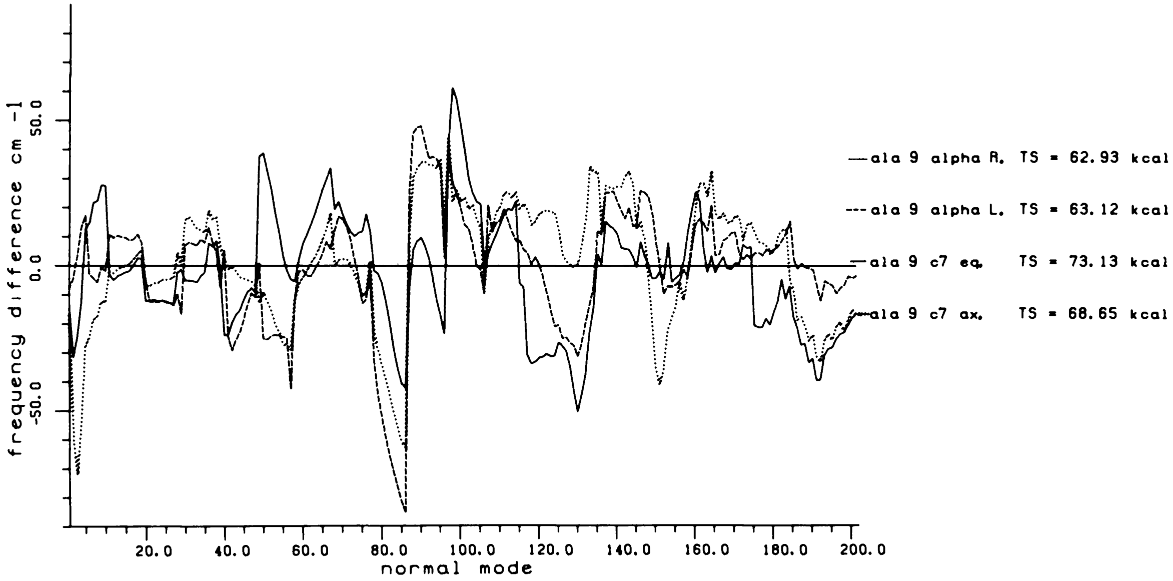


Figure 5. Difference spectra between the right-handed α-helix, the C-7 equatorial, C-7 axial, and the left-handed α-helical nonapeptide structures shown in Figure 4. The difference between the vibrational frequency in the α_R conformation and each of the other structures is plotted against the corresponding numerical value of the normal mode, demonstrating the conformational dependence of the vibrational spectra.

In Supercomputers in Chemistry; Lykos, P., el al.;
ACS Symposium Series; American Chemical Society: Washington, DC, 1981.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch012

Table IV

Low Frequency Modes of N-acetyl-(Alanyl)$_9$-N-methylamide

C_{7eq}			α_R		
Frequency cm^{-1}	Energy Kcal/mole	TS[a] Kcal/mole	Frequency cm^{-1}	Energy Kcal/mole	TS Kcal/mole
21.0	0.593	1.95	60.5	0.596	1.32
19.5	0.593	1.99	50.5	0.595	1.43
18.3	0.593	2.03	47.0	0.594	1.47
16.4	0.592	2.09	44.6	0.594	1.50
13.5	0.592	2.21	38.9	0.594	1.58
12.1	0.592	2.27	35.7	0.594	1.63
9.6	0.592	2.41	32.5	0.593	1.69
7.2	0.592	2.59	27.6	0.593	1.78
5.0	0.592	2.80	23.0	0.592	1.90
3.6	0.592	2.99	20.1	0.592	1.97

[a] Entropic contribution to the free energy at 298° K

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch012

Table V

Amide (NH) Stretching Frequencies

α_R Frequency cm^{-1}	α_R Residue	C_{7eq} Frequency cm^{-1}	C_{7eq} Residue
3516	3	3501	1
3510	2	3478	10
3490	1	3466	2
3469	10	3457	9
3443	4	3456	3
3436	9	3451	8,7
3429	8	3451	4,5
3428	6	3450	7,6,4
3422	5,7	3450	5,7,6
3422	7,5	3449	6,5,7

In Supercomputers in Chemistry; Lykos, P., el al.;
ACS Symposium Series; American Chemical Society: Washington, DC, 1981.

The three highest stretching modes in the α_R-helix are calculated to occur in the first three residues. These N-H groups are not involved in hydrogen bonds in the α_R-helix, since it is not until the fourth residue that the chain winds around so that N-H (4) hydrogen bonds to the carbonyl of the first residue (C=O (1)). The next highest frequencies correspond to the N-H(10) and N-H(4) stretches. These first and last hydrogen bonds are somewhat distorted as compared to the hydrogen bonds in the interior of the helix. Since it is well known that the N-H stretching frequency is down-shifted by hydrogen bonding, and that in general the frequency shift is proportional to the strength of the hydrogen bond, the pattern which emerges naturally from application of equation (2) in its entirety is gratifying. In comparing the N-H stretching region in the α_R-helix with the $C7_{eq}$ conformation we note that a much larger spread, 96 wavenumbers, obtains in the α_R-helix than in the $C7_{eq}$($52\ cm^{-1}$). This alone should aid in differentiating between these two conformations in solution. A still more powerful determination is indicated by Table V, however. While in the α_R-helix the N-H in the third residue is calculated to occur at $3500\ cm^{-1}$, in the $C7_{eq}$ it occurs in the centre of the N-H region at $3450\ cm^{-1}$. Thus the frequency shift obtained on insertion of N^{15} into the N-H bond of this residue would be a further sensitive indicator of the conformation existing in solution. Consideration of Figure 5, along with consideration of the relevant normal modes, indicates many regions of the 201 modes in the spectra where such differences occur. It is worth mentioning that there is nothing in the above analysis which restricts it to helices; it can just as readily be applied to any minimum energy conformation.

Let us now consider briefly peptide-water interactions. As we noted above, most biomolecules exist in an aqueous environment. This aqueous environment is known to play a major role in determining the conformational and thermodynamic properties of biomolecular systems. In the past several years there has been an increasing effort to explore the nature of the aqueous solvent and characterize its interaction with solute molecules. The problem in treating solvent effects lies in the disorder inherent in the liquid state. In order to simulate the properties of this phase we must include a representative sample of configuration space, or in other words simulate the many configurations accessible to the solvent molecules. This involves generating enough states of the system to calculate desired macroscopic properties by a Blotzmann average over these states. This may be stated mathematically by:

$$\langle X\rangle = \Sigma\ X_i\ \exp(-E_i/RT)/Z$$
$$Z = \Sigma\ \exp(-E_i/RT) \qquad (5)$$

In this equation <X> represents the property of interest, X_i and E_i are the values of the property, and the energy of the system in

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch012

the state i, and the sum runs, in principal, over all accesible states of the system.

The problems being addressed in recent work carried out in various laboratories include the fundamental nature of the solute-water intermolecular forces, the aqueous hydration of biological molecules, the effect of solvent on biomolecular conformational equilibria, the effect of biomolecule - water interactions on the dynamics of the waters of hydration, and the effect of desolvation on biomolecular association[17]. The advent of present generation computers have allowed the study of the structure and statistical thermodynamics of the solute in these systems at new levels of rigor. Two methods of computer simulation have been used to achieve this fundamental level of inquiry, the Monte Carlo and the molecular dynamics methods.

In the latter method we simply specify the initial conditions of the system, the coordinates of the atoms of the biomolecule in its initial conformation, as well as those of the water molecules constituting its environment, along with a set of initial velocities for these atoms. Having specified the initial conditions, Newtons equation of motion

$$- \partial V\left\{\vec{r}_1 \ldots \vec{r}_n\right\}/\partial r_i = \vec{F}_i \left\{\vec{r}_1 \ldots \vec{r}_n\right\} = m_i d^2\vec{r}_i/dt^2 ; \; i = 1,\ldots,n \qquad (6)$$

are integrated numerically in order to simulate the classical dynamics. The energy expression used is the same needed for energy minimization given in equation (2), where the last three terms comprising the nonbonded interatomic energy now apply to the intermolecular water - peptide interactions as well. Given the potential energy expression, the total force on each atom arising from its interaction with neighboring atoms is calculated in the same way as done for the minimization described above. For dynamics however, we use this force along with the mass of the atom to compute the acceleration from Newtons law (equation (6)). We then take a small time step, t, and applying the acceleration we update the velocity and position of the atom, to a new velocity and position.

$$\begin{aligned} \vec{v}_i(\text{NEW}) &= \vec{v}_i(\text{OLD}) + d^2\vec{r}/dt^2 \cdot \Delta t \\ \vec{r}_i(\text{NEW}) &= \vec{r}_i(\text{OLD}) + d\vec{r}/dt \cdot \Delta t \end{aligned} \qquad (7)$$

Having updated the position, we again calculate the force on each atom for the new configuration of the system and complete the next iteration by obtaining the new acceleration and integrating. In this way we compute the atomic trajectories as a function of time.

The need for powerful supercomputers may be appreciated by examining the size of a typical system to be treated. A single small protein contains on the average of $\sim$2000 atoms. If we in-

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch012

clude an environment of only 500 water molecules we are faced with a "3500 body" problem.

The Monte Carlo method, in contrast to the deterministic molecular dynamics, is a stochastic method in which the configuration space of the system is sampled by some random sampling method. Up to now, when used to treat hydration phenomena, the solute molecule has been maintained in a fixed position, and the solvent molecules allowed to relax around it. In the common Metropolis algorithum a water molecule is chosen at random, from some initial configuration of the system, a random displacement and rotation is applied to the chosen molecule and the change is energy, E, of the system examined. If $\Delta E < 0$, i.e. the energy of the system has been decreased by this random motion the displacement is accepted as a new configuration of the system, a new water molecule picked at random, and the process repeated. If the energy goes up, $\Delta E > 0$ still another random number, R on the interval 0 to 1 is chosen and the Boltzmann factor of the energy change $\exp(-\Delta E/kT)$ compared with it. If $\exp(-\Delta E/kT) >> R$, the new configuration is accepted, otherwise it is rejected and the old configuration counted again in the configurational average. (It can be seen that if the energy has increased by a large amount, $\Delta E >> 0$, the exponential will be small. Thus high energy states have a low probability of being included.) This is the Metropolis algorithm[18] and can be shown to generate configurations with probability $\exp(-E/kT)$. It then follows that the required configurational average in equation (5) are simply the arithmetic average over these Boltzmann weighted configurations.

The Monte Carlo method provides a space average, while molecular dynamics gives the time average of the given property. Each method has advantages for specific applications. Molecular dynamics is clearly preferable if we are interested in calculating a dynamic quantity such as the diffusion constant, while the Monte Carlo method is more flexible and may, for example, be more effective in sampling regions of configuration space which are separated by significant energy barriers.

The molecular dynamics method has been applied recently to do an extensive study of solvent interactions in a solution of an alanine dipeptide in water[17b,c]. The effect of solute proximity on dynamic behavior of the solvent, the range of influence of the solvent, the nature of the solvent in the neighborhood of various functional groups in the peptide, as well as the effects of solvent on the peptide dynamics were investigated in these works.

The Monte Carlo method has been applied more extensively to the study of solvation phenomena. Clementi has looked at solvent distribution around a large variety of amino acids and nucleotides. This work is summarized by Clementi in detail in this volume[17e]. It should be noted that all this work on solvent using these powerful simulation techniques has appeared within the last five years and most of it even more recently. This is a direct result of the advances in computer technology as well as the adaptation of the

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch012

methods and developments of the necessary potential energy functions to treat these systems[6].

Some of the most systematic and rigorous work in this area has been carried out in Beveridge's laboratory. Simulation of dilute aqueous solution of methane, a hydrophobic molecule, aqueous solutions of ions including Li^+, Na^+, K^+, F^-. and Cl^-, and polar compounds such as formaldehide (H_2 CO), formamide (NH_2 CHO), and glyoxal have been described recently[17a]. The emphasis in this work has been to develop methods of describing and analyzing hydration phenomena and to investigate the convergence characteristics of these simulations. The analysis has been based on the approach of quasi-component distribution functions, which has been developed by Ben-Naim[19]. The basic idea underlying this analysis is the assignment of water molecules in the first hydration layer to specific atoms or functional groups in the solute molecule. In this way one can investigate hydration of a given functional group and ascertain, among other things, the extent to which it depends on the particular molecule in which it is found.

We have been more concerned with the nature of the water around proteins and peptides. To this end we have investigated the structure and energetics of the solvent, both ordered and disordered around the enzyme lysozyme, in the triclinic crystal[17d]. In addition to lysozyme, we have characterized the water structure and fluctuations in the crystal of a cyclic hexapeptide, (L-Ala-L-Pro-D-Phe)$_2$[20], and studied the effect of solvent on the conformation of the dipeptide of alanine[21] and on the equilibria between extended and helical alanine polypeptides such as those discussed in the previous section[22]. The latter systems simulate aqueous solution conditions rather than crystalline environment.

The study of solvent structure in crystals has several advantages. In the case of crystals we are considering a real system insofar as the number of water molecules and the protein packing is that experimentally observed. The solvent in such systems may range from fully ordered to fully disordered. The approximation of periodic boundary conditions usually used to account for edge effects in solution studies apply by definition to a crystal system. The paramount advantage however is that these systems have been experimentally characterized by x-ray and thus the results of the simulation may be evaluated in terms of the degree to which they agree with the corresponding experimental quantities. Given agreement with the experimental results, the detailed picture at the molecular level, of the structure, energetics, and structural fluctuations in the system which give rise to the experimental observables but are unavailable from experimental techniques, may be obtained from the simulation.

It is worthwhile to present some examples of the types of results available from Monte Carlo simulations of peptide solvent systems. In Figure 6 we present the convergence of the energy of hydration of the lysozyme crystal over a million configurations. The initial energy is high (∿−0.5 Kcal/mole of water) due to the arbitrary placement of the water molecules in the initial configu-

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch012

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch012

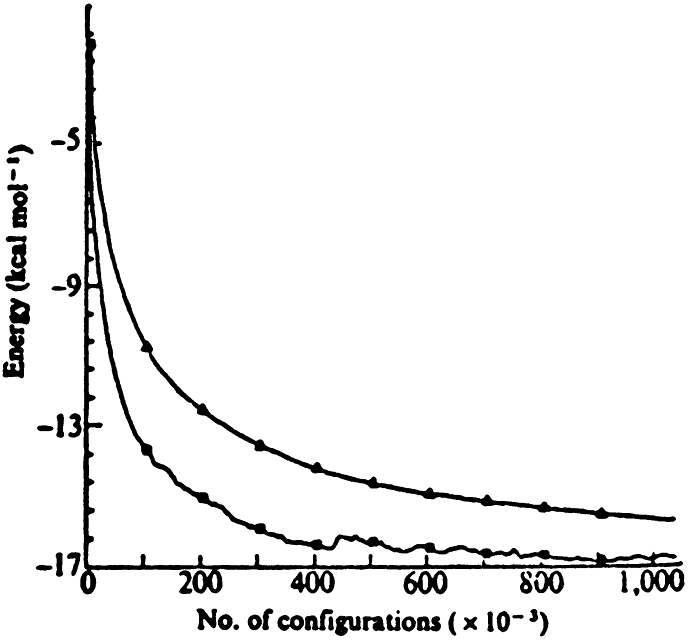


Figure 6. The average energy of the water of hydration of the triclinic lysozyme crystal as a function of the number of configurations generated in the Monte Carlo simulation. The upper curve (▲) corresponds to the cumulative statistical average, and the lower curve (■) gives the statistical average over sequential sets of 5,000 configurations.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch012

ration. The structure anneals slowly until an average energy of ∿−16.8 Kcal/mole is reached at which point it begins to fluctuate about this value. A stereo figure of the packing of four of the molecules in the cyclic hexapeptide crystal, including the relative positions of the ordered and partially ordered water molecules is given in Figure 7. A second type of information we can obtain from computer simulation is shown in Figure 8. Here we have calculated the probability of finding a water molecule in the section of the cyclic peptide crystal shown. This probability is contoured and two positions in the crystal, one angstrom apart, which are found by x-ray to contain a water molecule with approximately half occupancy are indicated by circles (positions 3 & 4 in Figure 7). The calculated density spans these two positions nicely, indicating the water molecule oscillates between these positions during the simulation. By consideration of individual configurations it can be seen that this motion is part of a cooperative fluctuation in the hydrogen bonded network of water molecules bridging the peptide. Two typical configurations, which represent instantaneous molecular structures, that is what one would see if he could take a snapshot of the system, are presented in Figure 9. From a series of these snapshots along with the energy of the system, and the energetics of each water, we gain insight into the nature of the cooperative fluctuations taking place in this hydrogen bonded system[20]. This type of information is unattainable by experiment. As indicated above, it is the average over millions of these configurations which gives rise to the thermodynamic and average structural features measured by the various experimental techniques. The extent to which these observables agree gives us confidence in the validity of interpretation based on the microscopic configurations and their energetics, which underly the average values.

Summary. In conclusion, the advent of current generation computers has allowed the development of a new level of rigor in statistical thermodynamic and dynamic studies of organic and biomolecular systems. We have discussed how we can now include the relaxation of molecular geometry, the treatment of conformational entropy, and the inclusion of solvent effects in theoretical treatments of biolmolecular systems, all of extreme importance in simulating the behavior of those systems. In addition we have indicated how the vibrational spectra can be calculated and its conformational dependence be used as a probe of conformation. It was pointed out that these developments have for the most part occurred within the last five years and in fact most publications in the area have been in the last year or two. Their full impact on this exciting field is yet to be felt.

Acknowledgements. This work was supported by grants from the National Insitute of General Medical Sciences (GM 24117 and GM 18694) and the U.S. Israel Binational Science Foundation. We are grateful to the Weizmann Institute computer center for provision of computer facilities.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch012

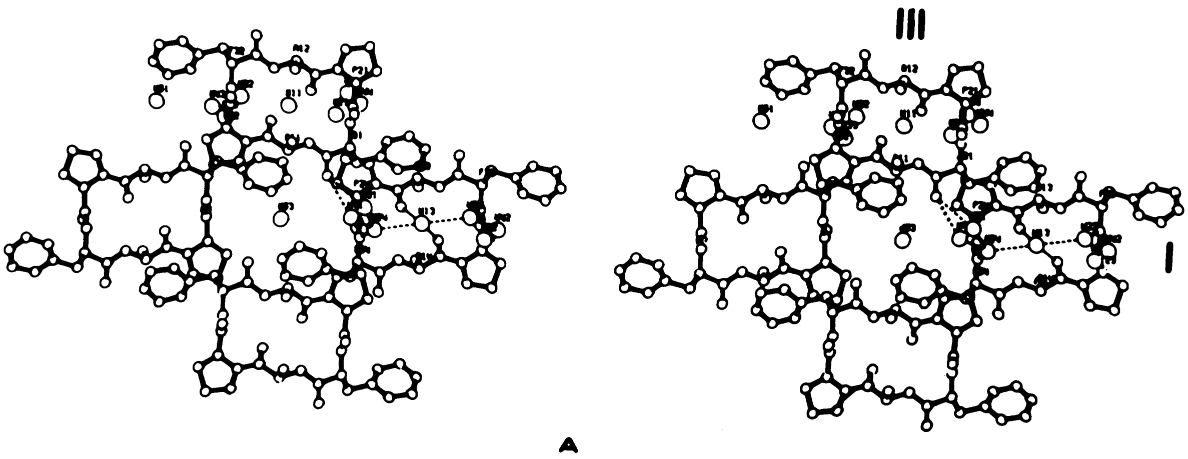


Figure 7. Stereo view down the Z-axis of the packing of cyclo(L-Ala-Pro-D-Phe)$_2$ in the crystal and the positions of the water molecules observed in the x-ray results.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch012

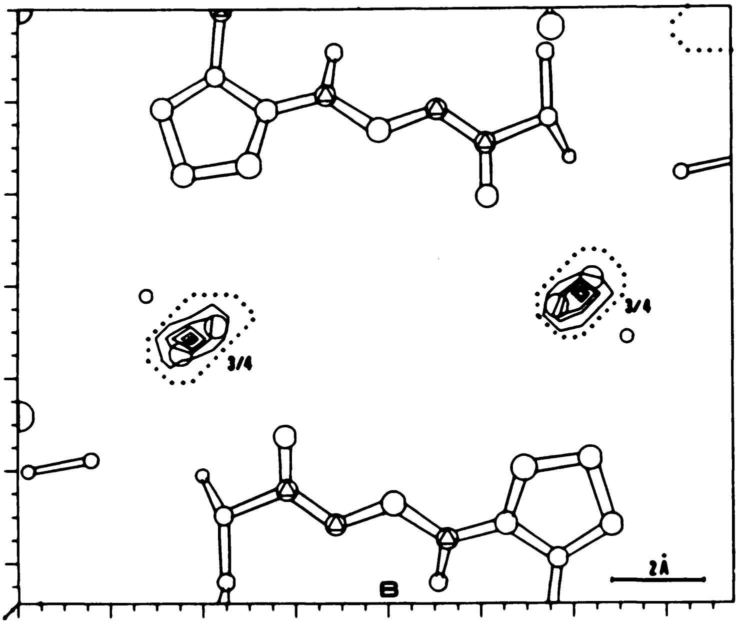


Figure 8. Probability map of the water positions obtained from the Monte Carlo simulation of cyclo(L-Ala-L-Pro-D-Phe)$_2$. In this section the density for the two water molecules with two alternate positions 3 and 4 are shown. The x-ray positions are indicated by the circles within the density. Note the density spanning these, indicating that the water molecules oscillate between these positions during the simulation.

In Supercomputers in Chemistry; Lykos, P., el al.;
ACS Symposium Series; American Chemical Society: Washington, DC, 1981.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch012

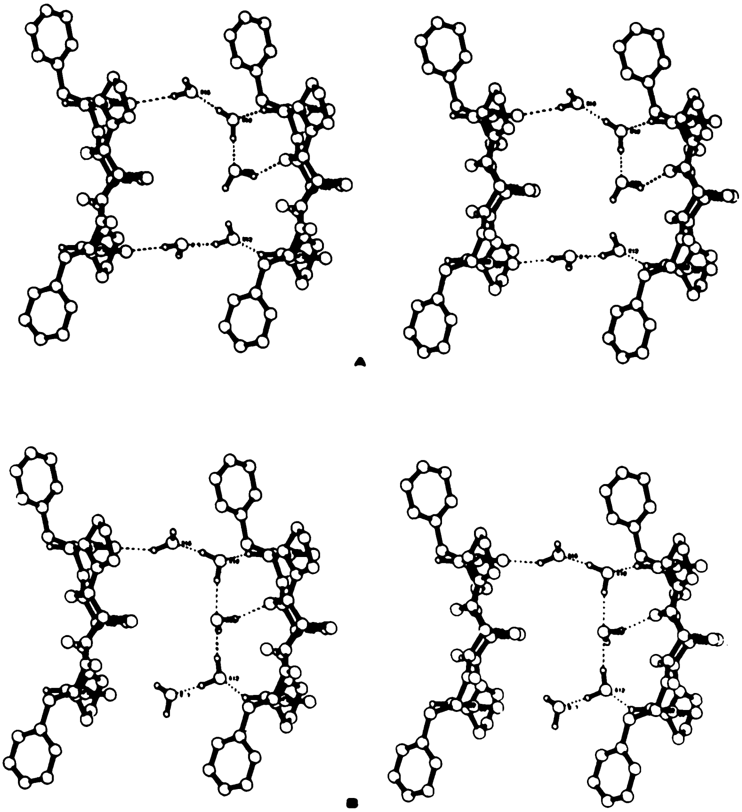


*Figure 9. Stereo "snapshots" of the arrangement of the water molecules in the crystal of cyclo(*L*-Ala-*L*-Pro-*D*-Phe)$_2$.*

Calculations were also carried out at the Biomedical Computer Resource at University of California, San Diego, supported by Grant No. RR 00757 of the Division of Research Resources, National Institutes of Health.

Literature Cited

1. Greer, J. *Proc. Natl. Acad. Sci. USA* 1980, *77*, 3393.

2. (a) Ramachandran, G.N.; Sasiekharan, V. *Advan. Protein Chem.* 1968, *23*, 283. (b) Liquori, A.M. *Quart. Rev. Biophys.* 1969, *2*, 65. (c) Ramachandran, G.N.; Ramakrishnan, Venkatachlam, C. M. *Biopolymers* 1965, *3*, 591. (d) Leach, S.J.; Nemethy, G; Scheraga, H.A. *Biopolymers* 1966, *4*, 369. (e) DeSantis, P.; Giglio, E; Liquori, A.M.; Ripamonti *J. Polymer Sci.* 1963, Part A, *1*, 1383.

3. Hagler, A.T.; Leiserowitz, L.; Tuval, M. *J. Am. Chem. Soc.* 1976, *98*, 4600.

4. Hagler, A.T.; Huler, E; Lifson, S. *J. Am. Chem. Soc.* 1974, *96*, 5319.

5. (a) Lifson, S.; Hagler, A.T.; Dauber, P. *J. Am. Chem. Soc.* 1979, *101*, 5111. (b) Hagler, A.T.; Lifson, S; Dauber, P. *J. Am. Chem. Soc.* 1979, *101*, 5122.

6. Dauber, P.; Hagler, A.T. *Accounts of Chem. Res.* 1980, *13*, 105.

7. (a) Scheraga, H.A. *Chem. Rev.* 1971, *71*, 195. (b) Nemethy, G.; Scheraga, H.A. *Quart. Rev. Biophys.* 1977, *10*, 239. (c) Ingwall, R.T.; Goodman, M. MTP Int. Rev. Sci. *Org. Chem. Ser. Two* 1976, *6*, 153.

8. (a) Ermer, O., in "Bonding Forces", Structure and Bonding *27*, Springer Verlag, 1976, p. 161. (b) Schachtschneider, J.H.; Snyder, R.G. *Spect. Chim. Acta* 1963, *19*, 117.

