21-3-

CONF·790902

2
1/25/80

SCIENTIFIC COMPUTER INFORMATION EXCHANGE MEETING

SEPTEMBER 12·13·1979 LIVERMORE·CALIFORNIA

THEME: IMPACT OF ADVANCED SYSTEMS ON SCIENTIFIC COMPUTATIONS

PROCEEDINGS

SPONSORED BY:
DEPARTMENT OF ENERGY
OFFICE OF BASIC ENERGY SCIENCES
LAWRENCE LIVERMORE LABORATORY

Printed in the United States of America
Available from
National Technical Information Service
U.S. Department of Commerce
5285 Port Royal Road
Springfield, VA 22161
Price: Printed Copy $; Microfiche **$3.00**

Page Range	Domestic Price	Page Range	Domestic Price
001–025	$ 4.00	326–350	$12.00
026–050	4.50	351–375	12.50
051–075	5.25	376–400	13.00
076–100	6.00	401–425	13.25
101–125	6.50	426–450	14.00
126–150	7.25	451–475	14.50
151–175	8.00	476–500	15.00
176–200	9.00	501–525	15.25
201–225	9.25	526–550	15.50
226–250	9.50	551–575	16.25
251–275	10.75	576–600	16.50
276–300	11.00	601–up	—[1]
301–325	11.75		

[1]Add $2.50 for each additional 100 page increment from 601 pages up.

Work performed under the auspices of the U.S. Department of Energy, under contract No. W-7405-Eng-48.

CONF-790902

SCIENTIFIC COMPUTER INFORMATION EXCHANGE MEETING

SEPTEMBER 12·13·1979
LIVERMORE·CALIFORNIA

THEME: IMPACT OF ADVANCED SYSTEMS ON SCIENTIFIC COMPUTATIONS

PROCEEDINGS

SPONSORED BY:
DEPARTMENT OF ENERGY
OFFICE OF BASIC ENERGY SCIENCES
LAWRENCE LIVERMORE LABORATORY

SCIENTIFIC COMPUTER INFORMATION EXCHANGE MEETING ON IMPACT OF ADVANCED SYSTEMS ON SCIENTIFIC COMPUTATIONS

Chairman:	Sidney Fernbach
Co-Chairman:	Bill Buzbee Mal Kalos Garry Rodrigue
Secretary:	Maylene Wagner
Session Chairmen:	Plasma Simulation - John Killeen Atmospheric Modeling - Joe Knox General Scientific Computation - Bill Buzbee Turbulence/Hydrodynamics - Stuart Patterson

PREFACE

The Scientific Computer Information Exchange Meeting was held at the Lawrence Livermore Laboratory, Livermore, California, on September 12–13, 1979. The theme of the meeting was the Impact of Advanced Systems on Scientific Computations. The meeting was sponsored by the Department of Energy Office of Basic Energy Sciences and the Lawrence Livermore Laboratory. The papers printed in these proceedings have been reproduced from camera-ready manuscripts furnished by the authors. They have not been refereed nor have they been extensively edited.

Advanced computing systems such as the CDC-STAR, CRAY-1, ILLIAC-IV, and TI-ASC have been on the market and used by scientific institutes for several years. Experience has led to the realization that different numerical and software techniques that fit these architectures are required. The papers in this symposium were presented by physical and computational scientists who use advanced computers in their work. The sessions covered plasma simulation, atmospheric modeling, turbulence and hydrodynamics, and topics in general numerical algorithms.

The success of any meeting depends, of course, on the work and support of the many people involved. It is my pleasure to thank Sid Fernbach, symposium chairman, for suggesting the idea for the meeting and for his handling of many of the necessary administrative decisions. The administrative support of Maylene Wagner and Virginia DuBose was invaluable to the smooth running of the meeting.

A personal thanks goes to the DOE sponsor of the symposium, Jim Pool of the Applied Mathematics Division of the Office of Basic Energy Science.

Finally, I am extremely grateful to my co-chairmen Mal Kalos and Bill Buzbee for their advice and cooperation, and to the session chairmen for their solicitation and selection of papers and their skillful handling of the sessions.

Garry Rodrigue

Garry Rodrigue
Coordinating Chairman

THIS PAGE

WAS INTENTIONALLY

LEFT BLANK

CONTENTS

PLASMA SIMULATION

THIS PAGE

WAS INTENTIONALLY

LEFT BLANK

VECTORIZED PIC SIMULATION CODES ON THE CRAY-1

D. W. Forslund and C. W. Nielson
Los Alamos Scientific Laboratory
Los Alamos, New Mexico 87545

and

L. F. Rudsinski
Consultant

ABSTRACT

The PIC simulation code WAVE has been almost completely vectorized for the CRAY-1 and is being routinely used on a production basis. We discuss here the vectorizing techniques for the particle mover and the field solver as well as the I/O routines which result in the code being nearly CPU-bound. The procedure used to vectorize the particle mover is to rewrite it in a series of small loops which then are readily converted by a vectorizer program into special vector macros recognized by the FTN compiler at LASL. This allows selective vectorization of different sections of code to determine the optimal vectorization strategy. As is well known the interpolation technique used in PIC simulation for the fields, charges and currents is not vectorizable. We find that the optimal strategy for the FTN compiler is to keep this interpolation process completely in scalar mode. If we separate the scalar fetch and then vectorize the interpolation, we find a degradation of 30% in speed. On the CRAY-1 we obtain speeds of 5.5 s/particle for a 2-D electrostatic mover, 11.5 s/particle for a 2 1/2-D (with all field quantities) non-relativistic electromagnetic mover and 12.4 s/particle for a 2 1/2-D relativistic electromagnetic mover. We also use a fully vectorized Poisson solve algorithm which uses FFT (Berglund real form) in one direction and tridiagonal solve in the other. Vectorization is achieved in the direction normal to the transform or tridiagonal solve. A 256 x 256 Poisson solve takes 100 ms and a 64 x 64 Poisson solve takes 5.1 ms. The FFT takes 2/3 of the time and the tridiagonal solve takes 1/3 of the time. In order to sustain these speeds for large problems, an efficient I/O algorithm is needed on the CRAY-1. The algorithm we have implemented in production is triple-buffering with two disk channels. Sustained transfer rates of 375,000 words/sec/channel are obtained which allow for nearly complete overlap with the relativistic mover. Exploratory tests have shown that it will be possible to obtain sustained rates of 450,000 words/sec on each of four channels driven simultaneously and overlapped with computation. The size of the required buffers to achieve this is dependent on the details of the operating system.

Dup

REALISTIC 3D RESISTIVE MHD CALCULATIONS ON THE CRAY-1*

H. R. Hicks and B. Carreras†
Oak Ridge National Laboratory
P. O. Box Y, Oak Ridge, Tennessee 37830

ABSTRACT

CPU times for nonlinear resistive MHD calculations are very strongly dependent on the input variables. Although some regions of parameter space can be studied very cheaply, the conditions that correspond to Tokamak plasmas are extremely time consuming to study. Unfortunately, it is not possible to extrapolate reliably from the easy cases to the interesting regime. In recent years we have improved the efficiency of our codes by two orders of magnitude. As a result it is now feasible for us to run some realistic Tokamak cases on a CDC 7600. An additional factor of four reduction in run time is obtained by going to the CRAY-1 where the most time consuming routines are vectorized. This last factor makes a program of systematic studies feasible.

INTRODUCTION

The device which presently holds the most promise for obtaining energy by controlled fusion is the Tokamak. Although there is much that is still not understood about Tokamak confinement, possible explanations of some Tokamak observations have emerged from resistive magnetohydrodynamic (MHD) calculations during the last several years.

The ability of a Tokamak to confine a plasma is limited by the disruptive behavior[1] of the plasma. In the case of major disruptions (total loss of confinement), damage to the device can result from the sudden deposition of energy to the walls. Plasma disruptions are generally accompanied by MHD activity. The observed timescale of the major disruption (about 10 μsec to 1 msec in present Tokamaks) is of the order of the <u>linear</u> tearing mode growth time.[2] Further analytic calculations[3] have shown that the <u>nonlinear</u> growth time of these resistive MHD modes, such as are seen as precursors of the major disruption, is on the order of the plasma skin time, τ_R (a fraction of a second to a few seconds). This result has made it difficult to identify tearing modes as the dynamical mechanism of the major disruption.

All of these analytic results were for large aspect ratio, low β (ratio of the plasma energy to magnetic energy) and single helicity. The low β assumption is valid for present ohmically heated Tokamaks. The single helicity approximation is valid only in cylindrical geometry and only when the plasma equilibrium is linearly unstable with respect to just one helicity.

The earliest numerical calculations[4,5,6,7] also retained all of these restrictions. The low β assumption allows the number of MHD partial differential equations (p.d.e.'s) to be reduced, and, in the single helicity approximation, the calculation can be performed in two dimensions (2D). The resistive MHD p.d.e.'s are solved as an initial value fluid problem on a 2D cylindrical grid. The initial condition corresponds to a plasma equilibrium plus a small perturbation. The single helicity case has yielded a wealth of results including possible mechanisms for internal disruptions,[8,9] Mirnov oscillations[9,10] and disruptions during the initial stage of the Tokamak discharge.[6,11] However, the explanation of the timescale of the major disruption remained elusive.

In nonlinear resistive calculations it is necessary to follow events on two disparate timescales, the aforementioned skin time τ_R, and the poloidal Alfvén time τ_{Hp}. In modern Tokamaks $\tau_R/\tau_{Hp} = S \simeq 10^7$. Computation time increases strongly with S. As a result, numerical efficiency is an important issue if one wishes to consider realistic values of S. Unfortunately, the results of faster calculations (at, say, $S=10^4$) can yield qualitatively different results, making extrapolation in S a questionable strategy.

We have speculated that a possible mechanism for the major disruption involves the interaction of 2 or more linearly unstable modes of different helicities, necessitating solving the problem in three dimensions. This approach has been quite fruitful, since it has revealed that a timescale of the order of the linear growth can re-enter the nonlinear problem[9,12] when it is generalized to 3D. Comparison of our coupled helicity results with experimental observations of the major disruption has been favorable in the few cases where sufficient data exists. Moreover, by modifying the boundary conditions we have determined that a feedback circuit[13] might be successful in preventing major disruptions.

Resistive 3D MHD calculations have also been carried out by Schnack[14]. He has directed his work to reversed field pinches where β is relatively high and S is not as high as in Tokamaks. He has dropped the large aspect ratio and low β assumptions, but retained the single helicity restriction.

THE EQUATIONS

We simplify [5,15,16] the full set of resistive MHD equations by assuming Tokamak ordering and low β. We solve the resulting equations in either a torus or a periodic cylinder. To simplify the presentation we write here the equations in their cylindrical form:

$$\frac{\partial\psi}{\partial t} = \eta\, J_\zeta - E^w_\zeta - \frac{\partial\psi}{\partial r}\frac{1}{r}\frac{\partial\phi}{\partial\theta} + \frac{1}{r}\frac{\partial\psi}{\partial\theta}\frac{\partial\phi}{\partial r} - \frac{\partial\phi}{\partial\zeta},$$

$$\frac{\partial U}{\partial t} - \frac{1}{r}\frac{\partial\phi}{\partial\theta}\frac{\partial U}{\partial r} + \frac{\partial\phi}{\partial r}\frac{1}{r}\frac{\partial U}{\partial\theta}$$

$$= S^2\left\{\frac{1}{r}\frac{\partial\phi}{\partial\theta}\frac{\partial J_\zeta}{\partial r} - \frac{\partial\psi}{\partial r}\frac{1}{r}\frac{\partial J_\zeta}{\partial\theta} - \frac{\partial J_\zeta}{\partial\zeta}\right\},$$

$$U = \vec{\nabla}^2_\perp \phi\ ,\quad J_\zeta = \vec{\nabla}^2_\perp \psi\ .$$

The functions ψ (poloidal flux), U(toroidal component of vorticity), ϕ(velocity stream function) and J_ζ(toroidal current) are scalar functions of all 3 spatial coordinates as well as of time. The cylindrical coordinates are r, θ and ζ where ζ is the coordinate along the cylinder. E^w_ζ and S are time-independent scalars. The resistivity, η, can be treated several ways. Here we shall assume it is only a function of radius and does not vary with time. The numerical problem is to integrate in time the two coupled 3-dimensional p.d.e.'s.

The boundary conditions are appropriate to a rigid cylinder,

$$\phi\ (r_{wall}) = 0,$$

and either constant toroidal current,

$$\frac{\partial\langle\psi\rangle}{\partial r}\ (r_{wall}) = 0,$$

or conducting walls

$$\psi(r_{wall}) = 0.$$

All functions are periodic along the cylinder, e.g.,

$$\psi(\zeta) = \psi(\zeta + 2\pi)\ ,$$

and are regular at the cylindrical axis.