9. Califano, S. *Pure Appl. Chem.* 1969, *18*, 353.

10. Wilson, E.B.; Decius, J.C.; Cross, P.C. "Molecular Vibrations" McGraw Hill, New York 1955.

11. Hill, T.L. "An Introduction to Statistical Thermodynamics", Addison-Wesley Reading, Mass. 1960.

12. (a) Miyazawa, T.; Shimanouchi, T.; Mizushima, S. *J. Chem. Phys.* 1956, *24*, 408. (b) Rey-Lafon, M.; Forel, M.T.; Garrigou-LaGrange, C. *Spect. Acta* 1973, *29A*, 471. (c) Fillaux, F.; Deluze, C. *J. Chim. Phys.* 1976, *73*, 1010.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch012

In Supercomputers in Chemistry; Lykos, P., el al.;
ACS Symposium Series; American Chemical Society: Washington, DC, 1981.

13. Hagler, A.T.; Stern, P.S.; Lifson, S.; Ariel, S. J. Am. Chem. Soc. 1979, 101, 813.

14. Hagler, A.T., unpublished results.

15. Hagler, A.T.; Stern, P.S.; Sharon, R.; Becker, J.M.; Naider, F. J. Am. Chem. Soc. 1979, 101, 6842.

16. See e.g. Krimm, S.; Bandekar, J. Biopolymers 1980, 19, 1.

17. (a) See Proceedings of The Conference on Quantum Chemistry in Biomedical Science, New York Academy of Science, in press, and references therein. (b) Rossky, P.J.; Karplus, M; Rahman, A. Biopolymers 1979, 18, 825. (c) Rossky, P.J.; Karplus, M J. Am. Chem. Soc. 1979, 101, 1913. (d) Hagler, A.T.; Moult, J. Nature 1978, 272, 222. (e) Clementi, E., this volume.

18. Metropolis, N.; Rosenbluth, A.W.; Rosenbluth, N.M.; Teller, A.; Teller, E. J. Chem. Phys. 1953, 21, 1087.

19. Ben Naim, A. "Water and Aqueous Solutions", Plenum Press, New York 1974.

20. Hagler, A.T.; Moult, J.; Osguthorpe, D.J. Biopolymers 1978, 17, 395.

21. Hagler, A.T.; Osguthorpe, D.J.; Robson, B. Science 1980, 208, 599.

22. Destree, G; Englert, A.; Hagler, A.T., to be published.

RECEIVED April 10, 1981.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch012

A Micro Vector Processor for Molecular Mechanics Calculations

DAVID N. J. WHITE

The University, Chemistry Department, Glasgow G12 8QQ Scotland

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch013

Certain chemical computations such as molecular mechanics, molecular orbital and X-ray crystallographic calculations have always required large and powerful mainframe computers. The traditional solution to the problem of performing these calculations within individual laboratories, an end which is desirable for a number of reasons, has been the purchase of a minicomputer and the cramming of quart-sized programs into pint-sized machines. The floating point arithmetic capability of the average minicomputer is strictly limited and the need arose for peripheral devices, often called "array processors", to perform these operations for the minicomputer. This is a satisfactory, though still expensive, method of providing "in-house" number crunching facilities. However, the advent of the microprocessor and single chip floating point processors raises the possibility of providing these facilities at very modest cost indeed.

Peripheral processors which are capable of performing floating point arithmetic operations at high speed are used to enhance the poor performance of popular general purpose minicomputers in this area. These devices are described in various ways but the following nomenclature will be used in this paper. It is assumed that all floating point operations will be performed on arrays of data and so the nomenclature reflects the nature of the hardware. Most floating point accelerators currently available commercially consist essentially of a single floating point processor, which is pipelined to maximize its throughput, together with a control processor, memory and I/O channel to exchange data with the host minicomputer. These units will be called attached floating point processors or AFPP's and typical devices are the Floating Point Systems AP120B and the CSPI MAP-200. Accelerators which contain a linear array (i.e. a vector) of identical floating point processors are called vector processors or VP's and a typical device is the CSPI MAP-300. Finally there are devices which contain a number of floating point processors arranged in a grid fashion and these devices will be called array processors or AP's of which the only

0097-6156/81/0173-0193$11.00/0
© 1981 American Chemical Society

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch013

commercial example is the ICL DAP. The architecture and usage of the commercial "array processors" mentioned in this paper are discussed in "High Speed Computer and Algorithm Organization" (1).

AFPP's have traditionally been a low cost method of providing minicomputer users with the number crunching capability of small mainframe computers. However, the low cost is only low relative to the enormous cost of a mainframe computer and a typical AFPP is still expensive in an absolute sense at £30k-£50k for a system with enough options to perform useful work. In recent years the performance of low cost microcomputer systems has reached that of popular midrange minicomputers which might cost upwards of four times the price of the microcomputer. To take a specific example the performance of a 4MHz Z80A microcomputer system running Microsoft Fortran slightly exceeds that of a standard DEC PDP-11/34 running Fortran V2.1 under RT-11 V3B for most scientific applications. However if the PDP-11/34 is configured with a floating point processor then it will run programs which contain a significant amount of floating point arithmetic up to five times faster than the microcomputer.

In order to make a microcomputer an attractive proposition for scientific work there must be some means of enhancing its floating point performance. There are multididinous ways of achieving this and some of the possibilities will be discussed in increasing order of performance.

Scalar Arithmetic Processors

The lowest cost method of improving the arithmetic capability of a microcomputer is to interface a scientific calculator type chip with the data bus of the CPU. The National Semiconductor MM57109 number crunching chip (2) is designed for just this purpose and its instruction set includes basic arithmetic operations, transcendental functions and data manipulation operations. Data are entered into the MM57109 one digit at a time (8 mantissa digits, 2 exponent digits) and a six bit instruction is loaded when the data is in place. A simple floating point instruction such as multiply takes an average of 32 mS exclusive of the time required to load data and retrieve results. The MM57109 is not an attractive option from the speed point of view because a 4MHz Z80A can perform a floating point multiply (32 bit) in much less than 32 mS. The MM57109 does save both the time required to code floating point arithmetic routines and the space required to store them but our prime consideration here is speed.

The second possibility for improving microcomputer floating point performance lies with the North Star Hardware Floating Point Board (3). This device executes floating point, add, subtract, multiply and divide with up to twelve decimal digits of precision. One byte of data is reserved for the exponent and the other six for the mantissa of each floating point number.

Each byte of data in the mantissa contains two BCD digits and the exponent byte contains the mantissa sign (MSB) and the exponent in excess 64 binary. Mean execution times for floating point add, subtract, multiply and divide (6 digit) are 6, 11, 194 and 175μS again exclusive of data transfer times. The speeds of execution with this device are attractive and certainly less than could be achieved by an eight bit microprocessor alone. Unattractive features in this case are the lack of transcendental functions and a rather odd floating point number format. These drawbacks could be tolerated if there was not a more attractive alternative.

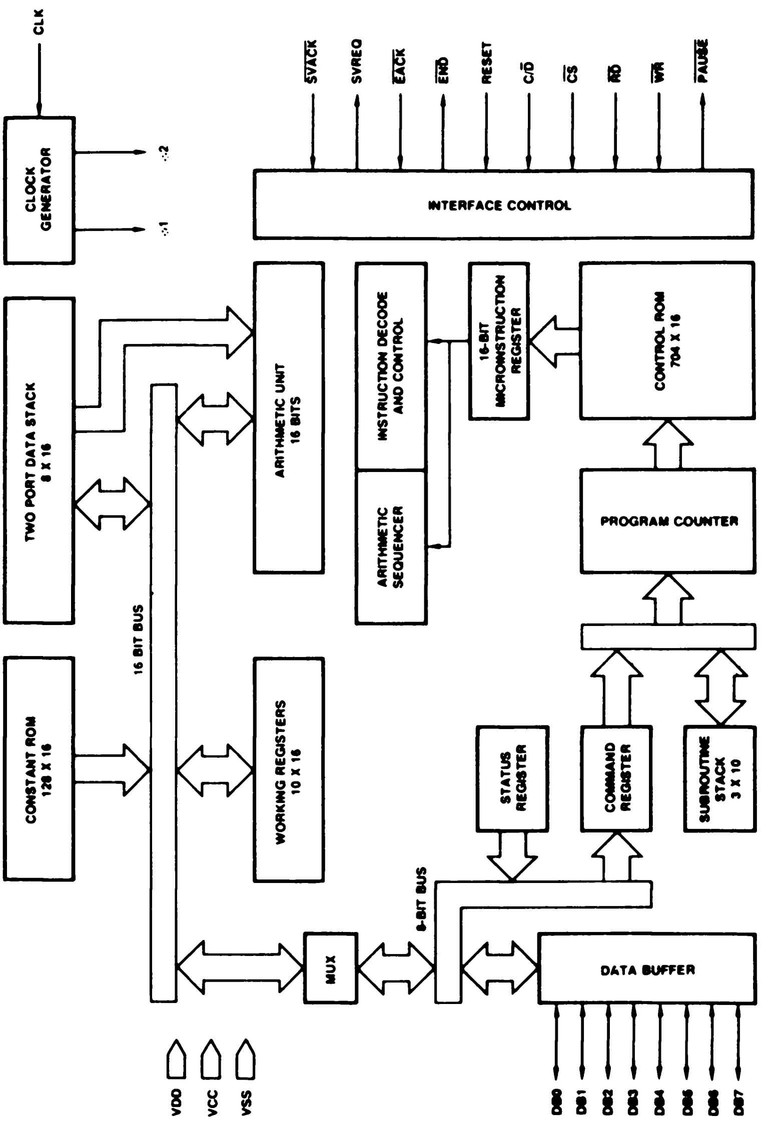
The third floating point processor option uses the Advanced Micro Devices Am 9511A arithmetic processor chip. This chip is available in several types; the Am9511, Am9511A-DC, Am9511A-1DC and Am9511A-2DC. The Am9511A-DC obsoleted the Am9511 and the DC, 1DC and 2DC variants refer to 2, 3 and 4MHz clock speeds. All further references in text and figures are to the Am9511A-1DC 3MHz chip. A block diagram of the functional units of this chip is shown in Figure 1 and its instruction set is shown in Table I. Mean times for each instruction are shown in Table II and these obviously vary for different patterns in the fixed or floating point arguments. Each floating point number is 32 bits long and the detailed format is shown in Figure 2. Data are entered into the Am9511A by pushing one byte at a time onto the 16 byte operand stack and results retrieved by byte wide pops of the same stack. Single byte commands enter the Am9511A via the same data lines (DB0-DB7) as the operands but are routed to the internal command register and execution proceeds using the top one or two operands on the stack. When instruction execution is complete the "consumed" operands are popped off the stack and the result pushed onto the top of the stack. The Am9511A is easily interfaced to most popular 8-bit microcomputers and Figure 3 shows one method of connection to the Zilog Z80A. Thus far the Am9511A appears to have satisfied requirements for range of operations, ease of interfacing and speed. The remaining requirement is for ease of software interfacing to a high level language.

The remaining options for improving microcomputer floating point performance which will be considered lead to much higher performance than the Am9511A option but have one or more major drawbacks.

The Intel 8086 or 8088 microprocessors could be used in conjunction with the Intel 8087 floating point processor chip (4) which is probably twice as fast as the Am9511A for on-chip operations and includes extended precision arithmetic in its instruction set. Unfortunately the 8087 was only laid down on paper, not silicon, when this work started. The 8087 is now (January 1981) available in sample quantities at a price far in excess of the Am9511A. In addition to the price and availability problem the instruction set of the 8087 is less suited to chemical computations than the Am9511A in that many transcen-

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch013

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch013



Advanced Micro Devices

Figure 1. Block diagram of the functional units of the Am9511A arithmetic processor integrated circuit (11).

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch013

Table I

Command Mnemonics in Alphabetical Order

ACOS	ARCCOSINE
ASIN	ARCSINE
ATAN	ARCTANGENT
CHSD	CHANGE SIGN DOUBLE
CHSF	CHANGE SIGN FLOATING
CHSS	CHANGE SIGN SINGLE
COS	COSINE
DADD	DOUBLE ADD
DDIV	DOUBLE DIVIDE
DMUL	DOUBLE MULTIPLY LOWER
DMUU	DOUBLE MULTIPLY UPPER
DSUB	DOUBLE SUBTRACT
EXP	EXPONENTIATION (e^{x})
FADD	FLOATING ADD
FDIV	FLOATING DIVIDE
FIXD	FIX DOUBLE
FIXS	FIX SINGLE
FLTD	FLOAT DOUBLE
FLTS	FLOAT SINGLE
FMUL	FLOATING MULTIPLY
FSUB	FLOATING SUBTRACT
LOG	COMMON LOGARITHM
LN	NATURAL LOGARITHM
NOP	NO OPERATION
POPD	POP STACK DOUBLE
POPF	POP STACK FLOATING
POPS	POP STACK SINGLE
PTOD	PUSH STACK DOUBLE
PTOF	PUSH STACK FLOATING
PTOS	PUSH STACK SINGLE
PUPI	PUSH π
PWR	POWER (X^{Y})
SADD	SINGLE ADD
SDIV	SINGLE DIVIDE
SIN	SINE
SMUL	SINGLE MULTIPLY LOWER
SMUU	SINGLE MULTIPLY UPPER
SQRT	SQUARE ROOT
SSUB	SINGLE SUBTRACT
TAN	TANGENT
XCHD	EXCHANGE OPERANDS DOUBLE
XCHF	EXCHANGE OPERANDS FLOATING
XCHS	EXCHANGE OPERANDS SINGLE

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch013

Table II

Execution Times in Microseconds

Mnemonic	Hex Code	Execution Time (μ secs)
ACOS	06	2101 - 2761
ASIN	05	2077 - 2646
ATAN	07	1664 - 2179
CHSD	34	9
CHSF	15	5 - 7
CHSS	74	7 - 8
COS	03	1280 - 1626
DADD	2C	7
DDIV	2F	65 - 70
DMUL	2E	65 - 70
DMUU	36	61 - 73
DSUB	2D	13
EXP	0A	1265 - 1626
FADD	10	18 - 123
FDIV	13	51 - 61
FIXD	1E	30 - 112
FIXS	1F	30 - 71
FLTD	1C	19 - 114
FLTS	1D	21 - 52
FMUL	12	49 - 56
FSUB	11	23 - 123
LOG	08	1491 - 2377
LN	09	1433 - 2319
NOP	00	1
POPD	38	4
POPF	18	4
POPS	78	3
PTOD	37	7
PTOF	17	7
PTOS	77	5
PUPI	1A	5
PWR	0B	2763 - 4011
SADD	6C	5 - 6
SDIV	6F	28 - 31
SIN	02	1265 - 1603
SMUL	6E	28 - 31
SMUU	76	27 - 33
SQRT	01	261 - 290
SSUB	6D	10 - 11
TAN	04	1631 - 1962
XCHD	39	9
XCHF	19	9
XCHS	79	6

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch013

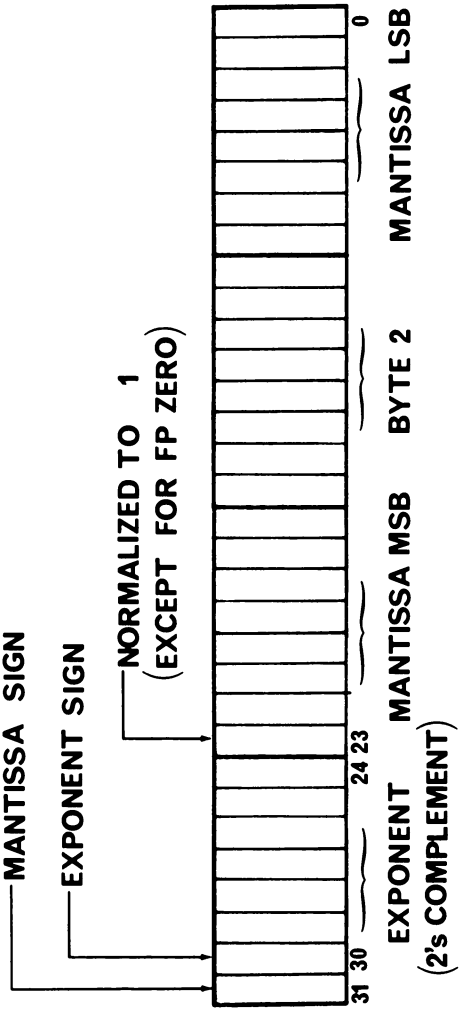


Figure 2. The floating point number representation format used by the Am9511A.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch013

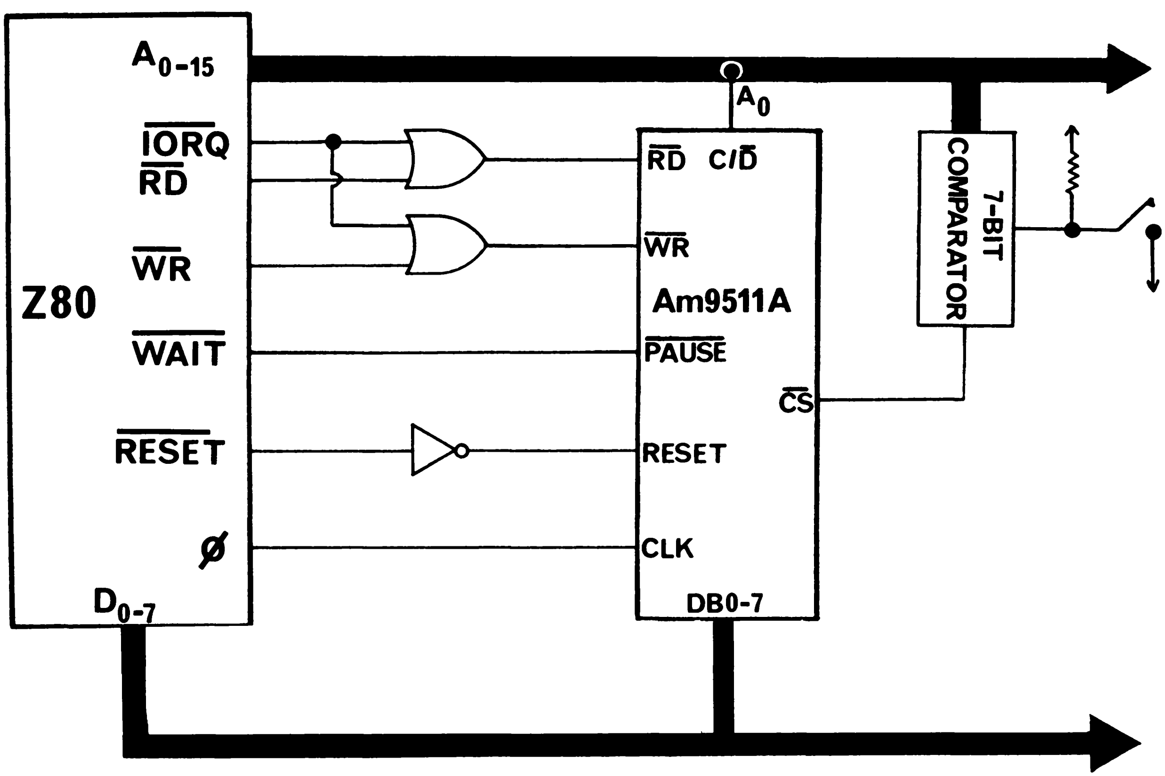


Figure 3. Schematic logic diagram of a hardware interface between a Zilog Z80A microprocessor and the Am9511A arithmetic processor unit.

In Supercomputers in Chemistry; Lykos, P., el al.;
ACS Symposium Series; American Chemical Society: Washington, DC, 1981.

dental functions must be computed off chip using a minimal set of on chip functions. For these operations (e.g. ASIN, ACOS) the 8087 is much slower than the Am9511A. Furthermore the 8086 and 8088 are the end of a series of processors based on the Intel 8080 and are not in any way upwards compatible with Intel's new products. (Contrast this with the Zilog Z8000 whose instruction set is a superset of the Z80's at the assembler level). Enhancement of an 8086/8087 design would therefore be somewhat difficult. The final point which mitigates against the 8086/8087 is the lack of a supporting high level language at present. If or when such a language is available it is unlikely to be cheap. The plus points for the 8086/8087 are therefore speed and extended precision arithmetic but these are outweighed by the disadvantages discussed above.

Either of the remaining two options to be discussed would provide at least an order of magnitude increase in speed on the Am9511A and are much coveted by the author, but unfortunately they would require financial resources on a very large scale for development into useable floating point arithmetic processors.

The first of these options is the American Microsystems Inc.'s S2811 microprocessor chip which contains 256 bytes each of data ROM and RAM, 256 x 17 instruction ROM, a 300nS 12 x 12 multiplier and an add/subtract unit. The chip runs at a 20MHz (minimum) clock speed and all instructions (including multiply) execute in 300 nS. Interfacing to a control processor is simple and straightforward. The whole processor is extensively pipelined and would perform a 32 bit floating point multiply in around 3 microseconds as opposed to 50 microseconds on the Am9511A. There is also sufficient space in the microcode ROM to emulate the entire Am9511A instruction set which makes the S2811 appear to be a very attractive proposition; so what's the drawback? Unfortunately the S2811 is only mask programmable, at the factory, and this involves a fee of £5000 plus a guaranteed minimum order of 500 chips at around £200 each! AMI have no immediate plans to release versions of the S2811 using EPROM or RAM for instruction memory, which would make it economic to use in small quantities. The S2811 is at present therefore restricted to large commercial users except in one specialized instance. The S2814 performs real or complex, forward or reverse fast fourier transforms using 32 bit floating point arithmetic and a single chip will perform 32 complex, forward transformations in 1.5mS whilst an array of 32 chips takes 3.35mS to perform a 1024 point transformation! This chip is available ready programmed at around £400 for single quantities.

The final option considered involves the use of the TRW MPY-24 bit multiplier in conjunction with 2900 series bit-slice microprocessors in order to design a floating point unit from the ground up. This would result in a unit similar to the Floating Point Systems FPS-100 AFPP. The design principles for such an approach are well known, but somewhat time-consuming and

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch013

relatively expensive. This option would be the fastest of all those discussed and would yield floating point add, subtract, multiply and divide times of around 100nS.

As a general rule of thumb the breakpoint for academic development of floating point units in non-engineering or computer science environments is currently in the 1-10μS range; in order to go much below this requires funds and personnel on an increasingly large scale.

Almost all applications programming in chemistry, and structural chemistry in particular, is performed in the FORTRAN language and the molecular mechanics calculations for which this hardware/software design exercise was undertaken is no exception. There are two problems which must be solved in order to build a FORTRAN microcomputer system with good scalar and array computational performance and these are firstly, the design of an efficient scalar processor with a transparent FORTRAN interface to the hardware and secondly, the design of an efficient AFPP, VP or AP and supporting library of FORTRAN callable subroutines.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch013

Scalar FORTRAN System

The system was designed around a Vector Graphics Inc. Vector MZ microcomputer consisting of 4 MHz CPU, 48k bytes of memory and two 315k byte Micropolis mini floppy disc drives. The system utilises the S-100 bus structure and the interface for an Am9511A is shown in Figure 4. The only suitable FORTRAN compiler which would run under the Digital Research CP/M operating system is Microsoft's F80 package which in the event proved to be a very sound and robust piece of software. Microsoft also provide sufficiently detailed documentation of the F80 run time library to enable a relatively problem free replacement of the software floating point arithmetic routines by calls to subroutines which invoke the Am9511A. Provided then, that the Microsoft naming and calling procedures are adhered to the end result is an F80 system which will run previously prepared programs without any changes in the FORTRAN code being required. These same programs will now though run about an order of magnitude faster. Detailed comparisons of the execution times for various F80 functions and library subroutines are given in Table III. It is obvious from Table III that the Am9511A has been used to implement a number of functions not supplied by Microsoft. The times given are for 10,000 function evaluations.