The poloidal flux function, ψ, is initialized to correspond to a plasma equilibrium (which is a function only of radius) plus a small perturbation. The perturbation is normally chosen to approximate the sum of several linear eigenfunctions. Under these conditions, the earliest phase of the calculation is characterized by the exponential growth of the true eigenfunctions. Each eigenfunction is of the form

$$\psi(r,\theta,\zeta) = \psi_{mn}(r)\cos(m\theta + n\zeta)$$

where m (n) is the poloidal (toroidal) mode number. Any part of the initial perturbation which does not project into one of these growing solutions becomes relatively insignificant and can be ignored. At high values of S, this exponential phase is followed by a clear nonlinear phase of slow algebraic growth, in agreement with theoretical predictions.[3] In the 2D approximation either the mode saturates, giving a new non-axisymmetric equilibrium, or, through reconnection, the mode is no longer resonant in the plasma and a new axisymmetric equilibrium is formed. In the 3D case,[9,12] under certain conditions, the growing modes interact nonlinearly generating other MHD modes and accelerating their growth. The number of modes caught up in this process increases rapidly with time. The time scale of this phase is like that in the earlier exponential phase. In agreement with experiment, the toroidal current is severely deformed in a way that the self inductance of the plasma decreases and a negative voltage spike is produced at the plasma edge. The flux surfaces are destroyed over a large region in which the magnetic field wanders stochastically (Fig. 1).

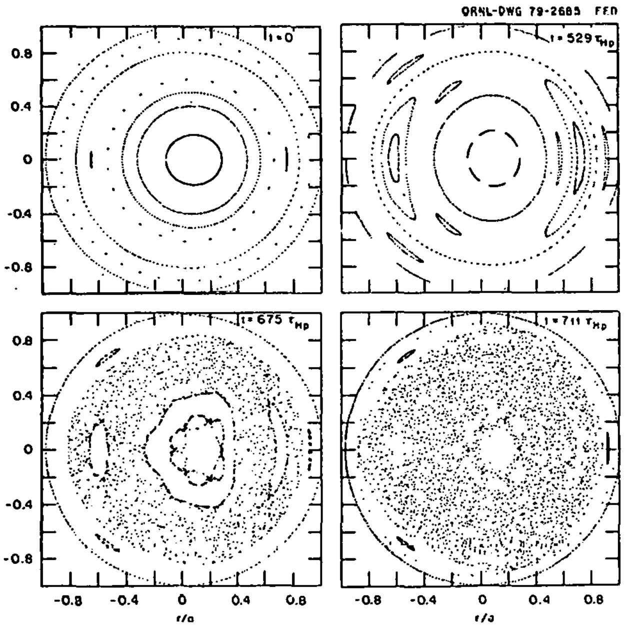


Fig. 1. The magnetic field evolves from a state of well defined flux surfaces to a state with a large volume filled with a stochastic field line.

During this phase there is a transition from large scale phenomena to small scale phenomena. At this point the fluid model starts losing its validity and the calculation should be stopped. Except for this final phase, the calculation is dominated by a small number of modes. It is this feature which leads us to the method RSF, described in the next section.

NUMERICAL METHODS

The technique[7,12,16] we employed in implementing our first 3D computer code, RS3, is the most obvious choice. The functions are represented on a cylindrical grid. Unequal spacing in radius is permitted. Finite difference expressions are used for spatial derivatives in all 3 directions. The equations are advanced in time explicitly except for the ηJ_ζ term which is advanced implicitly.

This formulation has two problems. Execution times, even for rather fast cases, are so long that systematic studies are impossible. This is compounded by the fact that at the singular points, where accuracy is most important, the truncation error can have a significant effect.[17]

Our second formulation, RSF, is designed to eliminate the truncation problem which originates from discretization in the θ and ζ coordinates in RS3. The finite difference approximations in those directions are replaced with a Fourier series expansion of the dynamical functions,[17] e.g.,

$$\psi(r,\theta,\zeta) = \sum_{mn} [\psi^c_{mn}(r)\cos(m\theta + n\zeta) + \psi^s_{mn}\sin(m\theta + n\zeta)] .$$

Except for the end of our multihelicity calculations, each run is dominated by a small number of terms in the series. Therefore the series expansion is actually a much more economical representation. When the calculation enters a stage where small scale structures become important, both methods are inadequate.

The code using the series expansion, RSF, executes about 2 orders of magnitude faster than RS3. The exact factor depends on the problem being solved as well as the accuracy desired. The nonlinear terms take the bulk of the execution time in RSF, due to the convolutions necessary to multiply 2 series expansions together.

At the time we started to develop RSF, installation of a CRAY-1 at the NMFE Computer Center was about 6 months away. As we wrote Fortran code, we used a style which could be largely vectorized by the CRAY Fortran compiler. This was not difficult and in fact probably resulted in more readable (and thus easily modifiable) code. However, the most time consuming operation, the convolution, appeared inherently difficult to vectorize since it involves nonlinear subscripting. For example:

```
      F = 0.
      DO 100 LP=LPMIN,LPMAX
  100 F = F + G(LG(LP))*H(LH(LP))
```

However, since we retain a finite difference grid in the radial direction it is necessary to perform the above calculation at each radial grid point. Putting the loop over grid points on the inside gives the vectorizable code

```
      DO 102 J = 1,JMAX
  102 F(J)=0.
      DO 100 LP=LPMIN,LPMAX
      DO 101 J=1,JMAX
  101 F(J)=F(J)+G(J,LG(LP))*H(J,LH(LP))
  100 CONTINUE
```

In this form the convolution routine gains a factor of 5 in speed over the unvectorized version.

A coupled helicity run can take as little as a few minutes or as much as several hours on the CRAY-1 depending primarily on S and on the accuracy desired. In addition, execution is greatly slowed by inclusion of temperature evolution[18] (factor of 10) and/or toroidal effects[19] (factor of 10). It is a general feature of our coupled helicity runs that the time step drops rapidly as the fast nonlinear effects become important (Fig. 2).

ORNL-DWG 79-2994 FED

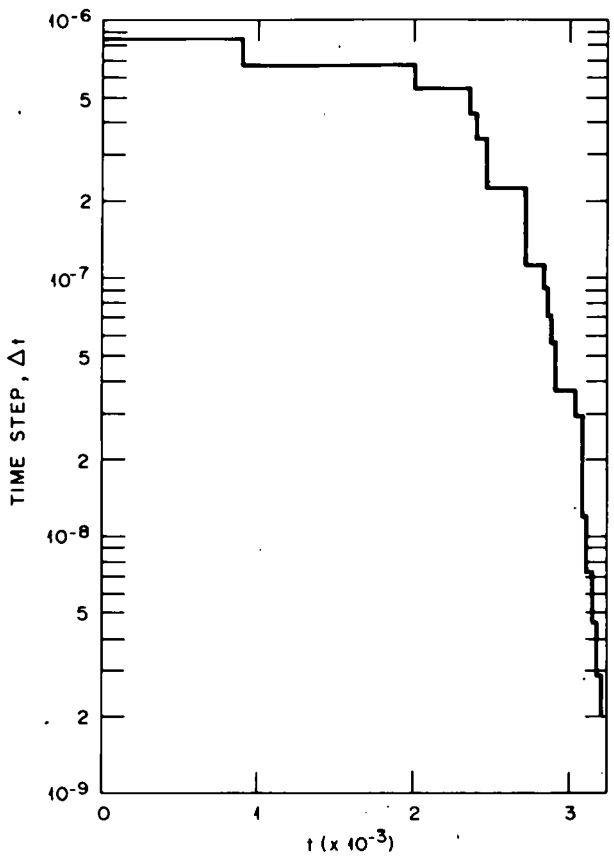


Fig. 2. Finite difference timestep size decreases significantly late in the run when mode coupling dominates. Most of the run time is concentrated at the end of the calculation.

With the convolution routine highly optimized for each machine, execution time on the CRAY-1 is about one fourth as long as on the CDC 7600. This factor has made it possible for us to run the necessary numerical validation runs and to analyze a large number of initial conditions. Without a computer in the CRAY-1 class, a very large part of this work could not have been done.

REFERENCES

*Research sponsored by the Office of Fusion Energy (ETM), U.S. Department of Energy under contract W-7405-eng-26 with the Union Carbide Corporation.

†Visitor from Junta de Energia Nuclear, Madrid, Spain.

1. E. P. Gorbunov, et al., At. Energ. 15, 363 (1963)[Sov. At. Energy 15, 1105 (1963)];
L. A. Artsimovich, et al., At. Energ. 17, 170 (1964)[Sov. At. Energy 17, 886 (1964)];
L. A. Artsimovich, et al., in Plasma Physics and Controlled Nuclear Fusion Research (International Atomic Energy Agency, Vienna, 1971), Vol. I, p. 443;
S. von Goeler, et al., Phys. Rev. Lett. 33, 1201 (1974);
V. S. Vlasenkov, et al., Nucl. Fusion Suppl. 1, 1 (1975);
L. A. Berry, et al., in Plasma Physics and Controlled Nuclear Fusion Research (International Atomic Energy Agency, Vienna, 1977), Vol. I, p. 49;
I. Hutchinson, Phys. Rev. Lett. 37, 388 (1976);
D. B. Albert, et al., Nucl. Fusion 17, 863 (1977);
S. V. Mirnov, et al., in Plasma Physics and Controlled Nuclear Fusion Research (International Atomic Energy Agency, Vienna, 1977), Vol. I, p. 291;
N. R. Sauthoff, et al., Nucl. Fusion 18, 1445 (1978);
K. Toi et al., "Current Density Profile Control by Programming of Gas Puffing and Plasma Current Waveform in the JIPPT-II Tokamak," IPPJ-372, March 1979.

2. H. P. Furth, et al., Phys. Fluids 6, 459 (1963);
H. P. Furth, et al., Phys. Fluids 16, 1054 (1973).

3. P. H. Rutherford, Phys. Fluids 16, 1903 (1973).

4. B. V. Waddell, et al., Nucl. Fusion 16, 528 (1976);
D. Biskamp, et al., in Plasma Physics and Controlled Nuclear Fusion Research (International Atomic Energy Agency, Vienna, 1977), Vol. I, p. 579.

5. M. N. Rosenbluth, et al., Phys. Fluids 19, 1987 (1976).

6. R. B. White, et al., in Plasma Phvsics and Controlled Nuclear Fusion Research (International Atomic Energy Agency, Vienna, 1977), Vol. I, p. 569.

7. B. V. Waddell, et al., in Theoretical and Computational Plasma Physics (International Atomic Energy Agency, Vienna, 1978) p. 79.

8. B. V. Waddell, et al., Nucl. Fusion 18, 735 (1978);
G. L. Jahns, et al., Nucl. Fusion 18, 609 (1978).

9. J. D. Callen, et al., in Plasma Physics and Controlled Nuclear Fusion Research (International Atomic Energy Agency, Vienna, 1979), Vol. I, p. 415.

10. R. B. White, et al., Phys. Fluids 20, 800 (1977);
B. Carreras, et al., "Poloidal Magnetic Field Fluctuations in Tokamaks", Nucl. Fusion (to be published).

11. B. Carreras, et al., Nucl. Fusion 19, 583 (1979).

12. B. V. Waddell, et al., Phys. Rev. Lett. 41, 1386 (1978);
B. V. Waddell, et al., Phys. Fluids 22, 896 (1979).

13. J. A. Holmes et al., "Stabilization of Tearing Modes to Suppress Major Disruptions in Tokamaks," (to be published in Nucl. Fusion);
H. R. Hicks et al., "Stabilization of Tearing Modes: Feedback Stabilization and Profile Tailoring," IAEA Technical Committee Meeting on Disruptive Instabilities, Garching, Germany, February, 1979.

14. D. Schnack, "Nonlinear Numerical Studies of the Tearing Mode," Ph.D. Thesis, Univ. of Calif., Davis, UCRL-52399 (1978);
D. Schnack, J. Killeen, Nucl. Fusion 19, 877 (1979);
D. Schnack, J. Killeen, (to be published in J. Comput. Phys.).

15. H. R. Strauss, Phys. Fluids 19, 134 (1976).

16. H. R. Hicks, et al., "Interaction of Tearing Modes of Different Pitch

16. H. R. Hicks, et al., "Interaction of Tearing Modes of Different Pitch in Cylindrical Geometry," ORNL/TM-6096 (December 1977).

17. H. R. Hicks, et al., "Fourier Transform vs. Finite Difference Techniques in Nonlinear Resistive MHD Codes," 8th Conf. on Numerical Simulation of Plasmas, Monterey, CA, June 1978, CONF-780614.

18. B. Carreras, et al., "Multiple Helicity Tearing Mode Calculations: Major Disruptions," International Atomic Energy Agency Technical Committee Meeting on Disruptive Instabilities, Garching, Germany, February 1979.

19. B. Carreras, et al., "Reduced Set of Resistive MHD Equations in Toroidal Geometry," Sherwood Theory Meeting, Mt. Pocono, PA, April 1979;
H. R. Hicks, et al., "Effects of Toroidicity on the Nonlinear Interaction of Tearing Modes," ibid.

SIMULATION OF GRADIENT-DRIFT STRIATIONS ON THE ASC

B. E. McDonald, S. L. Ossakow, and S. T. Zalesak
Plasma Physics Division
U.S. Naval Research Laboratory
Washington, D. C. 20375

and

N. J. Zabusky
Mathematics Department
University of Pittsburgh
Pittsburgh, Penn. 15260

ABSTRACT

The evolution of many artificial ionospheric plasma clouds is governed by a simple two dimensional model consisting of a continuity equation and a variable coefficient elliptic equation. This type of model applies also to some non-plasma fluid flows. Despite the simplicity of the model, state-of-the-art methods are required to maintain integrity of the solution. These are explained in moderate detail. Our one-level striation code is highly vectorized and achieves in excess of 80% execution efficiency on the ASC. We give typical results and timings for the code and point out areas of current investigations.