The only unfortunate feature of this whole implementation is the fact that the AMD floating point format (Figure 2) and the Microsoft floating point format (Figure 5) is different, requiring a Microsoft to AMD conversion before using any of the Table III functions/subroutines and an AMD to Microsoft conversion before returning to the F80 calling program. The source listing of a Z80 assembly level subroutine to perform this conversion in the forward (Microsoft → AMD) direction is shown in Table IV. The forward and reverse conversions take slightly

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch013

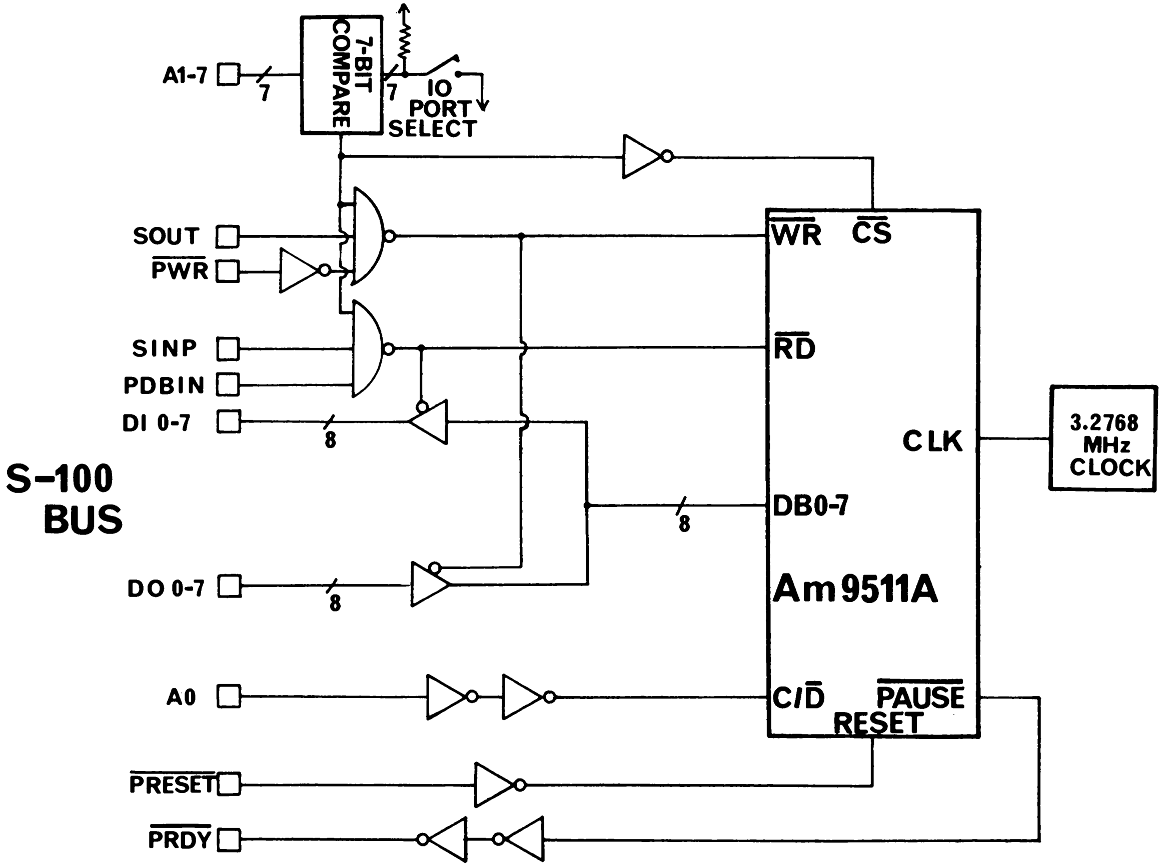


Figure 4. Logic diagram of an Am9511A arithmetic processor unit to S-100 bus hardware interface.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch013

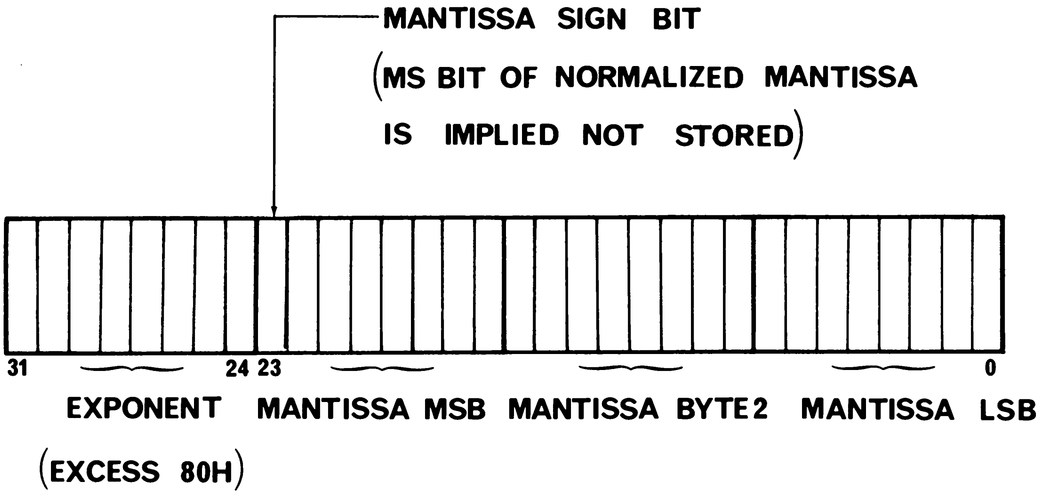


Figure 5. The floating point number representation format used by Microsoft in the F80 Fortran compiler.

Table III

Benchmark Timings (Seconds)

FUNCTION	MSP-9500	SOFTWARE	DESCRIPTION
SQRT	5.2	126.2	SQUARE ROOT
SIN	13.7	109.0	SINE
COS	14.9	113.7	COSINE
TAN	17.4	*	TANGENT
ASIN	21.6	*	ARC SINE
ACOS	22.4	*	ARC COSINE
ATAN	18.2	103.2	ARC TANGENT
ALOG10	17.2	138.7	LOG TO BASE 10
ALOG	16.8	138.7	LOG TO BASE e
EXP	16.0	119.9	e TO POWER
SINH	32.8	*	HYPERBOLIC SINE
COSH	32.4	*	HYPERBOLIC COSINE
TANH	20.5	144.4	HYPERBOLIC TANGENT
FLOAT	2.4	2.8	INTEGER -> REAL
INT	2.1	6.5	REAL -> INTEGER
ABS	1.5	1.3	ABSOLUTE VALUE
AINT	3.8	4.9	REAL->REAL TRUNCATION
NINT	3.7	*	NEAREST INTEGER
AMIN0	9.5	12.0	REAL MIN OF INT LIST
AMAX0	8.8	11.0	REAL MAX OF INT LIST
MIN1	14.7	20.4	INT MIN OF REAL LIST
MAX1	15.4	19.2	INT MAX OF REAL LIST
AMIN1	16.2	15.2	REAL MIN OF REAL LIST
AMAX1	13.1	14.4	REAL MAX OF REAL LIST
AMOD	8.8	28.2	REAL REMAINDER
DIM	4.5	5.3	REAL POSITIVE DIFF.
ATAN2	19.8	123.9	ARC TANGENT a1/a2
MOD	1.3	7.2	INTEGER REMAINDER
$M9	0.9	2.7	INTEGER*INTEGER
$D9	1.0	5.7	INTEGER/INTEGER
$AB	4.5	3.1	REAL+REAL
$SB	5.3	4.5	REAL-REAL
$MB	4.7	6.7	REAL*REAL
$DB	5.3	19.4	REAL/REAL
$EB	31.1	245.8	REAL**REAL
$CJ	2.7	2.7	REAL -> LOGICAL
$EA	30.9	66.2	REAL**INTEGER
$NB	0.5	0.9	NEGATE
$E9	1.6	5.3	INTEGER**INTEGER
RAN	5.7	*	RANDOM NUMBER 0 -> 1.

* - Operation not available

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch013

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch013

Table IV

Microsoft to AMD Floating Point Format Conversion

```
;VPPUT SUBROUTINE
;INVOKED FROM FORTRAN BY CALL VPPUT(A,C,N) WHERE A IS AN ARRAY
;IN HOST FORMAT AND C IS THE CONVERTED ARRAY IN VPU FORMAT. N
;IS THE LENGTH OF THE ARRAYS. A MAY BE CONVERTED IN PLACE IF
;THE SYMBOLIC ADDRESSES A AND C ARE THE SAME

        .Z80
        EXT     OFLOV,UFLOV,ABASE,CBASE,LD3ARG,NCOMP

VPPUT:: PUSH    DE          ;SAVE DE ON STACK
        CALL    LD3ARG      ;(ABASE)<-HL , (CBASE)<-DE , DE=NCOMP
        LD     (NCOMP),DE;SAVE NUMBER OF ARRAY ELEMENTS IN MEMORY
        POP     DE          ;DE POINTS TO BOTTOM BYTE OF C ARRAY
        SUB     A           ;CLEAR CARRY FLAG
        SBC     HL,DE       ;ARE A AND C ARRAY BASE ADDRESSES EQUAL?
        JR      Z,NOMOV     ;YES CONVERT A ARRAY IN PLACE
        LD     HL,(NCOMP);GET NO. OF ARRAY ELEMENTS INTO HL
        ADD     HL,HL       ;AND MULTIPLY BY TWO
        ADD     HL,HL       ;AND BY TWO AGAIN TO GET NO. BYTES IN ARRAY
        LD      C,L         ;BC IS BYTE COUNTER IN LDIR SO LOAD
        LD      B,H         ;HL INTO BC
        LD     HL,(ABASE);HL->A ARRAY,DE->C ARRAY,BC CONTAINS LENGTH
        LDIR                ;MOVE (HL)->(DE) BC TIMES
NOMOV:  LD     HL,(CBASE);GET READY TO CONVERT C ARRAY IN PLACE
        LD     BC,(NCOMP);BY SETTING UP HL AND BC(BC=0 NOW)
CONLOOP:INC     HL          ;HL->BYTE 2 OF FP WORD
        INC     HL          ;HL->BYTE 3 OF FP WORD
        PUSH    HL          ;HL->MS MANTISSA BYTE-SAVE IT!
        EXX                 ;GET HL',BC',DE'
        POP     HL          ;HL'->MS MANTISSA BYTE
```

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch013

```
        EXX                     ;BACK TO HL,BC,DE
        INC     HL              ;HL->BYTE 4(I.E. EXPONENT) OF MICROSOFT FP WORD
        LD      A,(HL)          ;LOAD MICROSOFT FORMAT EXPONENT INTO ACC.
        ADD     A,00H           ;ADD ZERO TO ACC-SETS CONDITION CODES
        JR      Z,NXNUM         ;ZER0 SAME IN BOTH FORMATS-DONT CONVERT
        CP      80H             ;IS EXP GE 80(HEX)
        JP      P,PLUS          ;YES-GO CHECK SIZE OF EXPONENT
        CP      40H             ;IS EXPONENT GT 40(HEX)
        JP      M,UFLOV         ;NO-NUMBER TOO SMALL FOR APU FORMAT
        JR      CNVRT           ;YES-NUMBER WITHIN RANGE-GO CONVERT
PLUS:   CP      0BFH            ;IS EXPONENT GT 0BF(HEX)?
        JP      P,OFLOV         ;YES-NUMBER TOO LARGE FOR APU FORMAT
        SUB     80H             ;NO-CNVRT TO 2'S COMPLEMENT
CNVRT:  LD      (HL),A          ;STORE CNVRTD EXPONENT-NO CONV FOR NEG EXP
        EXX                     ;GET HL',BC',DE'
        BIT     7,(HL)          ;TEST SIGN BIT OF MICROSOFT MANTTISSA 1= -
        JR      NZ,SET1         ;NEG MANTISSA-GO FIX SIGN IN APU FP FORMAT
        SET     7,(HL)          ;MS MANTISSA BIT ALWAYS SET IN APU FP FORMAT
        EXX                     ;GET HL,BC,DE
        RES     7,(HL)          ;POS MANT- RESET MS APU FMT EXPONENT BIT
        JP      NXNUM           ;CONVERSION TO APU FORMAT COMPLETE
SET1:   EXX                     ;GET HL,BC,DE
        SET     7,(HL)          ;MS MANT BIT ALREADY SET-SET APU MANT SIGN BIT
NXNUM:  INC     HL              ;HL->LS MANTISSA BYTE OF NEXT FP NUMBER
        DEC     BC              ;DECREMENT NO. OF ARRAY ELEMENTS LEFT TO FIX
        LD      A,C             ;LOAD C INTO ACC. AND OR IT
        OR      B               ;WITH B.RESULT IS ZERO ONLY IF B=C=0
        JP    NZ,CONLOOP;RESULT NOT ZERO-GO PROCEESS MORE ELEMENTS
        RET                     ;ALL CONVERTED-RETURN

        END
```

longer in total than a floating point addition on the Am9511A and so floating point addition and subtraction do not obtain full benefit from the use of the Am9511A. Fortunately the conversion times are a small proportion of the execution times for all other important Am9511A functions. No conversions are of course required for integer operations.

One further point should be mentioned and this is the fact that the Am9511A is restricted to 32 bit floating point arithmetic. This is quite adequate for all X-ray crystallographic and most molecular mechanics calculations. In general at least 48 bit representations are required for direct, rather than iterative, matrix operations which are widely used in quantum chemistry for example. This problem could be overcome by means of the Am9512, a companion chip to the Am9511A, which has 64 bit add, subtract, multiply and divide in its instruction set. The Am9512 is however atrociously slow in 64 bit operations and has no on-chip 64 bit transcendental functions. Use of the Am9512 was therefore not considered further.

A natural progression from the use of a single Am9511A arithmetic processor chip, which can enhance the performance of F80 functions/subroutines by a factor of up to 25x, is to use a number of such chips in a VP or AP architecture.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch013

Vector Arithmetic Processors

Having previously decided upon the use of the Am9511A the problem then is to find a satisfactory method of interfacing multiple devices to the S-100 bus. The hardware adopted is illustrated schematically in Figure 6. The Am9511A's are still accessed via I/O ports, one for data (low address) and one for commands (high address) mediated by address line A0, but the I/O address decoder circuitry (IC14, IC16 and SW1) defines a switch selectable block of 16 consecutive I/O ports any one of which is selectable during a CPU input/output operation. The system clock runs at 3MHz asynchronously of the Z80A CPU clock and data are buffered onto and from the S-100 bus by IC18 and IC19. The Am9511A read and write lines are driven by the S-100 bus SINP and PDBIN and SOUT and PWR signals (5). The arithmetic processor handshaking lines (PAUSE) are ANDED together by IC15 & IC1 so that lowering of any of the PAUSE lines will cause the Z80A to temporarily suspend execution, until all PAUSE lines are high again, in order to allow sufficient time for data transfer. Light emitting diodes are used to monitor the Am9511A END (of operation) lines and the S-100 bus PRDY line (useful for checking that everything is running).

It is possible to improve the performance of the hardware (hereafter called the MVP-9500) by providing interrupt facilities to signal completion of operation(s) or to request more data and also by moving data via direct memory access rather than under Z80 program control. These, and other refinements, were

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch013

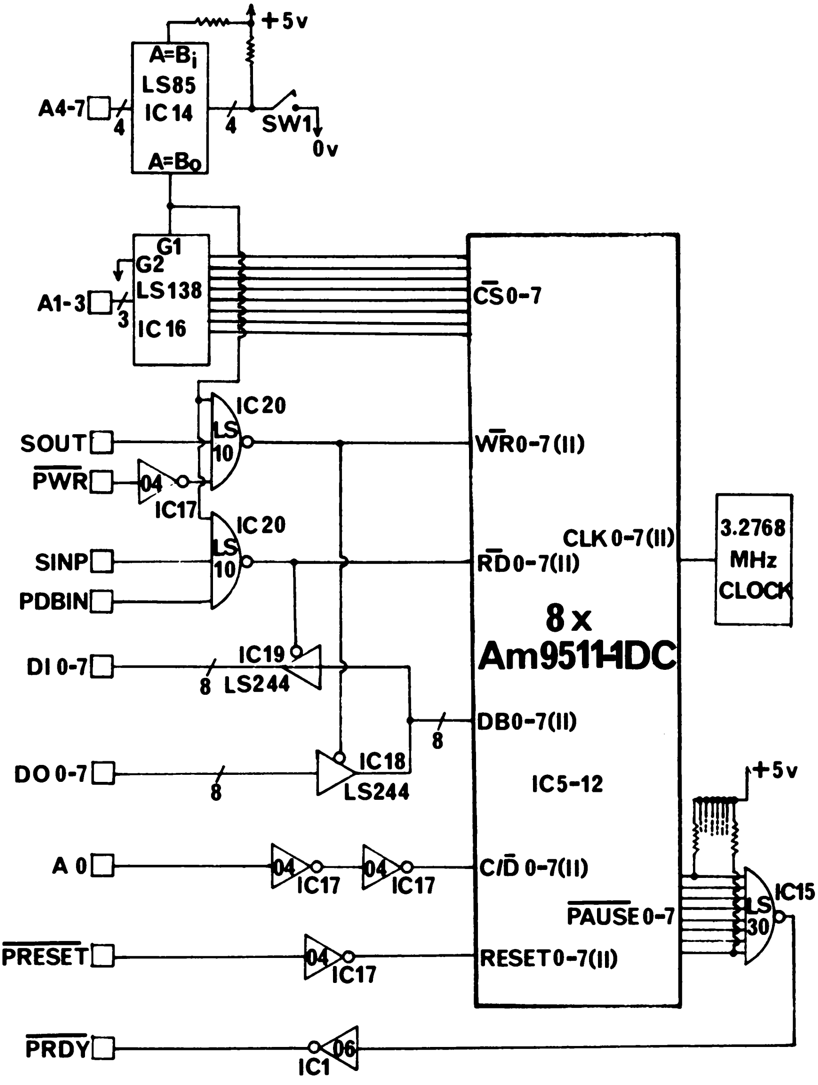


Figure 6. Logic diagram of the MVP-9500/8 vector processor card for S-100 bus microcomputers.

deferred to a later date and first priority was given to getting the basic MVP-9500 up and running.

There are two modes in which the MVP-9500 can be used in conjunction with the Z80A. In the first mode one or more arguments are loaded into each Am9511A in turn, and when all are loaded then command bytes are sequentially output from the Z80A to each Am9511A. This procedure has the drawback that the APU's spend a significant time loaded with data which is not being operated upon. An alternative approach involves "hiding" data transfer operations behind APU arithmetic operations. The first Am9511A is loaded with arguments and then a command to start it operating on the arguments. As soon as APU number one is executing, APU number two is loaded with data and set running and so on until all of the APU's are concurrently executing instructions. The APU's are then polled to see if they are busy, in the same order in which they were loaded, and reloaded with more data and a command as they finish the previous operation. Obviously this type of structure will give maximum efficiency if the APU load times are short compared with the instruction execution times. This is in fact the case as it takes around 30μS to load the APU with data and set it running compared with APU instruction execution times of roughly 50-4000μS. The MVP-9500 will obviously be more efficient with calculations involving transcendental functions than it will be with calculations involving floating point add and subtract.

The remaining obstacle to producing a working vector processor system is the library of FORTRAN callable subroutines.

Vector FORTRAN System

The MVP-9500 system contained all of the scalar system components with the addition of an MVP-9500/8 printed circuit board and the VPLIB library of F80 callable subroutines; it is shown schematically in Figure 7.

The design criteria for the VPLIB library were as follows: It should be possible to add Am9511A chips to the system as funds permitted without having to make any changes to the software; the library should incorporate the best features of those produced by Floating Point Systems for the AP120B and by CSPI for the MAP-200 series of AFPP's; the library should contain special subroutines for use in coordinate geometry and molecular mechanics calculations; a solution should be found to the problem of time consuming floating point format conversions; and finally the library should be as modular as possible without sacrificing efficiency by using large numbers of subroutine calls or the like (i.e. forget about structural programming as it's speed not beauty we're concerned with!).

The problem of floating point format differences was the most pressing one and was solved in the following way. All data arrays to be operated upon by the MVP-9500 are converted to the

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch013

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch013

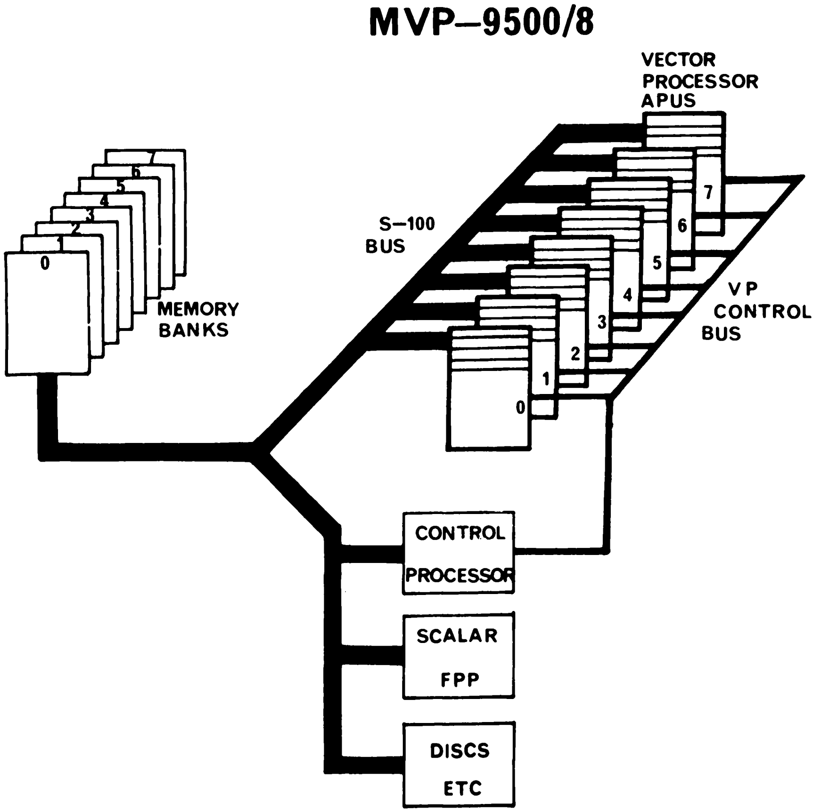


Figure 7. Block diagram of the MVP-9500/Vector MZ vector processor microcomputer system.

AMD format at the start of a series of vector operations, usually in place as the Z80A and MVP-9500 share memory. In order to permit scalar processing interspersed between vector calls a number of subroutines for performing arithmetic operations without format conversion are provided. The F80 code for the multiplication of two random matrices (Table V) illustrates this point and also the software independence of the number of Am9511A's used. R(1) and R(2) are random number generator seeds and exist in Microsoft floating point format at the start of the program. The call to VINIT initializes VPLIB with the number of Am9511A's actually present, five in this case. The call to VPPUT converts the array R, in place, to AMD format and the calls to VRAND use R(1) and R(2) as seeds to a random number generator which places 1600 random numbers between 0.0 and 1.0 in array A and a similar number in array B, in AMD format. The pause statements are for timing purposes and MMUL multiplies the matrices A and B together and places the result in matrix C. A, B and C are in AMD format at this stage and must be converted to Microsoft format, by calls to VPGET, before they are printed out. The similarity to FPS and CSPI Inc. routines is obvious.

The question of special purpose routines for structural chemistry will be discussed later and attention will now be directed to the construction of the library. It would have been possible to write a small number of strategic routines in Z80A/MVP-9500 assembly language and the remainder of the library in FORTRAN code which made use of the assembler level nucleus. This would certainly have resulted in an operational library in the shortest possible time but at considerable sacrifice in efficiency. All of the VPLIB subroutines were therefore written entirely in Z80A/MVP-9500 assembly language and this produced modules which contained, on average, one third of the assembler instructions produced by F80 for the same operation coded in FORTRAN. In addition to these straightforward savings a considerable amount of hand optimization was possible on the assembler level subroutines.

The above mentioned points are illustrated by a discussion of the VPLIB subroutine(s) for evaluating transcendental functions when supplied with a single floating point argument (Table VI). The entry points are the labels with double colons (e.g. VSIN) and the accumulator A is loaded with the Am9511A opcode for the particular operation. The code starting at VECTOR (a local symbol) is then common to all operations. The Am9511A opcode is stored in the alternate accumulator A' (the Z80 has two duplicate sets of registers) and LD3ARG moves pointers to argument addresses into a local area (ABASE for the source vector, CBASE for the destination vector, and the 16 bit register DE contains the number of components in the vector). The 16 bit register HL is loaded with the current address in the A array and a call to VLD1EX loads each APU and sets it executing as previously described. On exit from VLD1EX HL points to the

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch013

Table V

Fortran Code for Matrix Multiplication using VPLIB

```
      DIMENSION R(3),A(40,40),B(40,40),C(40,40)
      R(1)=1.
      R(2)=2.
      R(3)=3.
      CALL VINIT(5)
      CALL VPPUT(R,R,3)
      CALL VRAND(R(1),A,1600)
      CALL VRAND(R(2),B,1600)
      PAUSE START
      CALL MMUL(A,B,C,40,40,40)
      PAUSE STOP
      CALL VPGET(A,A,1600)
      CALL VPGET(B,B,1600)
      CALL VPGET(C,C,1600)
      WRITE(5,1) ((A(I,J),J=1,40),I=1,40)
      WRITE(5,2) ((B(I,J),J=1,40),I=1,40)
      WRITE(5,3) ((C(I,J),J=1,40),I=1,40)
1     FORMAT(10F8.3)
2     FORMAT(10F8.3)
3     FORMAT(10F8.3)
      STOP
      END
```

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch013

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch013

Table VI

```
;VECTOR TRANSCENDENTAL FUNCTIONS ROUTINE
;FORTRAN CALL IS, FOR EXAMPLE, CALL VSQRT(A,C,N), WHERE A IS AN
;ARRAY CONTAINING THE ARGUMENTS AND C IS AN ARRAY WHERE THE RESULTS
;ARE TO BE STORED. N IS THE NUMBER OF COMPONENTS IN EACH ARRAY TO BE OPERATED ON.
        .Z80
        EXT     ABASE,CBASE,ERR,STRES,LD3ARG,NPROC
BASEVP  EQU     0E0H    ;BASE OF VP-9500 I/O ADDRESSES

VSQRT:: LD      A,01H   ;APU SQRT OPCODE->ACC.
        JP      VECTOR  ;COMMON CODE
VSIN::  LD      A,02H   ;APU SIN OPCODE->ACC.
        JP      VECTOR  ;COMMON CODE
VCOS::  LD      A,03H   ;APU COS OPCODE->ACC.
        JP      VECTOR  ;COMMON CODE
VTAN::  LD      A,04H   ;APU TAN OPCODE->ACC.
        JP      VECTOR  ;COMMON CODE
VASIN:: LD      A,05H   ;APU ASIN OPCODE->ACC.
        JP      VECTOR  ;COMMON CODE
VACOS:: LD      A,06H   ;APU ACOS OPCODE->ACC.
        JP      VECTOR  ;COMMON CODE
VATAN:: LD      A,07H   ;APU ATAN OPCODE->ACC.
        JP      VECTOR  ;COMMON CODE
VLOG::  LD      A,08H   ;APU LOG OPCODE->ACC.
        JP      VECTOR  ;COMMON CODE
VLN::   LD      A,09H   ;APU LN OPCODE->ACC.
        JP      VECTOR  ;COMMON CODE
VEXP::  LD      A,0AH   ;APU EXP OPCODE->ACC.

VECTOR: EX      AF,AF'  ;SAVE APU OPCODE IN AF'
        CALL    LD3ARG  ;GET ARRAY ADDRESSES AND NO. OF COMPONENTS
VTRNLP: LD    HL,(ABASE);HL->NEXT COMPONENT OF A
        CALL    VLD1EX  ;LOAD VP-9500 AND EXECUTE
```

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch013

```
        LD      (ABASE),HL;HL->NEXT COMPONENT-SAVE IT!
        CALL      ERR     ;CHECK EACH APU FOR ERRORS
        LD      HL,(CBASE);HL->NEXT COMPONENT OF C
        CALL      STRES   ;UNLOAD VP-9500, RESULTS TO ARRAY C
        LD      (CBASE),HL;HL->NEXT COMPONENT OF C-SAVE IT!
        LD        A,D     ;LOAD HI-BYTE OF COMPONENT COUNTER INTO A
        OR        E       ;AND OR WITH LO-BYTE, RESULT ZERO ONLY IF DE=0
        JP       NZ,VTRNLP;MORE COMPONENTS TO CALCULATE?
        RET               ;NO,RETURN TO FORTRAN CALLING PROGRAM

VLD1EX::LD        C,BASEVP;LOAD C WITH VP-9500 BASE I/O ADDRESS
        EXX               ;SWITCH TO ALTERNATE REGISTER SET
        LD        E,00H   ;ZERO E', THE APU COUNTER.
        EXX               ;SWITCH BACK TO HL, BC, DE
PROCLP: LD        B,04H   ;ONE FP WORD PER APU
        OTIR              ;LOAD FP WORD ONTO EACH APU STACK
        INC       C       ;C->APU COMMAND PORT
        EX        AF,AF'  ;RETRIEVE OPCODE FROM AF'
        OUT       (C),A   ;AND OUTPUT IT TO APU COMMAND PORT
        EX        AF,AF'  ;RESAVE OPCODE IN AF'
        LD       A,(NPROC);LOAD NUMBER OF APU'S AVAILABLE INTO ACC.
        DEC       DE      ;DECREMENT COMPONENT COUNTER
        EXX               ;SWITCH TO ALTERNATE REGISTER SET
        INC       E       ;AND INCREMENT THE APU COUNTER E'
        CP        E       ;HAVE NPROC APU'S EXECUTED?
        EXX               ;BACK TO HL,BC,DE, FLAG REGISTER UNCHANGED
        RET       Z       ;RETURN IF YES
        LD        A,D     ;CHECK TO SEE IF BOTH HI- AND
        OR        E       ;LO-BYTE OF DE ARE ZERO
        RET       Z       ;DE=0 SO RETURN
        INC       C       ;MORE COMPONENTS,C->NEXT DATA PORT,HL->NXT. COMP
        JP        PROCLP  ;GO AND PROCESS NEXT COMPONENT
        END
```

next A component to be processed and it is saved in memory at ABASE. The ERR subroutines check each APU for errors such as overflow, attempt to divide by zero, argument out of range etc. and returns if things are going according to plan, or aborts and prints an error message if they aren't! HL is then loaded with the current address in the destination array C and STRES unloads the APU generated results into memory. On exit HL again points to the next component, but of the C array, and is saved in CBASE. The component counter DE, suitable decremented in VLD1EX, is then checked to see if it is zero (two statements are required because Z80 16 bit operations don't always affect the flags register!). If it is zero and all components of the A array have been processed and stored in the C array then a return is made to the calling program otherwise, another set of components is processed.