INTRODUCTION

Artificial plasma clouds deposited in the ionosphere at altitudes of 140 km or greater are observed to evolve in a matter of minutes from an initially smooth, roughly spherical shape to a state of intense small scale structure.[1] The quasi-final state usually consists of long parallel fingers of ionization when viewed in the plane perpendicular to the earth's magnetic field. The individual fingers or striations point in the direction of the neutral atmospheric motion relative to the bulk ion motion. Theoretical advances during the last twelve years have identified the driving mechanism for this structure as the gradient drift instability.[2-4] Numerical simulation capabilities for the nonlinear equations of motion have existed for approximately six years.[5-7] However some very recent advances in numerical solution methods[8,11] and computing hardware have greatly improved the simulations.

EQUATIONS OF MOTION FOR THE ONE LEVEL MODEL

For many cases of interest, an artificially produced plasma cloud will possess a field-line integrated cross-field conductivity which is scalar and which dominates that of the background plasma. In these cases the electrical currents are largely confined within the cloud, alleviating the need for modeling remote regions. The high ion mobility along the field two-dimensionalizes the cloud to a high degree. As a result, the cloud's evolution is concisely described by the following set of equations in the plane (x,y) perpendicular to the magnetic field:[4]

$$\frac{\partial \rho}{\partial t} = \frac{c}{B} \nabla \cdot \rho \nabla \phi \times \hat{z} + \nabla \cdot K(\rho) \nabla \rho \qquad (1)$$

$$\nabla \cdot \rho \nabla \phi = \underset{\sim}{E}_o \cdot \nabla \rho \qquad (2)$$

In (1) and (2) ρ, c, B, and ϕ are respectively the ion number density, the speed of light, the constant geomagnetic field strength (with $\underset{\sim}{B} = B\hat{z}$), and the cloud-induced electrostatic potential.

K and $\underset{\sim}{E}_o$ are the cross-field plasma diffusion coefficient and ambient electric field. We choose coordinates such that $\underline{E}_o = E_o \hat{y}$ = constant. Equation (1) is the plasma continuity equation cast in coordinates moving with the ambient plasma drift at a velocity $\underline{V}_o = c\underline{E}_o \times \underline{B}/B^2$ relative to the neutral atmosphere. The diffusion coefficient K(ρ) is included to account for single particle collisions between electrons and ions, and between charged particles and the neutral atmosphere. Although K depends upon parameters of the neutral atmosphere and the ion cloud,[12] we shall only note that typically

$$K \simeq 10^4 \ \mathrm{cm}^2/\mathrm{sec} \qquad (3)$$

This model assumes a homogeneous neutral atmosphere which is unaffected by the plasma cloud.

The boundary condition for the elliptic equation (2) is that ϕ should vanish at great distances from the cloud. In practice boundary conditions must be imposed at a finite distance from the cloud. We have chosen a Neumann condition ($\partial/\partial x = 0$) in the direction of the neutral flow (the x direction by arbitrary choice) to allow plasma to flow through the boundaries. In the transverse direction (y) we impose periodicity, which would be a reasonable approximation for simulation of a small portion of a large striated region. For a single isolated cloud the dipole nature of the source term keeps image effects small. This has been verified by tests in which the symmetry of the images is reversed by imposing a Neumann condition in y. Only minor differences were noticed.

Equation (1) is integrated as an initial value problem in time beginning with some specified distribution ρ(x,y) at t = 0. The spatial boundary conditions for ρ are the same as for ϕ. It is perhaps worth pointing out that eqs. (1) and (2) with minor modifications are descriptive of two dimensional incompressible flow problems not normally associated with plasma physics. One example is vorticity transport, in which ρ would play the role of the vorticity and ϕ the streamfunction. Another example is stratified porous medium flow, in which ρ would be the mass density and ϕ the streamfunction.

NUMERICAL METHODS

All work presented here has been performed using NRL's two-pipe Texas Instruments ASC. The simulation code is written entirely in IBM-compatible Fortran (except for parameter statements). The primary impact of the ASC's configuration upon our simulations has been to select out one particular method as being more efficient by at least a factor of two[11] than other methods currently available for solution of the elliptic equation (2). Thus the discussion in this section is primarily devoted to the solution of (2).

ELLIPTIC EQUATION

As is the case for many models requiring the solution of a variable coefficient elliptic equation, our one level gradient-drift code spends most of its time (60% or more) solving eq. (2). Of the solution methods currently available, the most efficient method for the ASC appears to be the Chebychev semi-iterative method.[10,11,13] This choice results from the following: (1) we have a good approximation to ϕ (by extrapolation from two previous timesteps in our case); (2) the method vectorizes completely on the interior of a multidimensional domain; and (3) good convergence can be attained with 32 bit precision in some cases where conjugate gradient methods require 64 bit precision. We precondition (2) by multiplying both sides by $\bar{\rho}^{-1}$, where $\bar{\rho}$ is a five point average of ρ to be defined later. This results in a matrix whose condition is not substantially different from that of the matrix for the discretized Poisson equation. Although the Chebychev method extends to non self-adjoint matrices,[10,11] the pre-conditioned equation (2) is self adjoint for the inner product

$$(A,B) = \sum_{i,j} A_{i,j}\, \bar{\rho}_{i,j}\, B_{i,j} \qquad (4)$$

and suitable boundary conditions. In (4) i and j are spatial indices discretizing x and y respectively.

In the results to be presented here, a uniform spatial grid of 162 by 82 points was used, with the exterior points being guard cells for boundary condition purposes. Second order centered differences were used to represent (2). Since ρ and ϕ are stored on the same spatial mesh, the ρ values appropriate for use on the left side of (2) are spatial averages between two points used to evaluate ϕ derivatives. For computational efficiency we use the following preconditioning divisor:

$$\bar{\rho}_{i,j} = [\,(2\rho_{i,j} + \rho_{i+1,j} + \rho_{i-1,j})/\delta x^2 + (2\rho_{i,j} + \rho_{i,j+1} + \rho_{i,j-1})/\delta y^2\,] \;/(\,4/\delta x^2 + 4/\delta y^2) \qquad (5)$$

Thus the preconditioned representation of (2) is

$$L\,\phi_{i,j} = S_{i,j}, \qquad (6)$$

where L is the five point operator

$$\begin{aligned} L\,\phi_{i,j} = {} & (2\bar{\rho}_{i,j}\,\delta x^2)^{-1} \\ & \times ((\rho_{i+1,j}+\rho_{i,j})(\phi_{i+1,j}-\phi_{i,j}) \\ & - (\rho_{i,j} + \rho_{i-1,j})(\phi_{i,j}-\phi_{i-1,j})) \qquad (7) \\ & + (2\bar{\rho}_{o,j}\,\delta y^2)^{-1} \\ & \times((\rho_{i,j+1}+\rho_{i,j})(\phi_{i,j+1}-\phi_{i,j}) \\ & -(\rho_{i,j} + \rho_{i,j-1})(\phi_{i,j}-\phi_{i,j-1})) \end{aligned}$$

and the driving term is

$$S_{i,j} = E_o(2\bar{\rho}_{i,j}\,\delta y)^{-1}\,(\rho_{i,j+1}-\rho_{i,j-1}). \qquad (8)$$

The choice of $\bar{\rho}$ in (5) results in the matrix L having constant diagonal coefficient

$$L_{diag} = -2/\delta x^2 - 2/\delta y^2 \qquad (9)$$

Preconditioning with $\bar{\rho}$ also yields an upperbound for the eigenvalues of L:

$$\begin{aligned} |\lambda|_{max} &\le \max_i \sum_j |L_{ij}| \qquad (10) \\ &= 4/\delta x^2 + 4/\delta y^2, \end{aligned}$$

which is identical to the maximum eigenvalue for Poisson's equation.

The Chebychev semi-iterative method produces a sequence of iterates ϕ^n converging to the solution of (6) by the following algorithm:

Let ϕ^o = trial solution (11)

$$\phi^1 = \phi^o - b^{-1}\,(L\,\phi^o - S) \qquad (12)$$

and $\phi^{n+1} = F_n((L-b)\,\phi^n - S) - G_n\phi^{n-1},$ (13)

$n > 0.$

The updates ϕ^{n+1} are defined on <u>interior</u> mesh points from (13). Values on exterior guard cells are then set to satisfy boundary conditions. The constants b, F_n and G_n are calculated as follows. The error vector $L\,\phi^n - S$ can be expressed as a linear combination of eigenvectors of L. The Chebychev method guarantees convergence for all eigenvector components of the error whose eigenvalues λ fall in the range

$$b+a \le \lambda \le b-a, \qquad (14)$$

where

$$b/a > 1, \qquad (15)$$

providing

$$F_n = \frac{2\,T_n(q)}{T_{n+1}(q)\,|a|}, \text{ and} \qquad (16)$$

$$G_n = \frac{T_{n-1}(q)}{T_{n+1}(q)}, \qquad (17)$$

where

$$q = -b/a \qquad (18)$$

In (14) we have taken a and b to be negative since the operator L is nonpositive for periodic - Neumann boundaries. In (16) and (17) T_n is the Chebychev polynomial of degree n, which is calculated from the recursion formulae

$$T_o(x) = 1$$

$$T_1(x) = x \qquad (19)$$

$$T_{n+1}(x) = 2x\, T_n(x) - T_{n-1}(x),\ n > 0$$

For this application we have chosen

$$b = L_{diag} = -2/\delta x^2 - 2/\delta y^2 \qquad (20)$$

and

$$a = b + \epsilon\,. \qquad (21)$$

The asymptotic convergence rate is then $(-2\epsilon/b)^{\frac{1}{2}}$. The choice of ϵ will be discussed below. We have chosen (20) for b for computational efficiency. This value of b results in the matrix L-b in (13) having zero diagonal. This reduces the basic operation count by approximately 16%, and in addition allows a "hopscotch" calculation in which ϕ^{n+1} is updated only on alternate gridpoints. This would effectively double the computational speed of the method. However, the results presented here have not taken advantage of the "hopscotch" feature.

ACCELERATION OF CONVERGENCE BY REGRIDDING

If we were to attempt to cover the entire eigenvalue spectrum of L on a single grid, we should take ϵ in (21) to be minus the eigenvalue of least nonzero magnitude. We estimate ϵ roughly by taking $L \approx \nabla^2$. Then for Neumann-periodic boundaries on a mesh of N_x by N_y interior gridpoints we take

$$\epsilon = \mathrm{Min}\left(\frac{\pi}{N_x \delta x}, \frac{2\pi}{N_y \delta y}\right)^2 \qquad (22)$$

For the results presented here, $(N_x,N_y) = (160,80)$, and $\delta x = \delta y$, so the asymptotic convergence rate per iteration is

$$C = 2^{-\frac{1}{2}}\, \pi/N_x \qquad (23)$$

Thus the number of operations required to attain a given error reduction is proportional to $N_x^2 N_y$. For the present case, the total operation count can be reduced by a factor in the approximate range of 4 to 7 by solving for the long wavelength components on a reduced grid of $(n_x,n_y) = (N_x/4, N_y/4)$ points.[11]

For the results presented here, an initial approximation to ϕ was taken from linear extrapolation from the two previous timesteps. The residual error $R = L\phi - S$ was extracted on the fine grid and placed on a coarse grid by taking block averages of 16 fine grid point values. The same averaging was used to obtain ρ values on the coarse mesh. A correction term ϕ_c was initialized to zero on the coarse grid and improved by 72 iterations on the equation $L_c\phi_c = R_c$. (subscripts c denote coarse grid quantities.) These iterations used parameters a,b, and ϵ from (20) - (22) with δx and N_x replaced by $4\delta x$ and $N_x/4$, respectively. Then ϕ_c was defined on the fine grid by linear interpolation and subtracted from the trial solution. Twelve follow-up iterations were then performed on the fine mesh with $\epsilon = -b/30$. This procedure resulted in typical rms residual errors (normalized to the rms source term) of a few parts in 10^4.

CONTINUITY EQUATION

We integrate eq. (1) forward in time by the method of flux correction,[8,9] which combines a fourth order spatial, second order temporal leapfrog scheme with a first order donor cell method. The resulting hybrid scheme maintains high order accuracy in regions where ρ is smooth and monotonic, but reverts to low order where ρ is sharply structured. Effectively it is a prescription for adding diffusion in a systematic, localized way to prevent generation spurious oscillations or anomalous extrema in ρ, which could have disastrous consequences for the electrostatic potential ϕ. (Note that (2) can be written as a modified Poisson's equation, with polarization charges proportional to $\nabla\rho/\rho$.)

In one dimension, the method of flux correction may be represented as follows:

$$\rho_i^* - \rho_i^n = f_{i-\frac{1}{2}} - f_{i+\frac{1}{2}} \qquad (24)$$

$$\rho_i^{n+1} - \rho_i^* = \alpha_{i-\frac{1}{2}}\, \delta f_{i-\frac{1}{2}} - \alpha_{i+\frac{1}{2}}\, \delta f_{i+\frac{1}{2}} \quad (25)$$

The superscripts and subscripts are respectively temporal and spatial indices. In (24), f is a flux derivable from a low order representation of (1) (e.g. donor cell) which does not generate spurious extrema. The flux correction δf in (25) is such that f + δf is identically the flux derivable from a suitable high order scheme (e.g. fourth order leapfrog-trapezoidal for the present case.) All α in (25) satisfy $0 \le \alpha \le 1$. One method of determining the α is to require for each point i that neighboring flux changes δf tending to increase ρ_i be limited multiplicatively so as to prevent a new maximum in ρ from being generated. Then neighboring flux changes tending to decrease ρ_i are limited to preclude generation of a new minimum. This procedure extends straightforwardly to two dimensions[8,9] without time-step splitting.