The VLD1EX subroutine operates as follows: The 8 bit C register is loaded with the base I/O address of the vector of Am9511A's. The addresses are contiguous and run in the order DATA PORT 1, COMMAND PORT 1, DATA PORT 2, COMMAND PORT 2, etc. EXX switches to the alternate register set and E', the counter for the number of APU's loaded, is zeroed and EXX switches back to the ordinary register set. The OTIR instruction is a block output from memory pointed to by register HL to the I/O port pointed to by the C register. After each output HL is incremented and B decremented and the process repeated until B is zero. Having loaded one FP word (4 bytes) onto the APU, C is incremented to point to the command port of the same '9511 and the EX instruction switches to the alternate accumulator which contains the APU opcode. (EX, AF, AF' switches accumulators independently of EXX which switches between register sets HL, BC, DE and HL', BC', DE'). The OUT instruction then sets the APU operating on the current argument. The following EX resaves the APU opcode in A' and the ordinary accumulator A is loaded with the number of '9511's available (VINIT loads location NPROC with this value). The component counter DE is decremented by one to reflect the fact that a component of the A array has been processed. EXX gives access to E' which is incremented and compared with the accumulator which contains the number of '9511's, EXX returns to the ordinary register set without affecting the flag, or condition code, register F. If all the Am9511's available are loaded and running then a return is made to the VSQRT∷, or whatever, subroutine. If not all '9511's are loaded the component counter is checked for zero and a return made to the VSQRT∷ etc subroutine. If neither of the previous conditions is fulfilled a loop is made back to PROCLP in order to set another APU running (note HL automatically points to the next A array element because of the OTIR instruction). The remaining VPLIB routines are variations on this theme and a complete list of the subroutines available and their purpose is given in Table VII.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch013

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch013

Table VII

Contents of the VPLIB Vector Processing Library

VINIT(N)	INITIALIZE MVP-9500
VPPUT(A,C,N)	PUT DATA INTO MVP-9500
VPGET(A,C,N)	GET DATA FROM MVP-9500
VCLR(A,N)	VECTOR CLEAR
VMOV(A,I,C,K,N)	VECTOR MOVE
VSWAP(A,I,C,K,N)	VECTOR SWAP
VFILL(A,C,N)	VECTOR FILL
VRAMP(A,B,C,N)	VECTOR RAMP
VNEG(A,C,N)	VECTOR NEGATE
VADD(A,B,C,N)	VECTOR ADD
VSUB(A,B,C,N)	VECTOR SUBTRACT
VMUL(A,B,C,N)	VECTOR MULTIPLY
VDIV(A,B,C,N)	VECTOR DIVIDE
VSADD(A,B,C,N)	VECTOR SCALAR ADD <V+S>
VSSUB(A,B,C,N)	VECTOR SCALAR SUBTRACT <V-S>
VSMUL(A,B,C,N)	VECTOR SCALAR MULTIPLY <V*S>
VSDIV(A,B,C,N)	VECTOR SCALAR DIVIDE <V/S>
VSQ(A,C,N)	VECTOR SQUARE
VSSQ(A,C,N)	VECTOR SIGNED SQUARE
VABS(A,C,N)	VECTOR ABSOLUTE VALUE
VSQRT(A,C,N)	VECTOR SQUARE ROOT
VLOG(A,C,N)	VECTOR LOG(10)
VLN(A,C,N)	VECTOR NATURAL LOGARITHM
VEXP(A,C,N)	VECTOR EXPONENTIAL
VSIN(A,C,N)	VECTOR SINE
VCOS(A,C,N)	VECTOR COSINE

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch013

Table VII (contd.)

Contents of the VPLIB Vector Processing Library

VTAN(A,C,N)	VECTOR TANGENT
VASIN(A,C,N)	VECTOR ARC SINE
VACOS(A,C,N)	VECTOR ARC COSINE
VATAN(A,C,N)	VECTOR ARC TANGENT
VAA(A,B,C,D,N)	VECTOR ADD AND ADD <A+B+C>
VAM(A,B,C,D,N)	VECTOR ADD AND MULTIPLY <(A+B)*C>
VADIV(A,B,C,D,N)	VECTOR ADD AND DIVIDE <(A+B)/C>
VSBSB(A,B,C,D,N)	VECTOR SUBTRACT AND SUBTRACT <(A-B)-C>
VSBM(A,B,C,D,N)	VECTOR SUBTRACT AND MULTIPLY <(A-B)*C>
VSBDIV(A,B,C,D,N)	VECTOR SUBTRACT AND DIVIDE <(A-B)/C>
VMA(A,B,C,D,N)	VECTOR MULTIPLY AND ADD <(A*B)+C>
VMSB(A,B,C,D,N)	VECTOR MULTIPLY AND SUBTRACT <(A*B)-C>
VMM(A,B,C,D,N)	VECTOR MULTIPLY AND MULTIPLY <A*B*C>
VMDIV(A,B,C,D,N)	VECTOR MULTIPLY AND DIVIDE <(A*B)/C>
VDIVA(A,B,C,D,N)	VECTOR DIVIDE AND ADD <(A/B)+C>
VDIVSB(A,B,C,D,N)	VECTOR DIVIDE AND SUBTRACT <(A/B)-C>
VDIVDV(A,B,C,D,N)	VECTOR DIVIDE AND DIVIDE <(A/B)/C>
VPWR(A,B,C,N)	VECTOR POWER <A**B>
VATAN2(A,B,C,N)	VECTOR ARC TANGENT OF A/B
VRAND(A,C,N)	VECTOR RANDOM NUMBERS
VMSA(A,B,C,D,N)	VECTOR MULTIPLY AND SCALAR ADD
VSMA(A,B,C,D,N)	VECTOR SCALAR MULTIPLY AND ADD
VSMSB(A,B,C,D,N)	VECTOR SCALAR MULTIPLY AND SUBTRACT
VSBSQ(A,B,C,N)	VECTOR SUBTRACT AND SQUARE <(A-B)**2>
VSMSA(A,B,C,D,N)	VECTOR SCALAR MULTIPLY AND SCALAR ADD
VMMA(A,B,C,D,E,N)	VECTOR MUL,MUL AND ADD <(A*B)+(C*D)>
VMMSB(A,B,C,D,E,N)	VECTOR MUL,MUL AND SUB <(A*B)-(C*D)>
VAAM(A,B,C,D,E,N)	VECTOR ADD,ADD AND MUL <(A+B)*(C+D)>
VSBSBM(A,B,C,D,E,N)	VECTOR SUB,SUB AND MUL <(A-B)*(C-D)>

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch013

VFLT(I,C,N)	VECTOR FLOAT
VFIX(A,I,N)	VECTOR FIX
SVE(A,C,N)	SUM OF VECTOR ELEMENTS
MEANV(A,C,N)	MEAN VALUE OF VECTOR ELEMENTS
MAXV(A,I,C,K,N)	MAXIMUM ELEMENT IN VECTOR
MINV(A,I,C,K,N)	MINIMUM ELEMENT IN VECTOR
VMAX(A,B,C,N)	VECTOR MAXIMUM
VMIN(A,B,C,N)	VECTOR MINIMUM
VCLIP(A,B,C,D,N)	VECTOR CLIP
VICLIP(A,B,C,D,N)	VECTOR INVERTED CLIP
LVGT(A,B,C,N)	LOGICAL VECTOR GREATER THAN
LVGE(A,B,C,N)	LOGICAL VECTOR GREATER THAN OR EQUAL
LVEQ(A,B,C,N)	LOGICAL VECTOR EQUAL
LVNE(A,B,C,N)	LOGICAL VECTOR NOT EQUAL
LVNOT(A,B,C,N)	LOGICAL VECTOR NOT
VLMERG(A,B,C,D,N)	VECTOR LOGICAL MERGE
VINDEX(A,I,C,N)	VECTOR INDEX
VSUBI(A,I,C,K,E,N)	VECTOR SUBTRACT INDEXED
VSUSQI(A,I,C,K,E,N)	VECTOR SUBTRACT AND SQUARE INDEXED
MTRANS(A,MC,NC)	MATRIX TRANSPOSE
MMUL(A,B,C,MC,NC,NA)	MATRIX MULTIPLY
MATINV(A,C,N)	MATRIX INVERT <GAUSS-JORDAN>

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch013

Performance of VPLIB Routines

In order to gauge the results of the design exercise, both in terms of system performance and cost-effectiveness it is necessary to benchmark the MVP-9500 against other machines for which the benchmark figures are available. Fortunately Floating Point systems have published details of a benchmark used to compare the AP120B with mini- and mainframe computer systems (6).

The formula for the benchmark is shown in Table VIII, the FORTRAN code for the MVP-9500 in Table IX and the benchmark timings for an eight Am9511A (/8) version of the vector processor in Table X. The performance of the MVP-9500/Vector MZ combination comfortably exceeds that of mid-range PDP-11's with (FIS) and without (NHD) floating point hardware and is roughly equivalent to the powerful PRIME-400 and an order of magnitude down on the AP120B and 370/195. Bearing in mind that an MVP-9500 system can support up to 128 Am9511's and that the increase in performance is only slightly less than linear in the number of APU's it is obvious that this size of system would be comparable in performance to the AP120B.

The estimated full commercial cost of an MVP-9500/8 system based on a Vector MZ with 5M bytes of disk space, printer and terminal is around £10,400, compared with around £25,000 for a PDP11/34 with similar peripherals and a floating point processor. This makes the vector processor option around eight times more cost effective than the PDP-11; PROVIDED THAT THE CALCULATION UNDER CONSIDERATION CAN MAKE EFFICIENT USE OF A VECTOR PROCESSOR. A similar comparison of the MVP-9500/128 and AP120B puts the former ahead by a factor of two in cost effectiveness.

The power of the MVP-9500 would be to no avail if the reliability of the device were not commensurate with its performance. Exhaustive reliability tests have not been carried out but the following observations are indicative of a certain robustness. An MVP-9500/Vector MZ has been in daily use in the author's laboratory for two years without a single fault in the MVP-9500 (and indeed the only fault in the microcomputer involved replacement of one voltage regulator chip in the I/O card). Furthermore the system will run calculations which take days to complete at both normal and elevated (35°C) temperatures, the latter "test" taking place during a weekend air conditioning failure in the computer room! (fast computers make good heaters!).

Some notes on Vectorization

The price to be paid for the maximum performance from any AFPP, VP or AP is some attention to the order in which the individual operations of a complete calculation are carried out. This usually involves arranging the calculation as a sequence of operations involving vectors (single dimensional arrays) of

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch013

Table VIII

Formula for the FPS Benchmark

$$\sum_{K=1}^{N} \frac{E^{TAN(SIN(COS(K)))}}{K+K\star LOG(K)}$$

Table IX

Fortran/VPLIB Code for the FPS Benchmark

```
        DIMENSION R(2),ARAK(2500),ARAW(2500)
1       WRITE(5,100)
100     FORMAT(' INPUT NUMBER OF TERMS IN SERIES   ')
        READ(5,101) N
101     FORMAT(I5)
        IF(N.EQ.0) GO TO 2
        WRITE(5,104)
104     FORMAT(' INPUT NUMBER OF REPETIONS   ')
        READ(5,101) ICNT
        R(1)=1.
        R(2)=1.
        CALL VINIT(5)
        CALL VPPUT(R,R,2)
        CALL VRAMP(R(1),R(2),ARAK,N)
        PAUSE START
        DO 300 J=1,ICNT
        CALL VMOV(ARAK,1,ARAW,1,N)
        CALL VLN(ARAW,ARAW,N)
        CALL VMUL(ARAK,ARAW,ARAW,N)
        CALL VADD(ARAK,ARAW,ARAW,N)
        CALL VCOS(ARAK,ARAK,N)
        CALL VSIN(ARAK,ARAK,N)
        CALL VTAN(ARAK,ARAK,N)
        CALL VEXP(ARAK,ARAK,N)
        CALL VDIV(ARAK,ARAW,ARAK,N)
        CALL SVE(ARAK,ANS,N)
        CALL VPGET(ANS,ANS,1)
300     CONTINUE
        WRITE(5,102) ANS
102     FORMAT(1X,E20.10)
        PAUSE STOP
        GO TO 1
2       STOP
        END
```

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch013

Table X

FPS Benchmark Timings on Various Mini- and Mainframe Computers

IBM 370/195	0.23 sec
AP-120B	0.40 sec
PRIME 400	6.70 sec
MVP-9500/8	7.00 sec
PDP-11/40(FIS)	27.0 sec
PDP-11/34(NHD)	105.00 sec

operands with preferably a unit address difference between operands in the same vector.

Take the case of matrix multiplication, this would usually be implemented in FORTRAN as a direct translation of the expression in Table XI which is the usual "row by column and add" sequence of operations. The evaluation of a single C_{jk} does not satisfy the criteriof given above because although $B_{\ell k}$ is a vector with unit address increment from component to component $A_{j\ell}$ is not and its address increment is N. This of course is a function of the FORTRAN compiler and can be circumvented by storing A in an "unnatural" order (i.e. by rows, giving the transpose of A in the usual method of storage by columns) but this is not usually worthwhile because of the potential for confusion. So, if we stick to the usual conventions every access of an element of A will involve computation of its address rather than a simple increment from the address of the previously used element. Is there any solution to this problem? Yes, instead of using the "inner product" algorithm previously described we use the so called "outer product" illustrated in Figure 8. In this instance a whole column of C is calculated as the summation of a series of vector by scalar products. The address structure in this case is beautiful as the scalars are components of a vector with unit address increment and run through the entire matrix A, component to component and vector to vector, involving unit address increments only. F80/MVP-9500 code for the algorithm of Figure 8 is shown in Table XII, although the VPLIB version is coded in assembler to take maximum advantage of the address structure, ideally suited to the MVP-9500, discussed above.

In most intstances however merely being able to arrange the calculation as a series of vector operations, without worrying over the "unit address increment" requirement, makes extremely good if not maximal use of AFPP, VP or AP. As an illustration of this point Table XIII shows "normal" FORTRAN code for a pivotal condensation matrix inverter (the author is unfortunately by now anonymous) and Table XIV shows the vectorized version for the MVP-9500 at about two thirds completion. The VPLIB version is completely (as far as the author can manage at least!) vectorized and written in assembler. Most of the vectorization is fairly obvious and only the reduction loops contain any obscurity. In order to maintain peak vector efficiency the MVP-9500 reduction loop does a little more work than is strictly necessary; an alternative would ruin the vector flow. It is left as an exercise to the determined reader to unravel the full correspondence between Tables XIII and XIV.

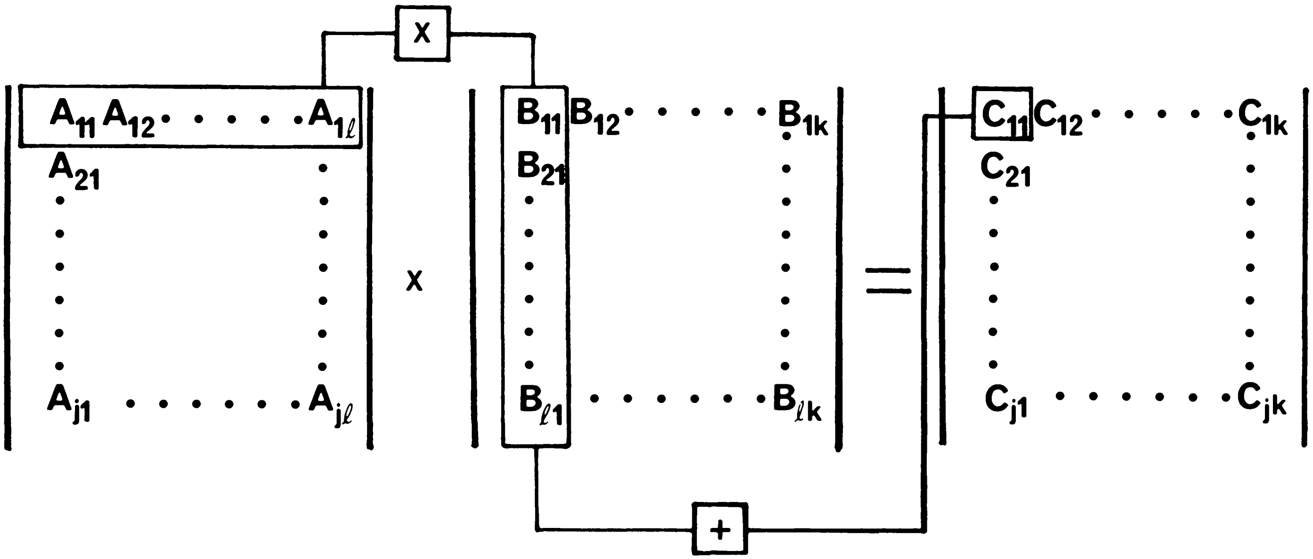
It has been the author's experience that most chemical computations, structural or otherwise, which involve a significant amount of floating point arithmetic are reducible to vector format. The exercise is usually worthwhile as in most cases the chemist is using the same program over and over again with different data sets.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch013

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch013

Table XI

The "Inner Product" Matrix Multiplication Algorithm (C=AxB)



$$C_{jk} = \sum_{l=1}^{N} A_{jl} B_{lk}$$

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch013

$$\begin{pmatrix} C_{1K} \\ \cdot \\ \cdot \\ \cdot \\ C_{NK} \end{pmatrix} = B_{1k} \cdot \begin{pmatrix} A_{11} \\ \cdot \\ \cdot \\ \cdot \\ A_{N1} \end{pmatrix} + B_{2k} \begin{pmatrix} A_{12} \\ \cdot \\ \cdot \\ \cdot \\ A_{N2} \end{pmatrix} + \ldots + B_{Nk} \begin{pmatrix} A_{1N} \\ \cdot \\ \cdot \\ \cdot \\ A_{NN} \end{pmatrix}$$

Figure 8. Pictorial representation of the "outer product" matrix multiplication algorithm.

Table XII

Fortran/VPLIB Code for Matrix*Matrix "Outer Product"

```
;ROUTINE TO MULTIPLY TWO MATRICES
;FORTRAN CALL IS CALL MMUL(A,B,C,MC,NC,NA) WHERE A & B ARE THE SOURCE
;MATRICES, C IS THE DESTINATION MATRIX, MC IS THE NUMBER OF ROWS IN A
;AND C, NC IS THE NUMBER OF COLUMNS IN B AND C, AND NA IS THE NUMBER
;OF COLUMNS IN A AND ROWS IN B. FORTRAN EQUIVALENT OF THIS ROUTINE IS
;
        SUBROUTINE MMUL(A,B,C,MC,NC,NA)
        DIMENSION A(MC,NA),B(NA,NC),C(MC,NC)
        DO 9000 K=1,NC
        CALL VSMUL(A(1,1),B(1,K),C(1,K),MC)
        DO 9010 L=2,NA
        CALL VSMA(A(1,L),B(L,K),C(1,K),C(1,K),MC)
9010    CONTINUE
9000    CONTINUE
        RETURN
        END
```

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch013

Table XIII

Scalar Fortran Coding of Gauss-Jordan Matrix Inverter

```
        SUBROUTINE MATINV(A,DET,N)
        DIMENSION A(20,20),INDEX(20,2),IPIVOT(20)
        DET=1.0
        DO 20 J=1,N
20      IPIVOT(J)=0
        DO 550 I=1,N
C       SEARCH FOR PIVOT ELEMENT
        AMAX=0.0
        DO 105 J=1,N
        IF(IPIVOT(J).EQ.1) GO TO 105
        DO 100 K=1,N
        IF(IPIVOT(K)-1) 80,100,740
80      IF(ABS(AMAX).GE.ABS(A(J,K))) GO TO 100
        IROW=J
        ICOLUM=K
        AMAX=A(J,K)
100     CONTINUE
105     CONTINUE
        IF(AMAX.EQ.0.0) GO TO 800
110     IPIVOT(ICOLUM)=IPIVOT(ICOLUM)+1
C       INTERCHANGE ROWS TO PUT PIVOT ELEMENT ON DIAGONAL
        IF(IROW.EQ.ICOLUM) GO TO 260
        DET=-DET
        DO 200 L=1,N
        SWAP=A(IROW,L)
        A(IROW,L)=A(ICOLUM,L)
200     A(ICOLUM,L)=SWAP
```

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch013

```
260     INDEX(I,1)=IROW
        INDEX(I,2)=ICOLUM
        PIVOT=A(ICOLUM,ICOLUM)
        DET=DET*PIVOT
C       DIVIDE PIVOT ROW BY PIVOT ELEMENT
        A(ICOLUM,ICOLUM)=1.0
        DO 350 L=1,N
350     A(ICOLUM,L)=A(ICOLUM,L)/PIVOT
C       REDUCE NON PIVOT ROWS
        DO 550 L1=1,N
        IF(L1.EQ.ICOLUM) GO TO 550
        T=A(L1,ICOLUM)
        A(L1,ICOLUM)=0.0
        DO 450 L=1,N
450     A(L1,L)=A(L1,L)-A(ICOLUM,L)*T
550     CONTINUE
C       INTERCHANGE COLUMNS
        DO 710 I=1,N
        L=N+1-I
        IF(INDEX(L,1).EQ.INDEX(L,2)) GO TO 710
        JROW=INDEX(L,1)
        JCOLUM=INDEX(L,2)
        DO 705 K=1,N
        SWAP=A(K,JROW)
        A(K,JROW)=A(K,JCOLUM)
        A(K,JCOLUM)=SWAP
705     CONTINUE
710     CONTINUE
740     RETURN
800     DET=0.0
        RETURN
        END
```

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch013

Table XIV

Fortran/VPLIB Code for Gauss-Jordan Matrix Inverter

```
      SUBROUTINE MATINV(A,DET,N)
      DIMENSION A(20,20),INDEX(20,2),IPIVOT(20)
      DET=1.0
      DO 20 J=1,N
20    IPIVOT(J)=0
      DO 550 I=1,N
      AMAX=0.0
      DO 105 J=1,N
      IF(IPIVOT(J).EQ.1) GO TO 105
      DO 100 K=1,N
      IF(IPIVOT(K)-1) 80,100,740
80    IF(ABS(AMAX).GE.ABS(A(J,K))) GO TO 100
      IROW=J
      ICOLUM=K
      AMAX=A(J,K)
100   CONTINUE
105   CONTINUE
      IF(AMAX.EQ.0.0) GO TO 800
110   IPIVOT(ICOLUM)=IPIVOT(ICOLUM)+1
      IF(IROW.EQ.ICOLUM) GO TO 260
      DET=-DET
      CALL VSWAP(A(IROW,1),20,A(ICOLUM,1),20,N)
260   INDEX(I,1)=IROW
      INDEX(I,2)=ICOLUM
      PIVOT=A(ICOLUM,ICOLUM)
      DET=DET*PIVOT
      A(ICOLUM,ICOLUM)=1.0
      DO 350 L=1,N
```

In Supercomputers in Chemistry; Lykos, P., el al.;
ACS Symposium Series; American Chemical Society: Washington, DC, 1981.

```
350     A(ICOLUM,L)=A(ICOLUM,L)/PIVOT
        NSQ=400
        CALL VPPUT(A,A,NSQ)
        DO 511 L=1,N
        IF(L.EQ.ICOLUM) GO TO 511
C
C       SNEG IS SCALAR NEGATE(AMD). ARG2= -ARG1
C
        CALL SNEG(A(ICOLUM,L),T)
        CALL VSMA(A(1,ICOLUM),T,A(1,L),A(1,L),N)
        CALL SNEG(T,A(ICOLUM,L))
511     CONTINUE
        CALL SNEG(A(ICOLUM,ICOLUM),T)
        CALL VSMUL(A(1,ICOLUM),T,A(1,ICOLUM),N)
        CALL SNEG(T,A(ICOLUM,ICOLUM))
        CALL VPGET(A,A,NSQ)
550     CONTINUE
        DO 710 I=1,N
        L=N+1-I
        IF(INDEX(L,1).EQ.INDEX(L,2)) GO TO 710
        JROW=INDEX(L,1)
        JCOLUM=INDEX(L,2)
        CALL VSWAP(A(1,JROW),1,A(1,JCOLUM),1,N)
710     CONTINUE
740     RETURN
800     DET=0.0
        RETURN
        END
```

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch013

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch013

VPLIB Routines for Structural Chemistry

It was mentioned earlier that a number of special purpose routines, which do not appear in the VPLIB index, have been developed for use in structural chemistry. The most frequent requirements encountered in this area are those concerned with molecular geometry and, more specifically, with the calculation of interatomic distances, angles and torsion angles. These geometric quantities are best evaluated by vector algebra and this will always involve the calculation of vector components, lengths, direction cosines, vector cross products and vector dot products. Attention should therefore be directed at the best possible way of implementing the calculations described in the latter list on the MVP-9500.