Let us consider (1) as a special case of the equation

$$\frac{\partial \rho}{\partial t} = \nabla \cdot \rho \underline{V}, \quad (26)$$

where $\underline{V}(x,y) = (u\,(x,y),\ v\,(x,y))$. The low order donor cell fluxes in (24) are obtained as follows. Let

$$\overline{u}_{i+\frac{1}{2},j} = (u_{i,j} + u_{i+1,j})/2. \quad (27)$$

Then

$$fx_{i+\frac{1}{2},j} = \frac{\delta t}{\delta x}\,(\rho_{i,j}\ \mathrm{Max}(\overline{u}_{i+\frac{1}{2},j},\ 0) + \rho_{i+1,j}\mathrm{Min}(\overline{u}_{i+\frac{1}{2},j},0)) \quad (28)$$

is a vectorizable expression for the x component of the low order flux $\underline{f}$ = (fx, fy). Equations (27) and (28) extend straightforwardly to the y component.

For the high order representation of (1) we have chosen the spatially fourth order leapfrog-trapezoidal scheme.[14] This scheme controls the computational mode by appending a trapezoidal integration step to the familiar leapfrog scheme. Denoting fourth order spatial differences with coefficients(1/12,-2/3,0, 2/3,-1/12)by D_x and D_y, define

$$\rho^{n+\frac{1}{2}} = \tfrac{1}{2}(\rho^n + \rho^{n-1}) - (\frac{\delta t}{\delta x} D_x + \frac{\delta t}{\delta y} D_y)\ \rho^n\, V^n. \quad (29)$$

Then the spatially fourth order update would be

$$\rho4_{i,j}^{n+1} - \rho_{i,j}^{n} = -Fx_{i+\frac{1}{2},j} + Fx_{i-\frac{1}{2},j}\ , \quad -Fy_{i,j+\frac{1}{2}} + Fy_{i,j-\frac{1}{2}} \quad (30)$$

where the high order fluxes are

$$Fx_{i+\frac{1}{2},j} = 1/3\ T_{i+\frac{1}{2},j} - 1/24\ (T_{i+3/2,j} + T_{i-\frac{1}{2},j}), \quad (31)$$

and

$$T_{i+\frac{1}{2},j} = \frac{\delta t}{2\delta x}\,(\rho^{n+\frac{1}{2}}_{i+i,j}\ V^n_{i+1,j} + \rho^{n+\frac{1}{2}}_{i,j}\ V^n_{i,j}). \quad (32)$$

In (30) ρ4 is never computed, but serves only to define the fluxes Fx and Fy. Equations (31) and (32) extend straightforwardly to the y fluxes Fy. Having defined high and low order fluxes, we define flux corrections

$$\delta fx_{i+\frac{1}{2},j} = Fx_{i+\frac{1}{2},j} - fx_{i+\frac{1}{2},j} \quad (33)$$

$$\delta fy_{i,j+\frac{1}{2}} = Fy_{i,j+\frac{1}{2}} - fy_{i,j+\frac{1}{2}}$$

for use in the two dimensional analog of (25).

The flux limiting procedure (choice of α in (25)) is adequately discussed elsewhere[8,9] and will not be detailed here. Decisions necessary for the limiting process require no scalar logic on the ASC. They are made by proper use of the vector functions AMIN1, AMAX1, and SIGN.

OPERATION COUNT AND RUNNING TIME

To illustrate the execution efficiency of the code, we will compare actual running speeds with theoretical maxima obtained from bare operation counts. The operation counts are converted to machine cycles per gridpoint per timestep through use of the values in Table 1.

Table 1. Machine cycles required per result for seven basic functions on the ASC in 32 bit vector mode. Store and fetch times are included.

Operation	Cycles per operation
+,-,*	1
Aminl, Amaxl	1
Sign	4
/	8

In Table 2 are presented theoretical and actual timings for the run from which results are presented below. The first column gives operation count converted to cycles per timestep per interior gridpoint for the main parts of the code. The second column gives theoretical running times obtained by multiplying the first column by 1943 timesteps x 160 x 80 interior gridpoints x 40 nanoseconds per cycle. (The NRL machine has a cycle time of 80 nsec; however there are two pipes delivering results simultaneously.) The third column in Table 2 are the actual running times of each portion of the code. The results in Table 2 are typical of all our experience with the code to date. (The contour plots to be presented below are out to 160 seconds, although the full 1943 timestep run went to 240 seconds.)

Table 2. Operation counts, theoretical and actual running time (seconds) for major parts of the code.

	Cycles per point	Theoretical cpu time	Actual cpu time
Driver	26	25.9	40.3
Continuity eq.	142	141.3	154.3
Elliptic eq.	288	286.5	352.1
	456	453.7	546.7

From Table 2 we see that the code executes at 83% efficiency (without machine coding). The operation counts in Table 2 apply only to operations executed on the full grid (162 x 82) or the interior grid (160 x 80). No allowance has been made for boundary point calculation or other operations which do not apply to a fully two dimensional set of points. Thus the true efficiency of the code is somewhat higher than 83% since the actual cpu times include all operations.

TYPICAL RESULTS

Figures 1-4 give contours of equal plasma density at four times following the release of a 1km-radius barium cloud in the ionosphere at an altitude of 190km. The barium plasma (ionized by solar radiation) is subjected to a neutral atmospheric drift of 100 m/sec to the right. The geomagnetic field is directed perpendicularly out of the plane. The grid interval for this calculation was 30 meters, so the dimensions of the rectangular integration volume are 4.8 by 2.4 kilometers. The initial condition is

$$\frac{\rho(x,y)}{\rho_o} = (1 + 4e^{-(x^2+y^2)/R^2})(1+\epsilon(x,y)),\quad(34)$$

where R = 1 km and $\epsilon(x,y)$ is a random phase perturbation of rms amplitude 0.03. The contour values are evenly spaced from 1.0 to 4.5.