The obvious place to start is with the evaluation of the components Δx, Δy and Δz (= x_i-x_j; y_i-y_j; z_i-z_j) of the interatomic vectors. In a given calculation there will be a set of (usually) orthogonal cartesian coordinates, x_i, y_i, z_i and the components will all be calculated from this single set of coordinates, but with different pairwise combinations of the indices i and j ($i \neq j$). Some form of indexing into the array of coordinates would seem to offer an efficient method of calculating these vector components. The routine VSUBI (A, I, B, J, C, N) performs the desired calculation; I and J are singly dimensional integer arrays, the contents of which point to elements of the A and B arrays respectively. For the present purpose A and B would be identical and would contain the atomic coordinates. The integer variable N determines the number of indexed subtractions performed. The overall operation performed would then be C = A(I(1)) - B(J(1)), A(I(2)) - B(J(2)),•••• A(I(N))-B(J(N)). The contents of the I and J arrays would obviously need to be known before the vector subtract indexed operation but this would almost certainly be the case in structural chemical calculations of any size.

The other fundamental requirement for the calculation of the geometric quantities mentioned above is a routine to facilitate the derivation of interatomic distances. This may be achieved in the main by a small extension of the VSUBI routine. Consider the state of the MVP-9500 when all of the APU's present have executed the subtract operation but have not been unloaded. The Am9511A internal stack can hold four floating point numbers; in the case under consideration the top of stack holds say, Δx and the other three slots are empty. The Am9511A allows us to push the stack and duplicate Δx at the new top of stack, so that the stack now contains Δx, Δx and two empty slots. An FMUL operation to the Am9511A will result in the stack containing Δx^2 and three empty slots. If we note that an interatomic distance $\ell_{ij} = (\Delta x^2 + \Delta y^2 + \Delta z^2)^{\frac{1}{2}}$ then it can be seen that the extension to VSUBI takes us one step further in evaluating ℓ_{ij} without the need to load the Am9511A with more data. VSUSQI (vector subtract and square indexed) has the same argument list with the same significance as for VSUBI but the destination array C contains elements which are the squares of those produced by the latter routine. Why bother with a separate VSUSQI routine when the same effect can be achieved with VSUBI followed by VMUL? Well, the MVP-9500 would need to be unloaded after the VSUBI and reloaded with the same numbers before the VMUL operation, which is obviously a waste of time. A general rule when programming the MVP-9500, and most other arithmetic processors, is to minimize the transfer of data to and from the arithmetic elements. This is the reason for routines like VMMA which performs the operations E = (A*B) + (C*D) where A to E are n element vectors; the intermediate results A*B and C*D can be stored on the Am9511A stack and added together when the products are complete.

Table XV illustrates the use of VSUSQI in a subroutine to calculate bond lengths where XO are the orthogonal coordinates and INDXI; INDXJ contain entries of the form $i_1, i_1, i_1, i_2, i_2, i_2, \ldots\ldots;$ $j_1, j_1, j_1, j_2, j_2, j_2, \ldots\ldots.$ VCOMPS is an array for holding ΔXO^2_{ij} in triples corresponding to $\Delta x^2 + \Delta y^2 + \Delta z^2$ and places the result in the BL array. Taking the square root (VSQRT) of the elements of BL gives the bond lengths which overwrite the original quantities in the BL array.

The astute reader will have noticed that it is possible to perform an entire bond length calculation on a four deep stack without the removal of intermediate results. However in order to take full advantage of this fact we require an Am9511A which has its own individual control processor or microsequencer. This is a projected improvement to the MVP-9500.

Interatomic distances, angles and torsion angles may be efficiently calculated with the routines discussed above plus the remainder of VPLIB. A number of other special purpose routines have been developed and are under development but these will not be discussed here.

Vector Processing and Molecular Mechanics Calculations

The primary reason for undertaking this whole exercise was to evaluate vector processors for use in molecular mechanics calculations and as an adjunct to chemical computer graphics systems.

Picture transformations (e.g. rotation, translation, scaling, perspective etc) in interactive computer graphics lend themselves naturally to representation in matrix notation, and implementation of the various algorithms on a vector processor is obviously straightforward and very worthwhile (particularly if moving pictures are required.). For this reason graphical applications of the MVP-9500 will not be discussed here and the interested reader is referred to one of the standard texts in this area (7).

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch013

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch013

Table XV

Calculation of Bond Lengths using VSUSQI

```
      SUBROUTINE BLEN(NBOND)
      COMMON/BOND/XO(3,60),INDXI(180),INDXJ(180),VCOMPS(180),BL(60)
C *** XO CONTAINS ORTHOGONAL COORDS IN ANGSTROMS,VCOMPS CONTAINS THE **
C *** VECTOR COMPONENTS OF EACH BOND,AND BL CONTAINS THE BOND LENGTH **
C *** INDXI & INDXJ POINT TO X,Y,Z COORDINATES OF ATOMS I & ATOMS J  **
C
      CALL VSUSQI(XO,INDXI,XO,INDXJ,VCOMPS,NBOND*3)
      IJ=1
      DO 10 I=1,NBOND
      CALL SVE(VCOMPS(IJ),BL(I),3)
      IJ=IJ+3
10    CONTINUE
      CALL VSQRT(BL,BL,NBOND)
      RETURN
      END
```

Molecular mechanics calculations (8) certainly involve a great deal of floating point arithmetic but a casual study of the problem does not reveal the suitability, or otherwise, of the calculations for implementation on a vector processor. The author has implemented his molecular mechanics program (9) on the MVP-9500/Vector MZ in a piecemeal fashion. That is, at the outset the obviously vectorizable parts of the program were converted to use VPLIB first and the less obvious or more difficult conversions tackled last. Several points immediately emerged, such as the fact that: some parts of the calculation are performed more effectively as scalar rather than vector operations; the price of speed with the MVP-9500 is a larger memory requirement than that for the all-scalar program; efficient use of the MVP-9500 requires substantial reconstruction of the serial, scalar program flow and finally (and paradoxically) it is sometimes worth doing a little more computation than is absolutely necessary in order to remove the requirement for conditional statements. These facts are discussed at length in the excellent Infotech State of the Art Report on Supercomputers (10) but will be mentioned here to the extent that they impact on the author's molecular mechanics program.

The program starts by reading in orthogonal coordinates for a molecule and calculating all of the bonded atom pairs from these and a table of bond radii. This information is then used to construct a bonding matrix which is operated upon to produce groups of three numerical identifiers (the atom numbers) for valency angles and groups of four identifiers for torsion angles. Although these operations can be expressed in MVP-9500 code or VPLIB calls the result is not satisfactory and best left in standard scalar FORTRAN.

However, the rest of the program which embraces 99% of the work can be efficiently vectorized. This remainder is exclusively concerned with the evaluation of the molecular potential energy consequent upon changes in the orthogonal coordinates, and with matrix inversion and matrix by vector multiplication. The former series of operations takes place during the calculation of numerical first and second derivatives and the latter during the calculation of new orthogonal coordinates for the particular iteration.

Evaluation of the potential energy involves first: calculating the bond lengths, angles, non-bonded distances and torsion angles from the previously constructed bonding matrix, valency angle identifier table and torsion angle identifier table and second; the use of these geometric quantities, together with a force field to calculate energies for individual interactions which are then summed to give the total energy. The calculation of bond lengths etc. by means of VPLIB calls has already been discussed. In the case of the energies, the same four algebraic expressions are used over and over again with

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch013

different force constants and bond lengths etc. so that this series of computations is also straightforwardly expressed in terms of VPLIB calls.

The remaining calculation of inverse matrices and matrix by vector products is handled by calls to MATINV and MMUL (a vector can be considered as an Nx1 matrix).

The construction of an MVP-9500 version of the molecular mechanics program which ran faster than the author's PDP-11/40 (FIS) version was an undemanding if tedious process but, as with construction of VPLIB routines, arriving at an efficient rather than brute force coding will take several iterations.

Conclusions

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch013

The author has presented details of a cost effective vector processor for use with S-100 microcomputers and produced a library of FORTRAN callable subroutines for general purpose floating point computations. Brief details of the construction of a molecular mechanics program using the vector processor have been given.

It is appropriate at this point to include a brief discussion of the possibilities for future, enhanced versions of the MVP-9500. The MVP-9500 was designed with flexibility and ease of enhancement very much in mind and some improvements have already been implemented.

Perhaps the most straightforward enhancement involves replacement of the Z80A control processor and the Am9511A-1DC arithmetic processors with the faster Z80B (6MHz as opposed to 4MHz for the Z80A) and Am9511A-2DC (4MHz as opposed to 3MHz). Using faster Am9511A's results in an increase in performance roughly proportionate to the clock speeds for VPLIB operations (i.e. run times are reduced to ≈0.8 of the original), but using a 6MHz Z80B results in an extra gain because not only is there a reduction in time required to move data around the system by virtue of the higher clock rate, but also the fact that this enhances parallelism amongst the Am9511A's because load times are a smaller proportion of the total multi-Am9511A operation than previously. Using a 6.55MHz Z80B reduces the time for the benchmark of Table X from 7.00 sec to 4.20 sec.

Another straightforward possibility is to increase the number of Am9511A's in the system at eight per card. Diminishing returns sets in fairly quickly and the optimum number of Am9511A's for different systems is as follows: 8 for a 4MHz Z80A, 16 for a 6MHz Z80B, 64 for a 6MHz Z8000 and 128 for the projected 10MHz Z8009. This leads on to replacing the 8 bit Z80 with a more powerful 16 bit Z8000 as the central processor. It is here that the advantage of choosing the S-100 bus becomes obvious in that the only hardware change required for this enhancement is replacement of the Z80 CPU card, The software needs to be changed of course and VPLIB can either be translated to Z8000 code with a commercially available program, or alternatively rewritten to take maximum advantage of the Z8000 architecture. The author has tested this hardware configuration. 16 bit wide data busses also offer the possibility of loading two Am9511A's in parallel with one Z8000 instruction.

The main limitation on the MVP-9500 system as described lies in its 64k byte address space. By the time the operating system and the requisite parts of VPLIB and the fortran library are loaded there are only ≈20k bytes available for data arrays on a 48k Vector MZ. A 16 bit control processor such as the Z8000 would help in this respect as it has an 8M byte data address space. However, the Z8000 program counter is only 16 bits wide and access to memory beyond 64k bytes is by the addition of the contents of a segment register to the program counter to give a 24 bit address. This can be simulated with a Z80 by means of bank switching in which an output instruction (analogous to loading the segment register in the Z8000) is used to select (usually) one of eight 64k byte banks of memory. This is done in the author's system, as shown in Figure 7. In this case a modified VPLIB dedicates one memory bank to each of the A, B, C, D and E vectors referenced by the VPLIB subroutine calls. Data are loaded into the banks either by way of a reserved area of permanently enabled memory or by I/O transfers in which case the memory, vector processor and control processor reside in a "vector coprocessor" attached to the Vector MZ by a high speed parallel link.

Finally, most of the matrix routines heavily used by the author such as MMUL and MATINV rely heavily on the VSMA (vector scalar multiply and add routine) and a means of improving the vector by scalar multiply would be most welcome. Fortunately this is easy to achieve with little extra hardware, and minimal software changes. The chip select lines of the Am9511A's are conditioned by a latched I/O port so that they are either logically separate and the Am9511A's individually addressable (as previously described) or logically connected one with the other so that addressing one Am9511A enables all of the other Am9511A's on the board too. A scalar load to a MVP-9500/8 is reduced from eight separate loads of the same scalar to each Am9511A, to one parallel load of the scalar to all eight Am9511A's simultaneously.

In aggregate the enhancements discussed above could lead to versions of the MVP-9500 up to 30 times more powerful than the original at very little extra (direct) cost.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch013

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch013

Acknowledgements

The author is indebted to the Science Research Council for financial support of this project and to Video Vector Dynamics Ltd. for the provision of printed circuit boards and technical support.

Literature Cited

1. D.H. Kuck, D.H. Lawrie and A.H. Sameh (Eds) "High Speed Computer and Algorithm Organization"; Academic Press: New York, 1977.
2. Gupta, B.K. *Computer Design* July 1980, **19**, 85.
3. "North Star Computers, Product Catalog"; North Star Computers Inc.: Berkeley (U.S.), 1980.
4. Palmer, J., *SIGARCH Newsletter*, May 1980, **8**,174.
5. Elmquist, K.A., Fullmer, H., Gustavson, D.B. and Morrow G. *Computer* July 1979, **12**, 28.
6. Markham, S.; "Floating Point Systems, The Array Processor Company"; F.P.S. Inc.: Bracknell (U.K.), 1979.
7. Newman, W.M.; Sproull, R.F. "Principles of Interactive Computer Graphics"; McGraw-Hill: New York, 1979.
8. White, D.N.J.; "Molecular Structure by Diffraction Methods Vol. 6"; The Chemical Society: London, 1978; p.38.
9. White, D.N.J. *Computers & Chemistry* 1977, **1**, 225.
10. "Infotech State of the Art Report, Supercomputers"; Infotech International: Maidenhead (U.K.), 1979.
11. "Algorithmic Details for the Am9511 Arithmetic Processing Unit", Advanced Micro Devices.

RECEIVED June 18, 1981.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch014

14

Programming Language for Supercomputers

RICHARD L. LOZES

University of Florida, Quantum Theory Project, Gainesville, FL 32611

Computer architectures have evolved over the years from the classic von Neumann architecture into a variety of forms. Great benefits to operating speed have accrued. The major contributions to speed have been the introductions of parallel processors and of pipelining. For many years, these innovations were transparent to the programmer. For example, to program in Fortran to run on a CDC 6600 (1), one did not take cognizance of the existence of multiple functional units, nor did one consider the I/O channels when writing Fortran applications for the IBM 360 series (2). This was because the parallel processors were hidden behind appropriate hardware or software.

Unfortunately, other machines have not been as easy to use. Any machine with virtual memory suffers performance degradation when page swaps are too frequent. Some pipelined machines like the TI-ASC (3) or the CDC Star-100 (4) have rather long setup times for their arithmetic pipes. Multiprocessor machines like the Illiac IV (5) are next to useless if the programmer pays no attention to the architecture. These features all directly impact the user; they have not been effectively hidden by software at any level. Improvement of this situation could result if compilers took on the burden of optimizing code so as to promote efficient hardware utilization.

The reluctance of manufacturers to supply compilers for anything but Fortran (or derivatives thereof) has largely blocked this course. Fortran is a reflection of the von Neumann architecture. The programmer is forced to translate an algorithm to fit the confines of von Neumann architecture, and the compiler must then translate that sequential representation backward in order to generate a parallel representation. Lamport has cogently argued (6) that such procedures invariably result in low efficiency in programming and in execution. Among the contributors to these inefficiencies are Fortran's demand for details (since it is not a high-level language) and its lack of global structure (which leads to loss of information upon translation into Fortran). Fortran implementations in this category include Star-Fortran (7)

0097-6156/81/0173-0237$05.00/0
© 1981 American Chemical Society

for the CDC Star-100, IVTRAN (8) for the Illiac IV, and CFT (9) for the Cray-1.

In another class are languages written for specific architectures, including CFD (10) for the Illiac IV, Glypnir (11), also for the Illiac IV, and SL/1 (12) for the Star. Such languages are obviously not portable due to the small number of Star's and Illiac's in the world, and thus do not represent a viable alternative. Indeed the large number of available architectures makes portability of paramount concern; a scientific language must be architecture independent if it is to be useful.

The only languages relevant to this discussion which have achieved architectural independence are Actus (13) and Bohlender's Pascal extension (14). Actus ambitiously attempts to denote the possibility of parallel processing. Its syntax forces the programmer to specify the pattern of possible parallelism. The compiler is free to utilize or ignore that information. From one point of view, Actus can be regarded as an extension of Pascal explicitly for single instruction stream multiple data stream architecture which is compatible with any existing machine.

Bohlender's Pascal extension (14), on the other hand, is a true high-level language. It does not address architecture at all; its only concern is to implement algorithms. It is limited to the domain of linear algebra, providing vectors and two-dimensional matrices over the integral, real, and complex numbers, along with the necessary primitive operations. Its modest goals allow one to do away with indices when treating vectors and matrices as such. This is highly desirable, since it results in programming at a much higher level than otherwise possible, and gives the greatest possible freedom to the compiler to produce efficient code in particular implementations. One can write, for example, "A := herm(U)*B*U", where A, U, and B are compatible complex matrices, and herm(U) returns the Hermitean adjoint of U.

In this paper I propose a high-level language, Multilin, to treat problems of numerical multilinear algebra while taking advantage of the vector and matrix processing capabilities of supercomputers. Multilin is free from architectural assumptions, thus each realization of it in the form of a compiler can produce globally optimal code. Further, Multilin builds on an existing structure, inheriting control structures, I/O facilities, and a general von Neumann framework, lending a certain ease to its implementation.

Design Philosophy

There are at least four high-level definitions required in programming: the problem, the procedure blocking, the procedure interfaces, and the data structures. The first may be left aside in this discussion. A high-level language should help the programmer concentrate on what is to be done, rather than on how it is to be done (15). On the other hand, efficiency considerations

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch014

generally dictate that the programmer give some direction to answering the question of "how". Indeed, global optimization is still best done by humans. Careful attention to algorithm selection and its concomitant data structuring and procedure blocking is probably the best approach to global optimization. "How" should be exclusively defined by these structures. The rest of the programming task can be devoted to "what", with the compiler filling in all details, including register assignment, addressing of real and virtual storage, and local code optimization.

The design of a language should provide the means of defining procedures, their interfaces, and the control of flow. These aspects have evolved satisfactorily, so that I intend to take them over from an existing language. Language design should also provide a rich variety of data structures appropriate to the problems to be solved. These must be accompanied by the definition of operations on those structures, expressible in a clear and concise notation. To unburden the programmer further, careful distinction must be maintained between data and its representation. This is important both to aid conceptualization and to permit independence of architecture. It is nearly impossible in a low-level language like Fortran wherein all objects such as linked lists, strings, and trees must be represented by arrays. Furthermore, the definitions of data structures must be explicit and inviolate. For example, arrays are at their best when homogeneous -- all elements are to be treated equally. Thus, partitioned arrays should be partitioned by name, not by indexical relations; bordered arrays should have their borders stored separately.

Bohlender's Pascal extension (14) particularly well exemplifies these considerations. Numerical linear algebra using the native Pascal matrix representation is handled cleanly and succinctly.

Multilin

Much of numerical computation in chemistry revolves about numerical multilinear algebra. By this term I denote the manipulation of arrays whose dimension may exceed two. Mindful of Richard and Ledgard's admonition (16) that language design should never be overly ambitious, I propose only data structures, operations, and syntax for programming multilinear algebra. Multilin exceeds Bohlender's Pascal extension in two ways: its provision for more than two dimensions, and its explicit declaration of data representations.

In addition to the usual array representation which stores A(i,j,k) as A'(((i-1)n+j-1)o+k), where A is dimensioned (m,n,o) and A', (mno), Multilin provides representations taking advantage of sparsity and symmetry. Patterned sparsity such as that exhibited by a band diagonal matrix can be treated through another matrix. Thus a band matrix, A, dimensioned (n,r), with k lower

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch014

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch014

codiagonals and m upper codiagonals is mapped $A(i,j) \rightarrow A'(i,j-i+k)$ for $i-k \leq j \leq i+m$ and $A(i,j) \rightarrow 0$ otherwise. Likewise a lower triangular matrix is mapped $A(i,j) \rightarrow A'(i(i-1)/2+j)$. Randomly sparse arrays are represented by linked lists (17) or by ordered lists of p-tuples (for arrays of p-1 dimensions) (18). Algorithms appropriate to such storage may be found in the literature (19).

Multilin utilizes transpositional symmetry for any pairs of indices. Thus, if A is declared symmetric, antisymmetric, Hermitean, or anti-Hermitean in any index pair, an appropriate (multiply) triangular storage and addressing scheme is invoked. For example, the specification $A(h,i,j,k)=A(i,h,j,k)=A(h,i,k,j)=A(j,k,h,i)$ maps $A(h,i,j,k) \rightarrow A'(m(m-1)/2+n)$, where $m=\max(p(p-1)/2+q, r(r-1)/2+s)$, $p=\max(h,i)$, $r=\max(j,k)$, and n, q, s are the respective minima.

Implementation. As alluded to earlier, the features of Multilin are embedded into an existing language. (Rephrasing, Multilin is defined as an extension of an existing language.) There are many reasons for this choice, among them the prior existence of control structures, data structures, input/output facilities, and a general von Neumann framework. Further, a preprocessor can be implemented to compile Multilin into the embedding language and a selection of subroutine calls which can be coded specifically for each architecture.

The choice of embedding language is not easy. There is a strong case to be made for choosing Fortran just because of its wide availability. The case against Fortran is equally strong: it is of too low a level, it does not have the primitives to make it a good target for compilation, and it has no block structure and therefore no calling tree or scope information to aid global optimization and to be used to check for global errors. Pascal likewise lacks many primitives, especially concerning input and output, and it does not naturally encompass efficiency considerations.

PL/I has its problems, both theoretical (16) and practical (limited availability), but its structure and wealth of primitives make it highly desirable. I have therefore chosen PL/I as the embedding language of Multilin. Much of Multilin's syntax is therefore inherited from PL/I. The major extensions are the addition of new attributes and keywords, and the introduction of a summation convention along with an extension of the meanings of certain PL/I operators.

Arrays in Multilin may acquire attributes specifying sparseness and symmetry. For matrices the special keywords LOWER TRIANGULAR, UPPER TRIANGULAR, and BAND(k_1,k_2) are allowed. In the last, k_1 and k_2 respectively designate the number of lower and upper codiagonals. Since these concepts do not generalize to higher dimension except awkwardly, one may question the wisdom of their inclusion. It was felt that they were too common to neglect entirely. Randomly sparse arrays may be given either of the keywords LINKED or TUPLE to denote the two representations mentioned earlier.

Transpositional symmetries are declared by appending the attribute ((dummy indices=op$_1$(transposed dummy indices)=...= op$_n$ (transposed dummy indices)), where op$_1$...op$_n$ may be either "-", "CONJG", or "-CONJG". For example, the two-particle reduced density matrix of quantum chemistry would be declared

```
DECLARE GAMMA(N,N,N,N) COMPLEX ((I,J,K,L)=-(J,I,K,L)=(I,J,L,K)
=CONJG(K,L,I,J));.
```

Notice that the compiler construes all possible combinations of symmetries from the minimal set given in the declaration, and stores only a set of nonredundant elements. Where noncontradictory, one of the sparsity attributes may be combined with a symmetry attribute.

The power of Multilin revolves about its implicit loop variables and its summation convention. These variables denote what operations are to be performed on the arrays, not how they are to be implemented for particular representations. For example, assume an array V to be declared identically to GAMMA above. The trace of the product of these arrays is denoted by "V(I,J,K,L)* GAMMA(K,L,I,J)"; there is no mention of inequalities on index ranges, or factors of 2 Re or 4 Re due to those inequalities. Such peculiarities of representation find no place in the notation.

The attribute DUMMY_INDEX (D_I) defines a scalar for use as an index. D_I's can never be assigned values and can only appear as array indices; their sole function is to imply loops. The programmer is free to distinguish D_I's typographically by means of the DEFAULT statement.

Two conventions govern the meaning of D_I's. The range convention states that a D_I occurring once in an expression or in the target of an assignment takes on all integer values in the range of the dimension whose place it holds. The case of a D_I appearing only once in an assignment and on the right hand side is excluded; the cases of a D_I appearing only once in an assignment on the left hand side and zero or more times on the right hand side are permitted and have the obvious meanings. The summation convention states that the occurrence of a D_I in an expression more than once but not in the target of the possible assignment containing that expression implies summation with respect to that D_I, the range being taken from the (compatible) dimension attributes of the arrays involved.

For example, the first column below, written in Multilin, is equivalent to the second column, written in PL/I:

```
DECLARE #I DUMMY_INDEX;
DECLARE (A,B,C)(N);      DECLARE (A,B,C) (N);
A(#I)=B(#I);             DO I=1 TO N; A(I)=B(I); END
F=B(#I)*C(#I);           F=0; DO I=1 TO N; F=F+B(I)*C(I); END;
A(#I)=0;                 DO I=1 TO N: A(I)=0; END;
```

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch014

The implicit loop and summation conventions have the effect of extending the meanings of all PL/I unary and binary arithmetic operators. Some compromise of the principles set forth earlier is necessary in that, while the elementwise applications of the operators +, -, *, unary-, REAL, IMAG, COMPLEX, CONJG, etc. to arrays are identical to the application of the associated array operators, there are no such correspondences in the cases of / and **. The programmer is forced to consider these operators only in terms of array elements. Furthermore, some operations are properly outside the realm of multilinear algebra. The alternative, excluding these few operators from extension, was felt to bring too much asymmetry to the syntax.

The multiplication operator extends in two ways depending on the indexical relationships. One extension is the ordinary array (inner) product, while the other is the direct product.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch014

Concluding Remarks

The unbalanced evolution of computer architectures and computer languages has led to this proposition of Multilin, a high-level language suitable for the programming of numerical problems in multilinear algebra. Multilin ignores data representations except at the declarative level, to allow for greatest programmer efficiency. It ignores target architecture as well, to allow for greatest compiler optimization. As a modest extension of an existing language, Multilin should be straightforward to learn and to implement.

I must remark that I view this definition of Multilin as preliminary. It will be some time before the ramifications of the propositions become clear. Furthermore, certain aspects of representation are yet to be addressed: direct access files as user defined virtual storage, sequential files in the same light, etc.. For some array operations, we know how to represent data to minimize paging (20). Certainly such results should be extended and incorporated into compilers. Even more important is to derive similar algorithms for direct access files, allowing addressing far beyond the addressing range of most computers. Also, for sequential access to an array, it should not be necessary to make that array resident in primary memory. Thus I look for an evolution of Multilin so as to lift even the great burdens of secondary memory management from the programmer.

Acknowledgements

I wish to thank the organizers of this symposium, NRCC, NSF (CHE78-28088), AFOSR (79-0032 (A)), and the Department of Chemistry, University of Florida for supporting this research and its presentation. Especial thanks are due to Yngve Öhrn and Isaiah Shavitt.

Literature Cited

1. Thornton, J. E. "Design of a Computer: The Control Data 6600"; Scott, Foreman: Glenview, Ill., 1970.
2. International Business Machines Corporation "Introduction to IBM System/360 Architecture"; IBM: White Plains, N.Y., 1967; p. 3.
3. Watson, W. J. AFIPS Conf. Proc. 1972, 41, 221-228.
4. Hintz, R. G.; Tate, D. P. Proc. 6th Ann. IEEE Comput. Soc. Int. Conf. 1972, 1-4.
5. Barnes, G. H.; Brown, R. M.; Kato, M.; Kuck, D. J.; Slotnick, D. L.; Stokes, R. Q. IEEE Trans. Comptr. 1968, C-17(8), 746-757.
6. Lamport, L. ACM Sigplan Notices 1974, 10(3), 25-33.
7. Control Data Corporation "Star Programming Manual"; CDC: Minneapolis, 1976.
8. Millstein, R. E. Comm. ACM 1973, 16(10), 622-627.
9. Russell, R. M. Comm. ACM 1978, 21(1), 63-72.
10. Stevens, K. ACM Sigplan Notices 1975, 10(3), 72-80.
11. Lawrie, D.H.; Layman, T.; Baer, D.; Randal, J. M. Comm. ACM 1975, 18(3), 157-164.
12. Basili, V. R.; Knight, J. C. ACM Sigplan Notices 1975, 10(3), 39-43.
13. Perrott, R. M. ACM Trans. Prog. Lang. Sys. 1979, 1, 177-195.
14. Bohlender, G. Computing 1980, 24, 149-160.
15. Leavenworth, B. ACM Sigplan Notices 1974, 9(4), iii.
16. Richard, F.; Ledgard, H. F. ACM Sigplan Notices 1977, 12(12), 73-82.
17. Knuth, D. E. "The Art of Computer Programming, Vol. 1"; Addison-Wesley: Reading, Mass., 1968; pp. 295-302.
18. Horowitz, E.; Sahni, S. "Fundamentals of Data Structures"; Computer Science Press: Potomac, Maryland, 1977; pp. 51-62.
19. Bunch, J. R.; Rose, D. J., Eds. "Sparse Matrix Computations"; Academic: New York, 1976.
20. Fischer, P. C.; Probert, R. L. Comm. ACM 1979, 22(7), 405-415.

RECEIVED April 29, 1981.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch014

The Advanced Flexible Processor, Array Architecture

BRUCE COLTON

Control Data Corporation, P.O. Box 1249-B, Minneapolis, MN 55440

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch015

The Advanced Flexible Processor (AFP) is a relatively powerful computer employing a highly parallel architecture. It has been designed to stand alone, hosted by a general-purpose computer, or to function within arrays of Advanced Flexible Processors. All of the features for efficient interprocessor communication and control are built into each Advanced Flexible Processor to allow efficient computation in a multiprocessing environment. The Advanced Flexible Processor was developed by an advanced computer research division of Control Data called the Information Sciences Division (ISD). The Information Sciences Division began work on the Advanced Flexible Processor in 1976. Our primary goal was to develop a programmable computing machine that would provide the computational power and speed required by many of the intense algorithmic processes associated with image processing, while providing some of the flexibility of a general-purpose machine.(1)

0097-6156/81/0173-0245$05.50/0
© 1981 American Chemical Society

Early Research and Development in Multiprocessing

In 1968 Control Data began research into the feasibility of performing some of the tasks associated with image processing functions, such as modular change detection on digital computers. CDC began this effort by testing algorithms on the super computer of that day, the CDC 6600. Based on this preliminary algorithmic study effort on the modular change detection problem, ISD began development of the 1280 Change Detection System which was a hardwired-logic implementation built specifically to perform the Change Detection Algorithm. Our experience in the designing and building of special-purpose hardwired systems for image processing applications indicated the need for a more flexible approach to the development of special-purpose systems.

In 1972 the Information Sciences Division began development of the Flexible Processor (FP), which was a programmable, special-purpose computer employing a highly parallel architecture. The Flexible Processor, like its successor the Advanced Flexible Processor, was designed to operate as an individual programmable processing element in an array of other individually programmable elements. The Flexible Processor used a global bus interconnection system between processors. Later investigations began to determine other interconnection network architectures which might prove to be more optimally suited for a Multiple Instruction, Multiple Data Stream (MIMD) type of array architecture(2,3). The products of this initial research into various interconnection schemes resulted in ISD developing a ring connected architectural approach to linking Flexible Processors in large multiprocessing arrays.

In 1976 Control Data delivered its first modular change detection system built around the Flexible Processor ring connected architecture to Wright-Patterson Air Force Base. Research indicated that a processor capable of performing at computational rates 10 times that of the Flexible Processor was in order and would be required to meet the burgeoning computational demands of the 1980's (4-7). Thus, Control Data Corporation began the development of the Advanced Flexible Processor using the latest LSI technology which was developed by CDC for use in its most advanced Cyber computers.

AFP Hardware Overview

An Advanced Flexible Processor is implemented on four large scale integrated (LSI) circuit panels. The component technology is emmitter coupled logic (ECL) chips. Each LSI panel carries a total of approximately 500 F200K ECL logic chips and 1,100 ECL 100K logic chips. The Advanced Flexible Processor employs the same freon cooling system used in CDC's Cyber 200 series computers. This technology provides an increased reliability

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch015

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch015

figure at the chip level of approximately 100 times that achievable using ECL 100K logic chips in an air-cooled environment. The rough computational capabilities provided by an array of 16 Advanced Flexible Processors would be approximately 800 billion arithmetic and logical operations per second. A far larger number of operations could be added to the total if one were to count the many operations associated with operand transfer and data management which are concurrently performed by the AFP in support of the arithmentic and logical computations.

Interconection Technology. AFP systems employ a ring connected architectural concept. The interprocessor communication between two adjacent Advanced Flexible Processors in the communications ring is approximately 800 million bits per second. A unique characteristic of the ring connected architecture employed by the Advanced Flexible Processor provides a distinct advantage in the performance capability of multiprocessor systems. Program partitioning strategies allow one to realize proportional increases in available ring system intercommunication bandwidth as processors are added to the multiprocessor array. This feature is in direct contrast to other multiprocessor architectures in which interprocessor communication is strangulated as processors are added to the system. As a result of this unique feature, an array of 16 Advanced Flexible Processors may provide overall system bandwidth for intercommunications of 26 billion bits per second.

AFP Performance Benefits. Comparisons between the performance of the Advanced Flexible Processor and other current super computers have been made on the image processing Change Detection Algorithm. The Advanced Flexible Processor has been determined to be approximately 2,000 times faster than a CDC 6600 on the Change Detection Algorithm, and to provide approximately 100 times the capability of the CDC 7600 computer. The Advanced Flexible Processor is found to perform 20 times faster than its predecessor, the Flexible Processor. In terms of cost effectiveness, the Advanced Flexible Processor appears to be at least two orders of magnitude more cost-effective than any of the current super computers on the Change Detection Algorithm, and one order of magnitude more cost-effective than its predecessor, the Flexible Processor.

Architectural Concepts

The following discussion will review some of the issues related to the choice of a multiprocessing solution for those problems for which general-purpose uniprocessors do not provide adequate solutions.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch015

Pipelined Processing. Consider a processing facility composed of a single processor to which is presented an incoming stream of data elements. Computations are to be performed upon these incoming data elements, and output results are to be provided on a real or near real time basis (Figure 1a). When the number of operations to be performed on the incoming data elements increases to the point where a single processor cannot provide output results within the required real or near real time constraints, or where an input backlog is steadily growing, then it would be required that the compute power of the single processor be augmented by the addition of processors into the system to work jointly on the common task at hand.

The common task would be partitioned among the added processors in a pipeline fashion where each processor would operate only upon a single serial stage of the entire computation, and would pass its intermediate results onto the next cooperative processing element, which would be working on the next sequential stage of the computation (Figure 1b). As each computational stage completes the processing of a data element, the next data element in sequence may be input to that stage, and stage processing initiated. The pipeline is "full" when there is a data element simultaneously being processed through each and every stage of the pipeline. Each of the N processors, corresponding to the N stages of the pipeline, would then be busy, contributing to the total processing power brought to bear on the problem.

Parallel Processing. If the incoming data rate of the proposed system were to increase to a point beyond the individual I/O capability of a single processor, then it would be required that processors be added to the computation in a parallel fashion, each performing identical operations on a parallel set of data elements (Figure 1c). In summary, one can state that when the number of instructions in a particular algorithm increases beyond the capability of a single processor to provide real time or near real time results, then additional required processors would be added in a pipeline fashion, whereas when the I/O rate of an individual processor is exceeded, then processing elements are required to be added in a parallel fashion. The Modular Change Detection system developed by the Information Sciences Division consisted of four pipelines with ten processors in each pipe.

General Multiprocessor Taxonomies

In general it is not adequate to simply provide capability for only parallel or pipeline configurations of processing elements, or for that matter some parallel-pipelined combination thereof. Algorithms are generally more complex than that, and require more complex feedback paths, such as exemplified in recursive types of algorithms.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch015

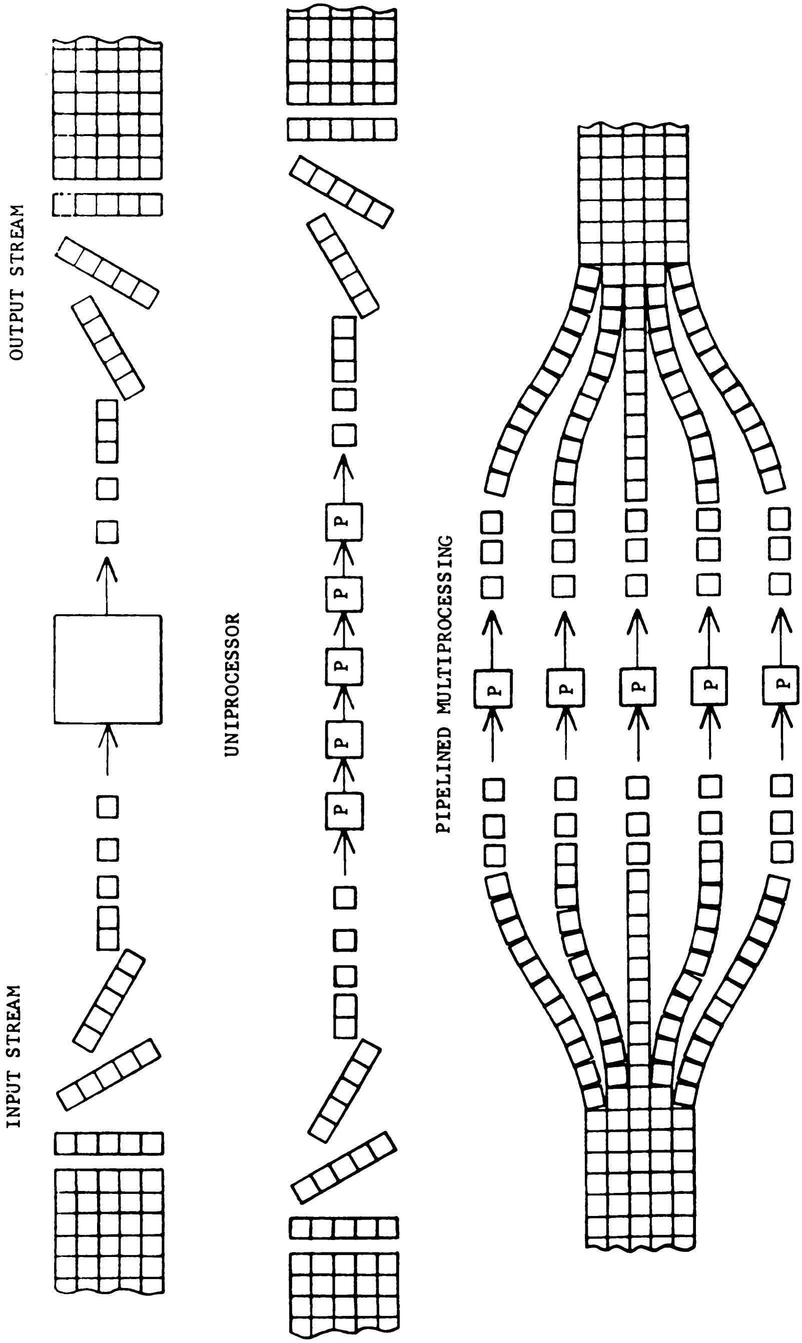


Figure 1. Pipelined and parallel processing.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch015

Four generalized types of intercommunications architecture for multiprocessing that may be considered are shown in Figure 2 and are: 1) a global bus interconnection architecture where all communication between the processors occurs on a single, common data bus; 2) a fully interconnected architectural scheme where each cooperating processing element has a unique data path to every other processing element in the array; 3) a shared memory type of processing element interconnection where all communication and data transfer occurs through a common, shared memory facility; and 4) a ring connected architectural concept which consists of a circular interconnection of processing elements, where each processing element is directly connected to its two neighboring processors in the ring. The Advanced Flexible Processor uses the latter two interconnection schemes. The Advanced Flexible Processor uses both a dual, counter-rotating ring interconnection system, as well as a common shared memory facility.

Each of the previously mentioned interconnection architectures possess characteristic strengths and weaknesses, requiring evaluation on the basis of several criteria. These architectures may be evaluated on the basis of: 1) system reliability, 2) expansion capability, and 3) cost.

System Interconnect Reliability. From the standpoint of reliability, the shared memory system in the global bus both have problems in the area of single-point failures. If a failure of the bus or the central memory occurs, the entire system is incapacitated. A ring system, when bypass hardware is employed, demonstrates very good fault tolerant characteristics.

The fully interconnected system is the best of four systems considered in the area of fault tolerance since each processor has a dedicated path to every other processor for intercommunications.

System Interconnect Expandability. From the standpoint of expansion limitations, the shared memory system has problems in that the number of ports are fixed. Expanders can be used to alleviate this problem to some degree, but physical construction problems are ultimately met. Also, the memory bandwidth of the shared memory system is fixed and is relatively slow, thus limiting the degree of practical expansion.

A global bus system has limited fanout capabilities; electrical problems are generally encountered after a relatively low number of processors are added to the system. Also, the global bus system demonstrates the lowest bandwidth capability of all of the systems, since all of the processing elements used the common shared bus. In fact, the operating bandwidth of the global bus system will never reach its theoretical maximum due to the idle time spent while processors access the bus, release the bus, and resolve bus access conflicts.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch015

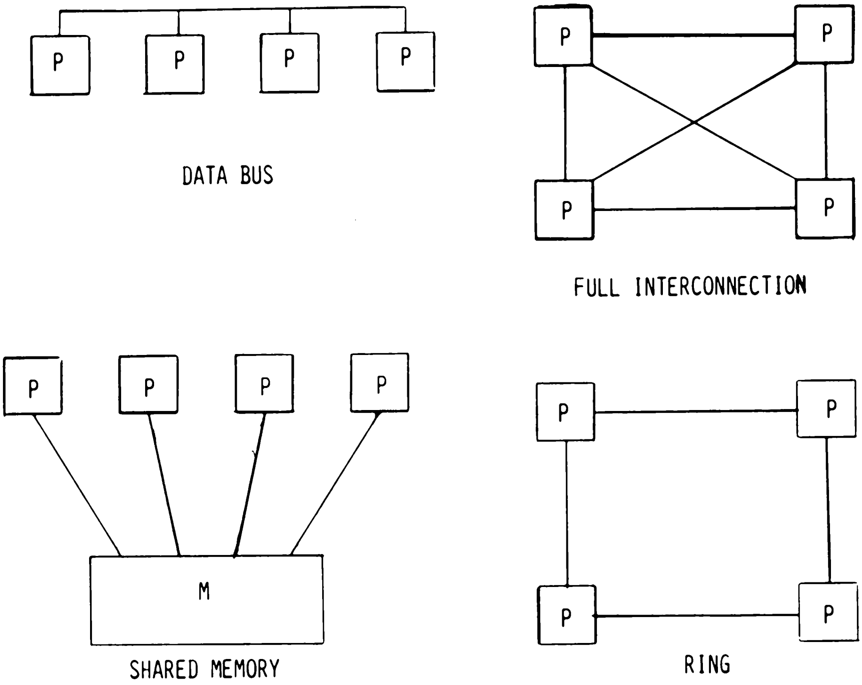


Figure 2. Interprocessor communication system.

The fully interconnected system is limited in that the number of ports are fixed with respect to the processing elements. While expanders can be used, ultimate physical problems will be encountered.

In the ring system, the bandwidth between adjacent processors is fixed; however, utilizing the special characteristics of a ring connected architecture provides system intercommunication bandwidths which tend towards the arithmetic product of this fixed interprocedure bandwidth and the number of processors in the system. Thus, a proportionate increase in supporting intercommunications bandwidth is available as processors are added to the system. Expansion within a ring connected system is, of course, virtually unlimited and has a very low cost impact.

System Interconnect Cost Performance. One may obtain a measure of the cost effectiveness of the four generalized architectures by plotting the cost to throughput ratio for each architecture as a function of the number of processors in the system (Figure 3).

The shared memory system is the most expensive of the four generalized architectures, with the global bus system coming in at close second. The fully interconnected system is about 5 times more cost-effective than a global bus approach for a 30-processor system; however, the ring system is superior to all when the process is partitioned to take advantage of the unique bandwidth characteristics that a ring connected architecture provides.

Advanced Flexible Processor Architecture

The Advanced Flexible Processor is a unique and powerful architecture providing an extremely high degree of flexibility and cost-effectiveness. It consists of 16 relatively autonomous functional units interconnected by a power 16 x 18 port, crossbar interconnect. Each of the data paths interconnected by the crossbar is 16 bits wide. Table 1 describes the functional unit breakdown of the Advanced Flexible Processor. A conceptualized functional organization of the AFP is shown in Figure 4.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch015

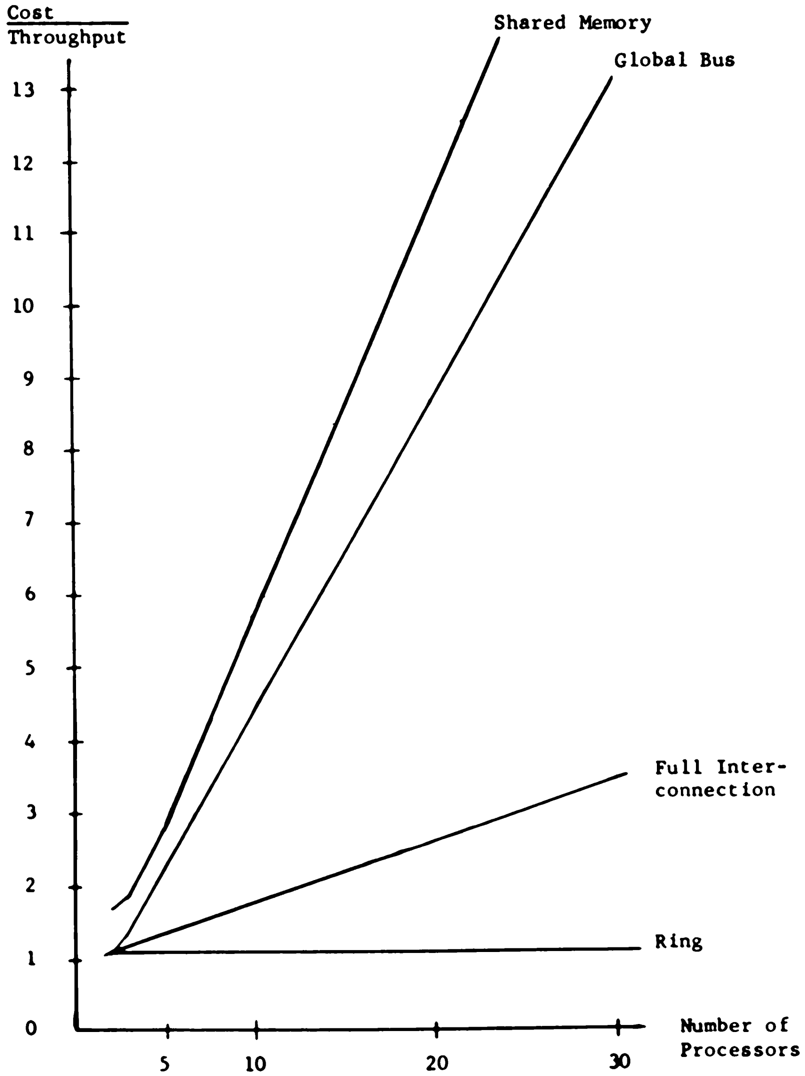


Figure 3. Normalized system cost/throughput vs. number of processors.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch015

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch015

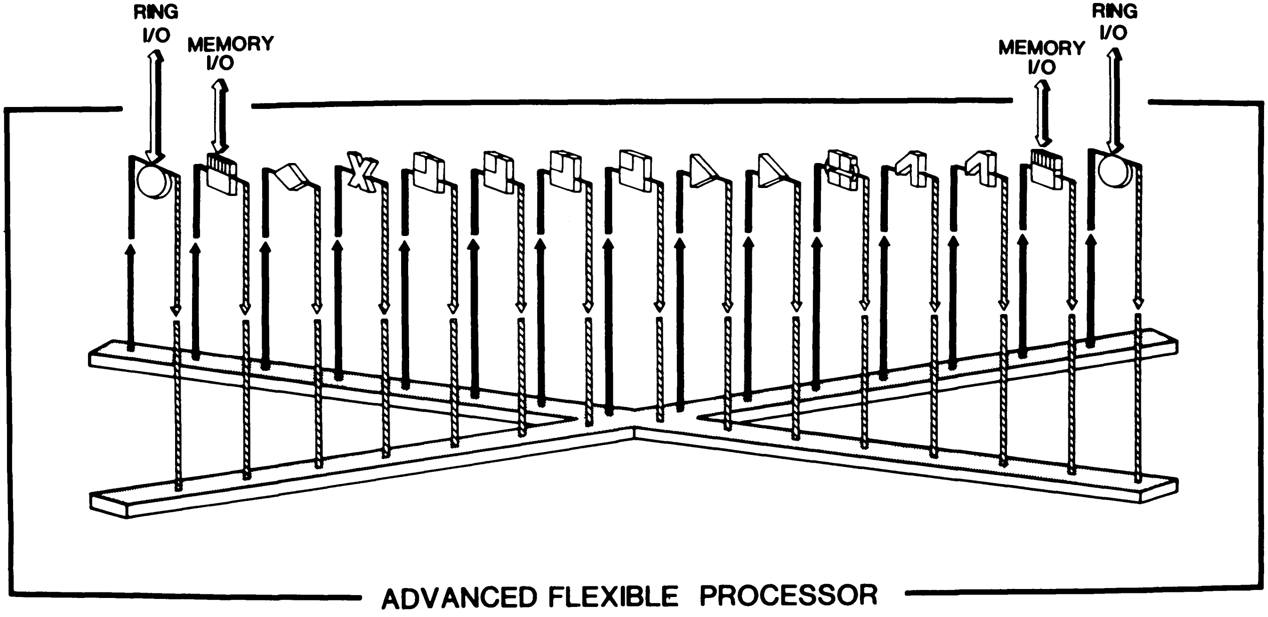


Figure 4. Functional organization of the advanced flexible processor.

TABLE I. FUNCTIONAL UNIT BREAKDOWN

Number of Units	Type of Functional Unit	Number of Pipelined Segments
2	External Memory Access Unit	1
2	Ring Port I/O Units	1
1	Control Unit	2
2	Adders Unit	2
1	Multiplier Unit	3
2	Shift Boolean/Logic Unit	2
4	2K Data Memory Units	2
2	8 Word File Registers	2

Computations may be streamed through the Advanced Flexible Processor very efficiently due to dual I/O port characteristics of the internal architecture. Data elements may be independently streamed in and out of the Advanced Flexible Processor through any one or all of the four I/O channels. For example, data may be streamed in through one of the memory I/O channels, computations performed, and then streamed out through one of the other three I/O channels simultaneously.

Multifunctional Parallelism. The internal architecture of the Advanced Flexible Processor allows multiple computational streams to be constructed and executed in parallel. By way of example, one might imagine the multiply unit requesting operands from one of the memory I/O ports and one of the data memories, while at the same time an adder may be requesting the product computed by the multiplier on a previous machine cycle and another data element from one of the remaining three data memories to serve as input operands for an addition operation. At the same time, the remaining adder may be using the sum produced by the first adder on a previous machine cycle and the result from a shift boolean operation to perform a subtraction. The ultimate goal, of course, is to get as many functional units

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch015

executing simultaneously as possible and thereby achieving the highest concurrency of execution.

Each of the internal functional units of the AFP are I/O buffered to their respective crossbar ports as shown in Figure 5. Each functional unit is equipped with input latch registers, buffering the crossbar inputs, and output latch registers, buffering the functional unit outputs to the crossbar. This design allows the intermediate storage of variables between the function units and thus allows the functional units of the AFP to be pipelined together with the maximum flexibility. Single or multiple pipelined chains are easily supported through the crossbar as a result of this method of "direct data hand-off" between the functional units.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch015

Advanced Flexible Processor Performance

The machine cycle time of the Advanced Flexible Processor is 20 nanoseconds. Every functional unit can provide results every 20 nanoseconds. Thus, 50 million 16-bit multiplies, 200 million 16-bit data memory references, and 100 million 16-bit adds or subtracts, etc. can be performed every second. The maximum operational speed of the Advanced Flexible Processor, therefore, is 800 million operations per second when all 16 functional units are executing.

AFP I/O Performance. The ring port I/O unit provides the interface for each Advanced Flexible Processor to the ring interconnect system. Two ring ports are provided to each Advanced Flexible Processor and thus the capability for dual-ring interconnection systems exists. The ring port I/O unit handles all of the data management, synchronization, and protocol required to communicate on the ring system without interrupting the arithmetic processing of the Advanced Flexible Processor.

The external memory access unit provides the interface between the AFP and the central, high-performance, random access memory store. Each external memory access unit can provide peak data I/O rates of 3.2 billion bits per second and sustained I/O rates of 800 million bits per second. Thus, the total sustained capability of an Advanced Flexible Processor from the two ring port I/O units and the two external memory access units is 3.2 billion bits per second.

AFP Computational Performance. The multiply unit of the Advanced Flexible Processor provides the capability to produce two 16-bit products or one 32-bit product every 20 nanosecond machine cycle. The multiplier also provides the capability to do population and significant counts. The two adders provide the capability of performing four 8-bit adds, two 16-bit adds, or one 32-bit add every 20 nanosecond machine cycle. The shift

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch015

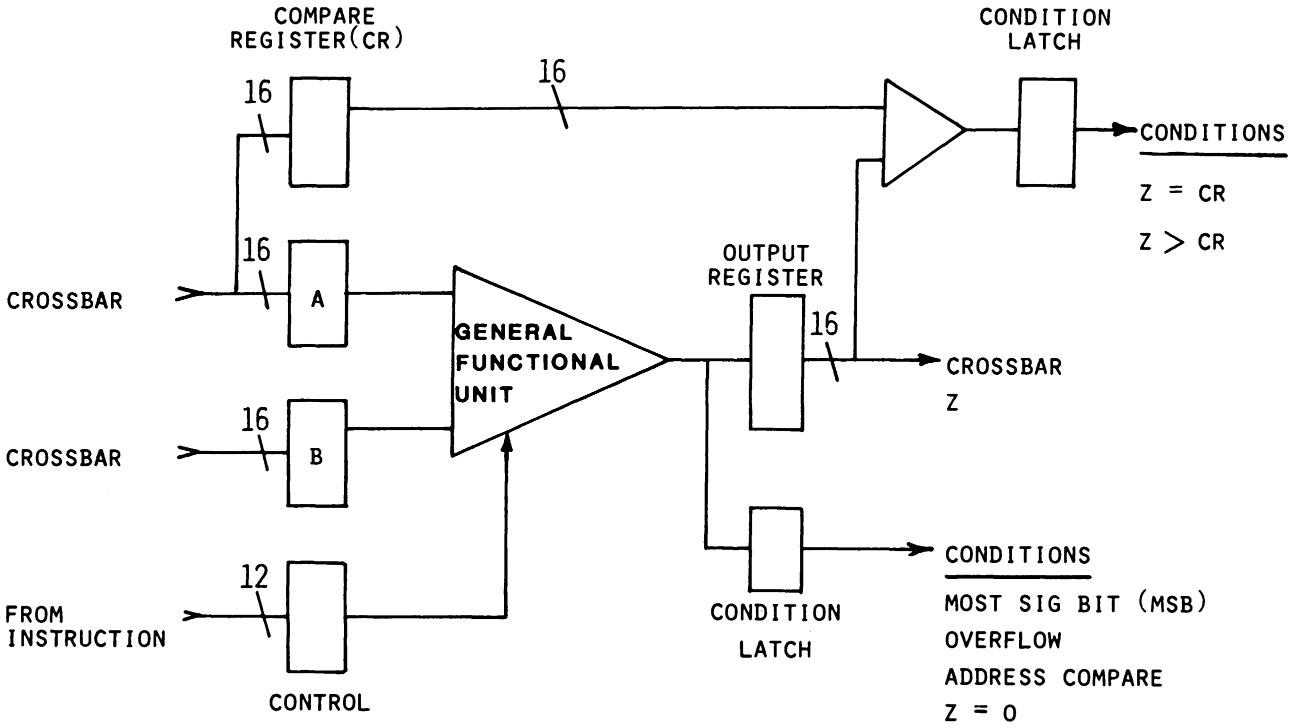


Figure 5. Register level organization of a generic AFP functional unit.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch015

boolean units allow barrel shifts of up to 15 bits performed every 20 nanosecond machine cycle and is capable of performing all of the 16 basic boolean logic functions. Each data memory allows the reading or writing of one 16-bit word every machine cycle. The file memories allow the reading and writing of four 16-bit words every machine cycle. The control unit manages program execution and handles branching and accessing of programming instructions.

The individual program memory within the control unit of each AFP consists of 1,024 program instructions. Each program instruction is 200 bits wide and provides the capability of issuing 39 instruction parcels every 20 nanoseconds. The control bandwidth of the AFP is thus very high, and allows a flexibility control flexibility in control for the easy management and execution of the 16 functional units and the crossbar reconfiguration on a machine cycle basis. As a result, the Advanced Flexible Processor is capable of performing 100 million, 250 million, or 500 million arithmetic and logic operations every second in the 32-bit, 16-bit, or 8-bit modes of operation respectively.

Comparison Testing. Latching registers as shown in Figure 5 are also provided within each funtional unit for the storage of input comprand values. Arithmetic computations and testing of resultant outputs can thus be concurrently performed within all of the functional units. The current conditional status of each functional unit can therefore be provided to the control unit every machine cycle for branch decision processing. When counting all of the arithmetic and logic operations plus all of the comparison results provided by each of the functional units, one finds the Advanced Flexible Processor capable of performing an astounding 2.9 billion, 8-bit operations per second. This compute capability represents the upper theoretical limit for the Advanced Flexible Processor.

On average, a typical computational process can keep four of the arithmetic functional units plus several memory and I/O units busy concurrently, allowing a single AFP to achieve an average computational rate of about 200 to 250 million 16-bit arithmetic and logical operations per second.

The features provided by a single AFP are summarized in Table 2. The very modular construction of AFP systems and of the AFP itself allows for very cost-effective system implementation. The modularization of functional units about the crossbar interconnect allows the enhancement of AFP performance specification by replacing existing functional units with specialized functional units designed specifically to meet performance requirements. Typical examples of specialized functions are: fast fourier transform units, floating point add, multiple, and divide/square root units.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch015

TABLE II SINGLE AFP FEATURES

FEATURE	ADVANTAGE
250 MILLION ARITHMETIC COMPUTATIONS PER SECOND FOR EACH AFP	EXPANDABLE COMPUTE POWER TO MATCH APPLICATION
FUNCTIONALLY DESIGNATED INTERMEDIATE OPERAND REGISTERS	ALLOWS UNINTERRUPTED COMPUTATION STREAMING, ELIMINATING REGISTER RESERVATION HICCUPS
DIRECT DATA HAND-OFF BETWEEN 16 FUNCTIONAL UNITS THROUGH CROSSBAR SWITCH	PROVIDES BROADEST CAPABILITY FOR MULTIPLE CHAINING WITH NO REQUIREMENTS ON OPERAND INDERDEPENDENCE
DATA FAN OUT OF 1:16 ON ALL FUNCTIONAL UNITS	ELIMINATES OPERAND CONTENTION, ALLOWING MULTIPLE USE OF A SINGLE OPERAND IN ONE MACHINE CYCLE.
FOUR INDEPENDENT DATA MEMORIES PROVIDING CONCURRENT ACCESS AND COMBINED CAPABILITY TO SUPPLY 16 INPUT REQUESTS SIMULTANEOUSLY	PROVIDES 8 KB OF CIRCULATING VECTOR STORAGE, AVOIDING COSTLY VECTOR LENGTH START-UP TIMES
FOUR INDEPENDENT I/O PORTS PROVIDING SIMULTANEOUS READ/WRITE ACCESS TO HPR MEMORY	ELIMINATES VECTOR LENGTH HICCUPS IN COMPUTATION STREAM PEAK BANDWIDTH 8 BILLION BITS/SECOND SUSTAINED BANDWIDTH 3.2 BILLION BITS/SECOND
200 BIT WIDE INSTRUCTION PACKET	INSTRUCTION ISSUE RATE IS 39 INSTRUCTION PARCELS /CYCLE OR 2 BILLION INSTRUCTIONS PER SECOND
TRANSPARENT SINGLE LEVEL INTERRUPT EXCHANGE MANAGEMENT	NO SPECIAL INTERRUPT EXCHANGE SOFTWARE PACKAGES REQUIRED
INSTRUCTION CACHE SIZE OF 1024 INSTRUCTION PACKETS, EACH 200 BITS WIDE	40 THOUSAND INSTRUCTION PARCELS PER PROGRAM INTERVAL

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch015

AFP System Architecture

System arrays of Advanced Flexible Processors are linked together and synchronized via facilities provided by the ring port functional units. Data elements 16 bits wide, along with 12 bits of control information, are passed between ring ports on adjacent AFP's. The control information provides all of the associated addressing information to define the single processor or subset of system processors to which the message is to be sent.

Information identifying the appropriate data register file in which the incoming data element is to be stored is also contained within the control field of the ring packet. Each data memory is capable of defining 16 independent data files. Designated bits within the control field provide interprocessor synchronization information as well. Facilities within the ring port provide the logic capabilities to use these designated bits to achieve cross file synchronization. These features assure that a processor is not capable of beginning a computational task until the appropriate single data file or set of data files which are to be used as operands in the pending computation are stored away in the processor.

These synchronizing control features also prevent another processor from over-writing files within a computing processor. Input and output FIFO buffering provides elasticity in communication between processors on the ring systems to minimize processor idle time. Thus, due to the built in capabilities of the ring port functional units, the processing elements are released from the inflexible lock-step synchronization required of other single instruction, multiple data stream (SIMD) machines and multiple instruction, multiple data stream (MIMD) machines. Further, the system allows for the construction of multiple elastic pipelines to be created across system AFP's, which function as powerful processing elements in the dual ring connected architecture.

A Minimum AFP System. A minimum AFP system configuration would consist of a host computer, presently a PDP 11/70, communicating with a single AFP via a modified ring port interconnection, MRP/C (Figure 6). An AFP operating as an attached processor in this configuration would enhance system performance of the host processor by providing a capability of 250 million additional arithmetic computations per second. The ring port interface units through which which rings of AFP's may be interconnected are indicated in figure 6 by the abbreviation RP().

Multiprocessor AFP Systems. AFP's can be easily added to the minimum system shown in Figure 6. A typical multiprocessor expansion is shown in Figure 7. AFP's are interconnected on the

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch015

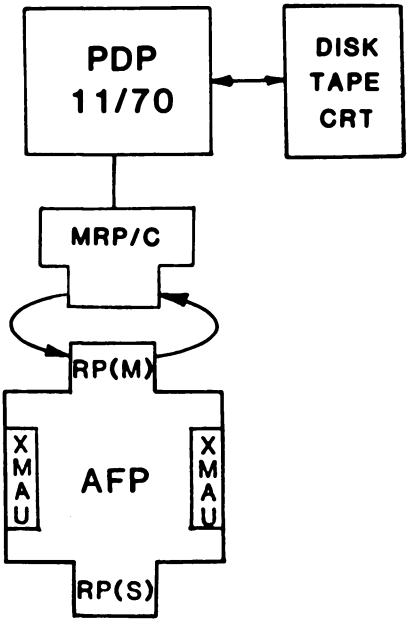


Figure 6. Minimum AFP system configuration.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch015

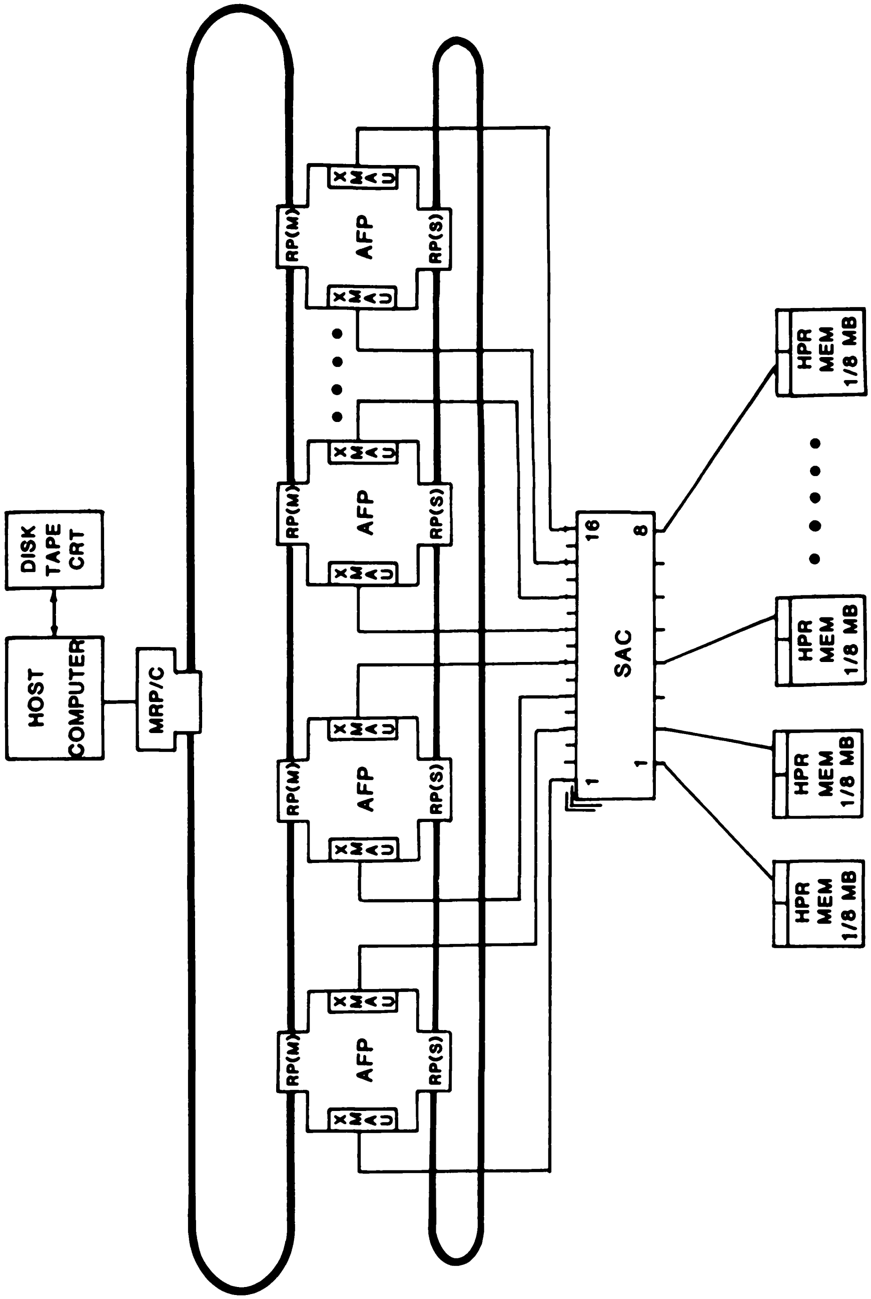


Figure 7. Typical AFP system configuration showing capabilities for expansion.

host ring with each additional AFP augmenting the computational capabilities of the system by 250 million arithmetic operations per second. An additional ring interconnection channel, shown in Figure 7, is also provided for interprocessor communication and control. Up to 256 Advanced Flexible Processors can be supported on each system ring.

Centralized High Performance Memory. A multiprocessor system of AFPs may share a common, high-performance random access memory store (HPR) between processors. All system HPR requests sent from the external memory access units (XMAU) of the AFP's are managed by the Storage Access Controller (SAC). Multiple SAC's may be employed as memory requirements are expanded. Each SAC is capable of transferring data to and from the AFP array at a sustained rate of 6.4 billion bits per second.

This centralized, high-performance memory store may be expanded from 125 kilobytes to 16 million bytes, providing a maximum memory bandwidth of 12.8 billion bytes per second. The advanced technique of processor intercommunications significantly reduces processor idle time. Processor idle time is further reduced through a sophisticated hierarchical approach to mass memory and I/O management, which ensures continuous data support to the processing elements and a continuous computational flow. All memory and communication paths are designed to support extremely high bandwidths.

Centralized Mass Memory Facility. In addition to the high-performance central memory, AFP systems may be configured to provide a centralized mass memory facility composed of slower, relatively inexpensive memory technologies that can be accessed by each system AFP. The mass memory hierarchy can be configured to meet individual requirements and may include MOS random-access storage, disks, tapes, and memory archives, as well as high-speed interfaces to general peripheral I/O dvices and display stations.

The MOS Memory technology provides low-cost random access storage at a fraction of a cent per bit. This cost compares to that of the higher performance technology used in the HPR central memory of 3-4 cents per bit. Sustained read/write data rates to and from MOS memory can exceed 1,000 million bits per second. MOS capacity may be expanded from 256 kilobytes to 1 billion bytes to provide ample random access storage at a low cost per bit ratio to cost-effectively meet the storage requirements for very large problems.

AFP System Performance for Specific Applications

A number of specific applications for the AFP have been studied at Informationa Sciences Division. The performance of single and multiprocessor systems of AFP's has been assessed for these

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch015

applications. The computational performance of the Advanced Flexible Processor on a representative set of these alogorithms is shown in Table 3. Beyond these areas of investigation there are yet broader applications for the Advanced Flexible Processor that are being investigated. Data retrieval systems(8) as well as floating point applications are starting to be addressed.

AFP Software Facilities

The programming of the AFP system is presently done through the use of two very powerful software development tools. The first of these tools is the AFP cross assembler, MICA. The second tool is the AFP instruction level simulator, ECHOS. The MICA cross assembler and the ECHOS instruction level simulator allows all programming to be done "off-line." AFP programs can be written using the editor facilities of either a PDP 11/70 or a Control Data Cyber 700 series computer. The edited files are then processed by the MICA cross assembler. MICA checks for all illeagal lexical and syntax usages as well as illeagal hardware usages. Functional unit and cross bar usage conflicts are identified to the programmer through the facilities of MICA. MICA produces a binary file of the submitted program which runs directly on the Advanced Flexible Processor.

The binary file produced by MICA also runs directly on ECHOS the AFP instruction simulator. ECHOS provides a register level simulation of the submitted program. ECHOS interactively executes the program in software precisely the way the program will run in the Advanced Flexible Processor. A programmer can single step through his program specifying the print out of all or a selected set of functional registers in the AFP. The accuracy, power, and detail of the ECHOS simulator allows a programmer to confidently expect his program to run the very first time it is run on an AFP. Thus, programming activities can be carried out with no interruption to useful AFP system data processing.

Development of higher level programming languages for the AFP is currently underway. FORTRAN, ADA, and the data flow language VAL which has been developed at the Massachusetts Institute of Technology and the Lawrence Livermore National Laboratory(9) are all candidates to be supported.

Summary

The Advanced Flexible Processor is a unique entry into the multiprocessing field. It provides the dynamic capabilities offered by an MIMD machine with advanced features provided by the interprocessor ring communications network; efficient utilization of the system processors is therefore effected. Within each Advanced Flexible Processor, dynamic multiple chaining can be achieved due to the superior flexibility of the

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch015

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch015

TABLE III AFP APPLICATION PERFORMANCE

APPLICATION	NUMBER OF AFP'S	KERNAL RATE	TOTAL TIME
COMPLEX FFT, 1024 POINT 16 BIT ACCURACY	1 4	80 ns/BUTTERFLY 20 ns/BUTTERFLY	0.4 msec 0.1 msec
GEOLOCATION, 100,000 MESSAGES 100 LOCATIONS OF INTEREST	1	40 ns/PAIR	0.4 sec 10 MILLION COMPARES
2-DIMENSIONSAL MATRIX DECONVOLUTION (55 X 80) ELEMENTS	1	20 ns PER MULTIPLY-ADD	6.6 msec
MULTISPECTORAL CLASSIFICATION (128 X 128) POINTS	16	480 ns/POINT COMPUTATION 1240 OPERATIONS PER POINT	8 msec 2.58 BILLION OPERATIONS/SEC
MATRIX INVERSE (50 X 50) POINTS	1	2 usec/POINT	5.0 msec
MATRIX TRANSPOSE (32 X 64) ARRAY (1024 X 1024)	1	10 nsec/POINT 20 nsec/POINT	20 usec 21 msec

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch015

intra-processor crossbar. Multiple functional units can be executed simultaneously with each functional unit providing a broad range of instruction defined operational capabilities. Functional unit make up within an Advanced Flexible Processor can be varied and optimized for variable data sets. Multiple comparisons are available within each machine cycle for simultaneous multiple condition sensing.

Special-purpose functional units can replace existing functional units within the Advanced Flexible Processor, allowing processor capabilities to be tailored to the precise application requirements. Modular system construction allows compute power modularity; thus, processing systems can be cost-effectively tailored to the users individual requirements.

Future Computational Trends. The trends in computational requirements over the last 25 years has increased logarithmically by an order of magnitude every 8 years. Users will continue to demand higher performance, computational facilities at a rate matching or surpassing that of the previous 25 years. The demand appears to be insatiable as long as cost effectivity can be sustained. Semiconductor technology has been able to meet these demands over the past 25 years, however, presently there appears to be a slowing in the rate of technological advances within the semiconductor area. A circuit density increase by a factor of two every year as predicted by Moore's law is not presently being met, due to the increased problems in semiconductor fabrication that are being confronted.

There is little evidence that this trend will reverse itself over the coming years. The current trend indicated by the widening of this technological gap, seems to indicate that the only way to meet the computational requirements of the scientific community in the mid 1980s is through the application of multiprocessor technology. Developing the skills to employ multiprocessor technology to solve the large scientific problems that are presently being proposed will provide the foundation to bridge the computational gap between 1985 and 1990. Future advances in semiconductor technologies will yield increases in computational speed at the circuit level. Those advances will certainly be incorporated into multiprocessing hardware, and thus the skills we develop to employ multiprocessing in the 1980's will as well provide the stepping stones to meet the computational demands of the 1990's.

Acknowledgements

The information presented in this paper is a direct result of the collective knowledge gained by the personel in the Information Sciences Division of Control Data. This knowledge has been gained from more than nine years of experience in the field of multiprocessing.

Literature Cited

1. Allen, G.R., A Reconfigurable Architecture for Arrays of Microprogrammable Processors, Special Computer Architectures for Pattern Processing, Purdue University, West Lafayette, Inc., CRC Publishing Corporation, to be published in 1981.
2. Hsu, T.T., On the Performance and Cost Effectiveness of Some Multiprocessor Systems, 1977 International Conference on Parallel Processing, August 1977.
3. Stenshoel, C.R., Production Image Processing System Design Study, Control Data Corporation Final Report to Centre National d'Etudes Spatiales, May 1977.
4. Juetten, P.G. and Allen, G.R., An Image Processor Architecture, Control Data Corporation, Minneapolis, Minnesota, 1977.
5. Allen, G.R. and Juetten, P.G., SPARC - Symbolic Processing Algorithm Research Computer, /Proc. Image Understanding Workshop/, Science Applications, Inc., Report No. SAI-79-814-WA, 1978.
6. Allen, G.R., Advanced Image Processing Systems Design Studies, Control Data Corporation Final Report to the Rome Air Development Center, Contract No. F30602-76-C-0362, March 1978.
7. Cyre, W.R., Allen, G.R., and Juetten, P.G., Symbolic Processing Algorithm Research Computer Progress Report, Control Data Corporation, Minneapolis, Minnesota, 1978.
8. Cyre, W.R., Applications of a Reconfigurable Array of Flexible Processors in Intelligence Information Retrieval, Control Data Corporation Final Report to the Rome Air Development Center, Contract No. F30602-78-C-0065, July 1979.
9. Ackerman, W.B. and Dennis, J.B., VAL-A Value-Oriented Algorithmic Language: Preliminary Reference Manual, Laboratory for Computer Science, Massachusetts Insititute of Technology

RECEIVED July 21, 1981.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ch015

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ix001

INDEX

A

Accuracy demands of simulations 132
Accuracy of the model 150
N-Acetyl(alanyl)$_3$-*N*-methylamide,
    conformations 169
N-Acetyl-*N′*-methylalanine, energy 165*f*
Activation barrier 55
Advanced flexible processor (AFP) 245
  array architecture 245–267
  cross assembler, MICA 264
  functional
    organization 254*f*
    unit breakdown 255*t*
    unit, register level organization 257*f*
  hardware overview 246–247
  instruction level simulator,
      ECHOS 264
  multifunctional parallelism 255
  performance 256
    application 265*t*
    benefits 247
    computational 256
    I/O 256
  software facilities 264
  specialized functions 258
  system(s)
    architecture 260
    comparison testing 258
    configuration(s) 260, 261*f*, 262*f*
    interconnection technology 247
    multiprocessor 260–263
    performance for specific
        applications 263
Advanced scientific computer (ASC)
  central processor 66
    and memory configuration 67*f*
  chemistry of reactive flow 74
  conditionality, effect 71
  disc storage 68
  hardware and software performance 69–73
  integration method 74
  software 68–69
  system hardware 66–68
  vector
    computer, chemical rate
        equations 65–77
    math functions 75
    length, effect 71
AFP (*see* Advanced flexible processor)
AFPP (Attached floating point
    processor) 193
Agarwal refinement program,
    myoglobin 149
Algorithm(s)
  change detection 246
  FCT 94
  integral method 12
  interpretive 117
  maximal-substructure 122
  Monte Carlo (MC) 125
  outer product 223, 225*t*
    matrix multiplication 225*f*
  parallel 112
  STIRS 119
  supermatrix method 12
Algorithm 1 18–21
Algorithm 2 21
  disadvantages 15
  for matrix multiply 14, 15
  theoretical performance 15
Alkane pyrolysis reaction mechanism 81
Allosteric change of protein 156
Amide (NH) stretching frequencies 180*t*
AP (Array processor) 193
Application performance, AFP 265*t*
Architectural concept(s) 247
  ring connected 250–252
Architectures, cost effectiveness 252
Arithmetic capability of microcomputer 194
Arithmetic processor(s)
  chip
    Am 9511A 195
    command mnemonics 197*t*
    execution times 198*t*
  integrated circuit 196*f*
  scalar 194–202
  unit, hardware interface between
      a microprocessor and
      Am9511A 200*f*
  unit to S-100 bus hardware
    interface 203*f*
  vector 208–210
Array(s)
  architecture, AFP 245–267
  in Multilin 240
  operations, maximum vectorization 71
  processing, extent 87
  processor (AP) 193

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ix001

- ASC (*see* Advanced scientific computer)
- Asynchronous iteration techniques 16
- ATMOL closed shell SCF program 12
- Atomic coordinates, precision 150
- Atomic trajectories, computation 182
- Attached floating point processor (AFPP) 193

B

- Back transformation procedure 60
- Basic linear algebra subroutine (BLAS) 132, 133
- Basis functions 52
- Basis sets, scheme for extending 31
- BELLCHEM code 81
- Benchmark
  - formula 220, 221*t*
  - FORTRAN/VPLIB code 221*t*
  - timings 205*t*
    - on various mini- and mainframe computers 222*t*
- Beryllium-like ions, Hartree–Fock calculations 31
- Binary diffusion velocities 96
- Biological macromolecule 161
- Biomolecular systems, computer simulation 161–191
- BLAS (Basic linear algebra subroutine) 132, 133
- Bloch equation 58
  - for density matrix 60
- Bohlender's Pascal extension 238
- Blotzmann average 181
  - distribution of polymer systems 137
- Bond length calculation 231
  - using VSUSQI 232*t*
- Bonding matrix 233
- Boundary grid used in the two-dimensional model 97*f*
- Box scheme 137, 140
- Breadth first search 113
- Brute force techniques 96
- Buoyancy effect on thermal ignition 89–107
- Buoyancy related to IPM 90

C

- Carbon monoxide, effect of buoyancy on thermal ignition 89–107
- CDC 7600 computer 129
- Central processor, ASC 67*f*
- Centralized high-performance memory of AFP system 263
- Centralized mass memory facility of AFP system 263
- Centrifugal sudden (CS) approximation 55
- Change detection algorithm 246
- Channel, definition 54
- Charged systems, simulations 128
- Chemical
  - rate equations on ASC vector computer 65–77
  - reactions, data base 114
  - reaction dynamics 52
- Classical many particle simulator (CLAMPS) 125, 126
  - programming considerations and timing results 128
  - timing results for computers 129*t*
- Close coupled equations 52, 53*f*
- Close coupling expansion, size 55
- Closed shell case 10–16
- Closed shell SCF calculation on dichromium tetraformate 17
- Coincident indices 11
- Command mnemonics, arithmetic processor chip 197*t*
- Communication system, interprocessor 251*f*
- Comparison testing of AFP systems 258
- Compiler optimization 241
- Computational
  - molceular biology related to supercomputers 146
  - performance, AFP 256
  - trends, future 266
- Computer(s)
  - comparisons 4–5
  - design 143
  - Japanese vs. U.S. 40*f*
  - resources in Japan 39–42
    - academic 39–63
  - resources of Japanese national univertities and institutes 41*f*
  - simulation 182
    - of bimolecular systems 161–191
- Conditionality, effect on ASC 71
- Configuration(s)
  - of AFP system 260
  - determination of polymer 137
  - interaction (CI) method 48
  - microscopic 186
  - related to energy 183
- Conformation(s)
  - of *N*-acetyl(alanyl)$_3$-*N*-methylamide 169
  - manipulations 156
  - of protein 146
  - thermodynamics 167
  - analysis 161–176
  - energy calculations 145
  - entropy 169
  - equilibria of peptides, entropy contribution 170*t*, 171*t*, 176*t*
- CONGEN 119
  - molecular structure postulation 121–122

- Correlation energy, diagramatic expansion 25*f*
- Correlation energy, plot 34*f*
- Cost
  - effectiveness of architectures 252
  - effectiveness improvement 83
  - /throughput vs. number of processors 253*f*
- Coulomb interaction 163
- Counting channels for the FH$_2$ reaction 57*f*
- Counting of channels for the H$_3$ reactions 57*f*
- Coupling terms 167
- CPU
  - CRAY-1 3*t*
    - times 27*t*
      - for hydrogen sulfide molecule 5
      - on IBM 360/195 computer 27*t*
- CRAY
  - FORTRAN compiler, CFT 10
  - and Honeywell hardware features, comparison 82*t*
  - performance 15
  - quantum chemistry codes 5–10
  - recent progress 23–33
  - vector mask feature 17
- CRAY-1 1, 130
  - CPU times 27*t*
  - main characteristics 3*t*
  - optimized 86*t*
  - quantum chemistry codes 6*t*
  - rate of computation for matrix multiplication 2
  - relative performance 4*t*
  - timings for quantum chemistry packages 7*t*
- Crystal structure determinations 143
- Crystallographic studies of biological macromolecules 149
- Cyclic hexapeptide crystal 186
  - packing 187*f*

D

- Data
  - base(s)
    - of chemical reactions 114
    - design considerations 156–157
    - in mass spectra, size 117
    - structure 143
  - collection, improved methods 149
  - memory of AFP system 260
  - representations 239
- Degrees of freedom, peptide bond 163
- Degrees of freedom of sulfur hexafluoride 62
- Density matrix 58
  - Bloch equation 60
- Depth first search 113
- Design
  - considerations for data bases 156–157
  - criteria for VPLIB library 210
  - philosophy 238–239
- Diagrammatic many-body perturbation theory 24
- Dichromium tetraformate, closed shell SCF calculation 17
- Difference spectra 177, 178*f*
- Diffusion coefficient related to polymer 138
- Diffusion fluxes, time to compute 99*f*
- Dipeptide unit 163
- Direct method (DM) 80
- Disc storage on ASC 68
- Divergence of velocity, equation 91
- Double resonance spectroscopy 62
- Dummy Index 241
- Dynamic effects 168

E

- ECHOS, AFP instruction level simulator 264
- Effective startup time, definition 21
- Eigenvalues 48
- Eigenvectors 48
- Einstein equations 167
- Electromagnetic radiation 58
- Electron density distribution 148
- Electron correlation 22–23
  - energy
    - in atoms and small molecules 29*t*
    - calculation(s) 23, 24
    - of ground state of water molecule 30*t*
- Embedding language 240
- Energy
  - of *N*-acetyl-*N'*-methylalanine 165*f*
  - deposition curve 102*f*
  - map of residue 163
  - minimization of conformation 166
  - refinement 162*f*
- Entropy 167
  - conformational 169
  - contribution to conformational equilibria of peptides 170*t*, 171*t*, 176*t*
- Envelope of polymer motion 139*f*
- Enzyme lysozyme 184
- Equilibrium properties, calculation 155
- Evaluation function in organic synthesis 113
- Evolution of biological macromolecule related to large-scale computers and computer graphics 143–158
- Ewald image potential 126
- Excitation dynamics of sulfur hexafluoride multiple photon 60, 61*f*

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ix001

Execution time(s)
  arithmetic processor chip 198*t*
  reduction 71
  with VAX 11-70 132
  for vector addition 70*t*
Expansion limitations of shared memory system 250

F

FH_2 reaction, counting channels 57*f*
FCT algorithm 94
Fingerprints 177
Finite basis set 28
Flexible processor (FP) 246
Floating point
  arithmetic 12, 193
  computation rates 3*t*
  format conversion, microsoft to AMD 206*t*–207*t*
  number representation format 199*f*, 204*f*
  operations (FLOPS) 126
  performance, microcomputer 194–202
  speed 132
Floppy disc 202
Fluid dynamics equation 94
Flux-corrected-transport technique 94
Fock matrix 14
  construction 33
Folded structure 169
Formula for benchmark 220, 221*t*
FORTRAN
  code 18, 20*f*, 26, 74
    for matrix multiplication using VPLIB 213*t*
    for a pairwise sum 127*t*
      vectorized version 131*t*
    for pivotal condensation matrix inverter 225*t*
  coding 126
    of Gauss–Jordan matrix inverter 226*t*–227*t*
  compiler (CFT on the CRAY) 5
  system
    microcomputer 202
    scalar 202–208
    vector 210–219
  vector math library functions 71*t*
  /VPLIB code, benchmark for 221*t*
  /VPLIB code for Gauss–Jordan matrix inverter 228*t*–229*t*
Fourier amplitude method 80
FP (flexible processor) 246
Frame of the $l = 20$, $N = 5$ system 139*f*
Frequency modes 179*t*
Function of biological macromolecule related to large-scale computers and computer graphics 143–158
Functional organization of AFP 254*f*
Functional unit breakdown of AFP 255*t*
Future computational trends 266

G

G-matrix, computation 14
Gas chromatography (GC) 117
GATHER function 130
Gather–scatter operations 62, 133
Gauss–Jordan matrix inverter, FORTRAN coding 226*t*–227*t*
Gauss–Jordan matrix inverter, FORTRAN/VPLIB code 228*t*–229*t*
Gauss–Rys quadrature method 8
Gaussian integrals, procedure for evaluating 9*f*
Gaussian integrals program 5
GC (gas chromatography) 117
Generate phase, structure elucidation 118
Geometric quantities, calculation 230
Geometry of interfaces 156
Global bus system 250
Global optimization 239
Globular proteins 144
  structures 147
Glycine 177
Graphical Unitary Group Approach (G.U.G.A.) 23
Graphics programs for analysis of static structures 151
Green's function (GF method) 80
  matrix 81
Grid size, effect on timings 98
Grid system used in two-dimensional model 95*f*

H

H_3 reaction, counting of channels 56*f*
Hamiltonian 60
  matrix 48
Haptoglobin protein 161
  alpha-carbon plot of the chain 162*f*
Hardware
  features, comparison of Honeywell and CRAY 82*t*
  interface, arithmetic processor unit to S-100 bus 203*f*
  interface between a microprocessor and Am9511A arithmetic processor unit 200*f*
  overview, AFP 246–247
  performance 73*t*
    of MVP-9500 208
  /software design 202
Harmonic potential 137

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ix001

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ix001

- Hartree–Fock
  - basis set 5
  - calculations on beryllium-like ions 31
  - energy surface search 16
  - equations 28
- Head-per-track (H/T) 68
- Helical conformations 169
- α-Helix 146
- Heuristic search 111, 118
- Hexafluoride, infrared multiple photon excitation dynamics of sulfur 60
- High
  - -performance liquid chromatography (LC) 117
  - -performance random access memory store (HPR) 263
  - -speed vector machine 79–88
- Honeywell and CRAY hardware features, comparison 82*t*
- Human component 157–158
- Hydration phenomena 183
- Hydrogen bonding 147, 163

I

- IBM 360/195 computer, CPU times 27*t*
- IBM 360/195 computer, timing 26
- IBM 370/165 2
  - CPU performance 5
  - timings for quantum chemistry packages 7*t*
- Ignition energy curves 103*f*
- Ignition energy, summary 106*f*
- Image processing functions 246
- Implementation of Multilin 240
- Index pairs 13
- Induction parameter model (IPM) 90
  - buoyancy related to 90
  - calculations 90
- Induction time for the $CO + 2O_2$ mixture 92*f*
- Infinite order sudden (IOS) approximation 55
- Information retrieval problem, organic synthesis 110
- Infrared multiple photon excitation dynamics of sulfur hexafluoride 60
- Institute for Molecular Science 42
  - machine-time requests 43*t*
- Instruction processing units (IPU) 66
- Integral
  - block type 10
  - method, effect of sparsity on relative performance 12
  - method, timings 12
  - transformation 17–18
- Integrated array processer (IAP), Japan 14
- Integrated circuit, arithmetic processor 196*f*
- Integration method for ASC 74
- Intercommunications architecture for multiprocessing 250
- Interconnection technology for AFP systems 246
- Interfaces, geometry 156
- Interfacing to control processor 201
- Interfacing multiple devices to S-100 bus 208
- Interpreter generation of possible structures 123
- Interpretive algorithms 117
- Interpretive system, functions 118
- Interprocessor communication system 251*f*
- Inverse matrices, calculation 234
- Ionospheric deionization timing test 76*f*
- IPM (*see* Induction parameter model)
- Isomorphous derivatives 148

J

- Japan, academic computer resources 39–63
- Japanese vs. U.S. computers 40*f*

K

- Kinetic sensitivity analysis, high-speed vector processor machine for chemical 79–88
- Kinetics of a chemical mechanism, ODE problem 79

L

- LiFH reaction, counting channels 59*f*
- Language design 239
- Laser pulse propagation 62
- Least-squares refinement 149
- Lennard–Jones potential 137
- Line drawings of macromolecules 154
- Linear array 193
- Lockstep synchronous procedure 16
- Lysozyme crystal, triclinic, water of hydration 185*f*
- Lysozyme, enzyme 184

M

- Machine speed, effects of optimization and vector length 72*f*
- Machine-time requests to Institute for Molecular Science 43*t*
- Macromolecular structure, function, and evolution 143–158
- Macromolecules, refinement techniques 149–150

- Manipulation of conformations 156
- Mapping studies 165–166
- Mass spectra, unknown, molecular structure elucidation 117–124
- Mass spectrometry (MS) 117
- Math library function, FORTRAN vector 71*t*
- Matrix
  - inverter, FORTRAN code for pivotal condensation 225*t*
  - multiplication 18, 223
    - timing 4*f*
    - using VPLIB, FORTRAN code 213*t*
  - operations 54
- Maximal-substructure algorithm 122
- Memory
  - configuration, ASC 67*f*
  - of CRAY-1 3*t*
  - size of simulations 132
- *N*-Methylacetamide 168
- Metropolis algorithm 183
- MICA, AFP cross assembler 264
- Microvector processor for molecular mechanics calculations 193–236
- Microcomputer(s)
  - floating point performance 194–202
  - system, vector processor 211*f*
  - vector processor card for S-100 bus 209*f*
- Microprocessor and Am9511A arithmetic processor unit, hardware interface between 200*f*
- Microscopic configurations 186
- Microsoft to AMD floating point format conversion 206*t*–207*t*
- MIMD (Multiple instruction, multiple data stream) 246
- Minimum AFP system 258
- Model polymer system 136–138
- Modular change detection system 248
- Molecular
  - calculations 28
    - mechanics 231–234
      - microvector processor 193–236
  - dynamics (MD) 125
    - flexibility and 166
    - method(s) 182, 183
    - simulation, relative performances 125–134
  - electronic structure 23
    - calculations 48–51, 48*f*–49*f*
  - geometry 230
  - mechanics program 233
  - structure(s)
    - elucidation from unknown mass spectra 117–124
    - generation of possible combinations 119
    - postulation with STIRS and CONGEN 121–122
- Monte Carlo (MC)
  - algorithm 125
  - method(s) 135, 182, 183
  - simulation, probability map of water positions 186, 188*f*
- MS (mass spectrometry) 117
- Multifunctional parallelism with AFP 255
- Multilin 238–240
  - arrays 240
  - implementation 240
- Multilinear algebra 239
- Multiple instruction, multiple data stream (MIMD) 246
- Multiple photon excitation dynamics of SF_6 61*f*
- Multiprocessing, early research and development 246
- Multiprocessing, intercommunications architecture 250
- Multiprocessor AFP systems 260–263
- Multiprocessor taxonomies 249–250
- MVP-9500, future possibilities using 234–235
- Myoglobin, refinement 149

N

- N–H stretching frequencies 177
- Newton–Raphson technique 16
- Newton equation of motion 182
- Nonbonded interaction 163, 168

O

- ODE problem, kinetics of a chemical mechanism 79
- ODEs, solution of matrix 81
- Open shell SCF case 16
- Operating system (OS) services 68
- Optimization, amount 70
- Optimization and vector length on machine speed, effects 72*f*
- Ordinary differential equations 54
- Organic synthesis
  - computer memory, limitations imposed 113–114
  - illustrating program's understanding of molecular symmetry 115*f*
  - illustrating reactions available in program 115*f*
  - information retrieval problem 110
  - need for computer 110–111
  - program input 114–116
  - software development 113
  - and supercomputers 109–116
- Outer product algorithm 223, 225*t*
  - matrix multiplication 225*f*

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ix001

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ix001

P

Packing in cyclic hexapeptide crystal 187*f*
Pair indices 11
Parallel algorithms 112
Parallel processing 112, 238, 248, 249*f*
Partitioning of A, Q, and X 19*f*
Peak vector efficiency using MVP-9500 223
Peptide(s)
  bond 163
    degrees of freedom 163
    torsion angle 163
  entropy contribution to conformational equilibria 170*t*, 171*t*, 176*t*
  secondary structures 172*f*–175*f*
  solvent systems 184
  structural unit 164*f*
  -water interactions 181
Performance
  AFP 256
    benefits 247
    for specific applications 263
  ACS hardware and software 69–73
  of VPLIB routines 220
Perturbation theory 58
  diagrammatic many-body 24
Pipeline structure 66
Pipelined and parallel processing 248, 249*f*
Point group symmetry, effects 13
Poisson equations 94, 96
Polymer(s)
  configuration determination 137
  diffusion coefficient related to 138
  dynamics 138–140
  motion, envelope 139*f*
  potential of model 136
  probability density 138
  properties, effects of excluded volume 135
  relaxation times related to 138
  research,theoretical models for 135
  simulations, need for supercomputers in time-dependent 135–142
  system(s)
    Boltzmann distribution 137
    dynamics 136
    model 136–138
  viscosity 140
Positioning-arm-disk (PAD) 68
Potential energy
  computation 125–126
  evaluation 233
  of a molecule 166
  surface(s) 52, 56*f*
    calculations 51*f*
    contour map 53*f*
    determination 50
  of trimer 269
Potential of model polymer 136
Precision of atomic coordinates 150
Primary structure of protein 146
Probability map of water positions from Monte Carlo simulation 186, 188*f*
Program
  input for organic synthesis 114–116
  optimization 87
  partitioning strategies 247
Programmer efficiency 239
Programming considerations using CLAMPS 128
Programming language for supercomputers 237–243
Protein(s)
  allosteric change 156
  chemistry, computational, approaches to problems 150–151
  conformation 146
  data bank 150
  haptoglobin 161
    alpha-carbon plot of chain 162*f*
  stability 147
  structure(s)
    determination 148
    globular 147
    need for supercomputers 155
    primary 146
    quaternary 147
    secondary 147
    tertiary 147

Q

Quantum
  chemical dynamics 52–58, 53*f*
  chemistry
    codes on the CRAY 5–10, 6*t*
    codes, integral evaluation 5–10
    packages, implementation and performance 2–5
    packages, timings 7*t*
    use of vector processors 1–37
  levels, number 62
  mechanical structure, determination 48
  mechanics, applications 28
  optics 58–62
Quasicomponent distribution functions 184
Quaternary structure of protein 147
Quench volume, effect 90

R

R-factor, measure of accuracy 148
R-matrix propagation 54
Reaction pyrolysis mechanism 84*t*, 86*t*
Reactive flow on the ASC, chemistry 74
Reactive flow calculations 74
Refinement procedure 148

Refinement techniques for macromolecules 149–150
Register level organization of AFP functional unit 257*f*
Relaxation times related to polymer 138
Reliability of shared memory system 250
Reliability tests 220
Retrieval programs 117
Richards box 148
Ring-connected architectural concept 250–252
Ring port I/O unit 256

S

SF_6, multiple photon excitation dynamics 61*f*
Scalar arithmetic processors 194–202
Scalar FORTRAN system 202–208
Scatter/gather hardware, requirement for 33
SCF procedures 17
Schematic diagram of sperm whale myoglobin 153*f*
Schrödinger equation 52, 58
Secondary structure of protein 147
Self-training interpretive and retrieval system (STIRS) 119
  algorithm 119
  molecular structure postulation 121–122
  possible improvements 122
  prediction(s)
    from the mass spectra of *n*-propyl-*p*-hydroxybenzoate 120*t*
    of maximal substructures 122
    of substructures not present 123
Sensitivity analysis 79–88
  computation, timing 84*t*, 85*t*, 86*t*
Shared memory system 250–252
β-Sheet 146
Simulation(s)
  accuracy demands 132
  of charged systems 128
  memory size 132
  methods, molecular dynamics 125
  methods, Monte Carlo 125
  needs 132
  problem of 126
  relative performances of molecular dynamics 125–134
  techniques 161
Simultaneous differential equations 80
SISCOM 118
Slow flow algorithm 94
Smoluchowski equation 138
Software
  ASC 68–69
  facilities, AFP 264
Software (*continued*)
  tools for analysis of static structures 151
  vectorization 69*t*
Solvation phenomena 183
Solvent effects 181
Solvent interactions 183
Space-filling model of sperm whale myoglobin 152*f*
Space-filling representations 154
Sparse matrix processing 16
Sparsity considerations 12
Specialized functions, AFP 258
Spectra, unknown, with minimal additional information 121
Spectral data, structural information from 118–119
Sperm whale myoglobin, schematic diagram 153*f*
Sperm whale myoglobin, space-filling model 152*f*
Stability of proteins 147
Stabilization energies, estimates 154
Stillinger–Lembery model for water 128
Stiff ordinary differential equations 74
  solver 81
STIRS (*see* Self-training interpretive and retrieval system)
Stress, effect on performance 157
Structural
  chemistry, VPLIB routines 230–231
  information from spectral data 118–119
  patterns, analysis 154
  unit of peptide 164*f*
  units, identification 155
Structure(s)
  of biological macromolecule related to large-scale computers and computer graphics 143–158
  of data base 143
  determination 147–149
    of biological macromolecules, computational aspects 146
  elucidation, generate phase 118
  elucidation, test phase 118
  –generation programs 123
  interpreter generation of possible 123
  postulated, testing 119
  of protein, primary 146
Subroutines for evaluating transcendental functions, VPLIB 212–219, 214*t*–215*t*
Subroutines, optimization 83
Substructures, maximal, STIRS prediction 122
Substructures not present, STIRS prediction 123
Sulfur hexafluoride, infrared multiple photon excitation dynamics 60

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ix001

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ix001

Supercomputer(s)
  approach 122
  at Los Alamos 47
  programming language 237–243
  requirements for theoretical chemistry 47–63
  in the United Kingdom 1–2
  usefulness 112
Supermatrix
  definition 14
  method 11
    timings 12
  P 22
Superposition of 50 frames 139*f*
Symmetries, transpositional 241
Synchronizing control features of AFP system 260
Systems
  dynamics of polymer 136
  hardware, ASC 66–68
  interconnect
    cost performance 252
    expandability 250
    reliability 250

T

Taxonomies, multiprocessor 248–250
Test phase, structure elucidation 118
Tertiary structure of protein 147
Texas Instruments Advanced Scientific Computer (ASC) 65
Theoretical chemistry 39–63
  contributions in Japan 4–6
  related to supercomputer 49*f*
  supercomputer requirements 47–63
Theoretical models for polymer research 135
Thermal ignition of carbon monoxide, effect of buoyancy 89–107
Thermodynamic stability of functional states of biopolymers 144
Thermodynamics of conformation 167
Time
  to compute diffusion fluxes 99*f*
  -dependent polymer simulations, need for supercomputers 135–142
  percentage for vector of length *n* vs. scalar mode 100*f*
Timing(s)
  grid size, effect 98
  of IBM 360/195 computer 26
  for quantum chemistry packages 7*t*
  results using CLAMPS 128
    for computers 129*t*
  of sensitivity analysis computation 84*t*, 85*t*, 86*t*
  test, ionospheric deionization 76*f*
Torsion angles 164*f*
  peptide bond 163
Transition metal chemistry related to supercomputers 50, 51*f*
Transpositional symmetries 241
Triple-excitation energy 28, 29*t*
  component 26

U

United Kingdom, vector processors use 1
Universal Basis Set 28

V

Valence force field 166
van der Waals interactions 147
VAX 11-70
  with attached floating point systems array processor 132
  computer 129
  execution time 132
Vector
  addition, execution time 70*t*
  arithmetic processors 208-210
  coprocessors 235
  FORTRAN system 210–219
  length, effect on ASC 71
  length on machine speed, effects of optimization 72*f*
  mode 89
  parameter file (VPF) 70
  processing 231–234
    library, VPLIB 217*t*–219*t*
  processor(s) (VP) 193
    machine, high speed, for chemical kinetic sensitivity analysis 79–88
    microcomputer system 211*f*
    card for S-100 bus microcomputers 209*f*
    use in quantum chemistry 1–37
    use in United Kingdom 1
  temporary space (VTS) 69
Vectorization 220
  of G-matrix build 14
  of pairwise sum, problems 130
  software 69*t*
  techniques 96–98
Vectorized multispecies diffusion algorithm 99*f*
Vectorized selected asymptotic integration method (VSAIM) 65
  rate equation solver 75
Velocity(ies)
  of atom 182
  binary diffusion 96
  calculation 138

Vibrational
  frequencies 167
  spectra 177
  spectroscopy 166
Viscosity of polymers 140
von Neumann architecture 237
Vorticity, equation 91
Vorticity, relative change 96
VP (*see* Vector processor)
VPLIB
  FORTRAN code for matrix multiplication 213*t*
  library, design criteria 210
  routines for structural chemistry 230–231
  subroutines for evaluating transcendental functions 212–219
  vector processing library 217*t*–219*t*
VSAIM (*see* Vectorized selected asymptotic integration method)

W

Water
  of hydration of triclinic lysozyme crystal 185*f*
  molecule arrangement in crystal 186, 189*f*
  molecule, electron correlation energy of ground state 30*t*
  Stillinger–Lemberg model 128
Wavefunctions, computation 48
Wavefunction diagrams 24

Jacket design by Carol Conway.
Production by Katharine Mintel and Gabriele Glang.

Elements typeset by Service Composition Co., Baltimore, MD.
Printed and bound by R. R. Donnelley & Sons Co., Chicago, IL.

Publication Date: November 6, 1981 | doi: 10.1021/bk-1981-0173.ix001

In Supercomputers in Chemistry; Lykos, P., el al.;
ACS Symposium Series; American Chemical Society: Washington, DC, 1981.