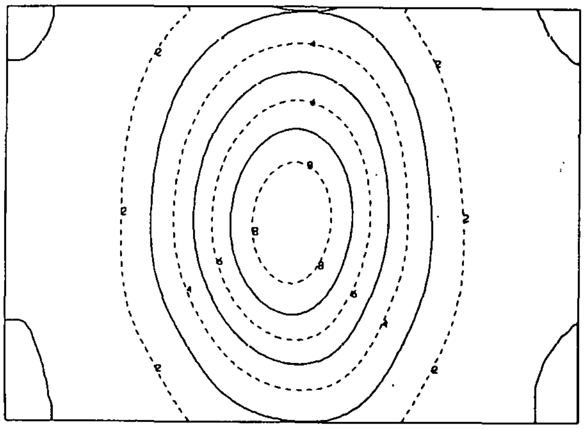


Fig. 1. Plasma density contours at 0 sec. The horizontal scale has been compressed by 30%

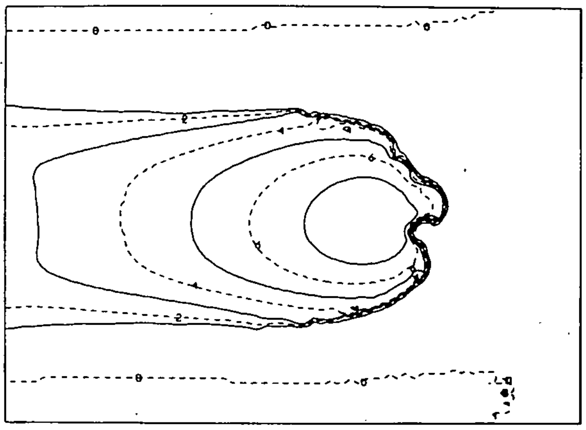


Fig. 2 Plasma density contours at 80 sec.

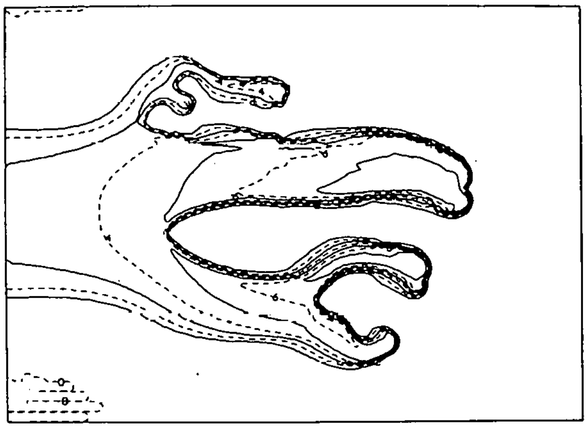


Fig. 3 Plasma density contours at 120 sec.

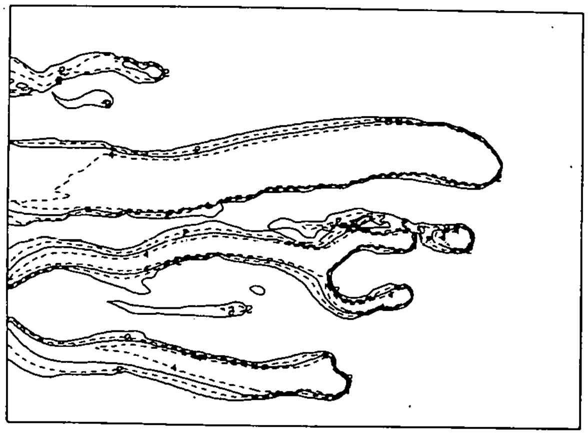


Figure 4 Plasma density contours at 160 sec.

Three areas of current investigation will receive brief description here. All are oriented toward gaining knowledge concerning the spatial and spectral properties of the "well developed" state. First, we are interested in the degree to which diffusion, a linear process, determines minimum scale sizes. This has been discussed elsewhere.[12] We are also interested in nonlinear saturation mechanisms which determine the point of break-away from linear growth and help determine the well developed state (including its power spectrum).[15] The last area we will mention here concerns an apparently anomalous mechanism which causes some observed barium clouds to "freeze up" after conforming to the model (1) - (?) for some ten to twenty growth times. It may be possible to test various candidate mechanisms within the framework of our model by invoking anomalous transport coefficients or altered conductivities.

ACKNOWLEDGMENT

This work was supported by the Defense Nuclear Agency.

REFERENCES

1. T. N. Davis, G. J. Romick, E. M. Westcott, R. A. Jeffries, D. M. Kerr, and H. M. Peek Planet. Space Sci., 22, 67 (1974).

2. G. Haerendel, R. Lust, and E. Reiger, Planet. Space Sci., 15, 1 (1967).

3. L. M. Linson and J. B. Workman, J. Geophys. Res., 75, 3211 (1970).

4. F. W. Perkins, N. J. Zabusky, and J. H. Doles III, J. Geophys. Res., 78, 697 (1973).

5. F. W. Perkins, N. J. Zabusky, and J. H. Doles III, J. Geophys. Res. 78, 711 (1973).

6. F. W. Perkins, N. J. Zabusky, and J. H. Doles III, J. Geophys. Res. 81, 5987 (1976).

7. A. J. Scannapieco, S. L. Ossakow, D. L. Book, B. E. McDonald, and S. R. Goldman, J. Geophys. Res. 79, 2913 (1974).

8. S. T. Zalesak, NRL Memo Report 3716 (1978).

9. S. T. Zalesak, J. Comp. Phys., 31, 335 (1979).

10. B. E. McDonald, NRL Memo Report 3541 (1977).

11. B. E. McDonald, J. Comp. Phys. in press.

12. B. E. McDonald, Proc. Ionospheric Effects Symp., ed. J. M. Goodman, Naval Research Lab. (1978).

13. R. S. Varga, "Matrix Iterative Analysis," Prentice-Hall, Englewood Cliffs, N.J. (1962).

14. A. Grammeltvedt, Monthly Weather Rev., 97, 384 (1969).

15. P. K. Chaturvedi and S. L. Ossakow, J. Geophys. Res. 84, 419 (1979).

NUMERICAL EXPERIMENTS IN THE DYNAMICS OF GALAXIES ON ILLIAC IV

R.H. Miller
University of Chicago
Dept. of Astronomy and Astrophysics
1100-14 East 58th Street
Chicago, IL 60637

B.F. Smith
NASA-Ames Research Center
Theoretical and Planetary Studies Branch
Mail Stop 245-3
Moffett Field, CA 94035

ABSTRACT

Fully three-dimensional n-body integrations that run on ILLIAC IV have been in routine use for the past three years in a variety of astronomical studies in the dynamics of galaxies. Some 10^5 particles move self-consistently under forces of Newtonian gravitation. The particles themselves are the source of the gravitational forces. The dynamical problem lends itself to a structure that fits the ILLIAC IV architecture very naturally; it should fit other array processors as easily. The complexity of results, coupled with their frequently unexpected nature, make graphic methods the only feasible way to study the results of a calculation. The programs and the astronomical problems studied will be briefly described and motion pictures generated in some runs will be shown. The motion pictures were produced as our usual method of studying the computational results.

I. SCIENTIFIC MOTIVATION AND PROGRAM DEFINITION

A galaxy can be seen from only one side; we cannot go around to look at it from the other side. We cannot kick it to see if it bounces. Galaxies develop so slowly (fast changes occur in 10^8 years) that no visible changes of shape occur in human lifetime. Yet galaxies are known to involve a complicated interplay of many physical effects, and the processes that determine the behavior of them are highly nonlinear. These features necessitate some kind of check on theoretical speculations. Numerical experiments are the only substitute available to replace laboratory experiments. The analogy between numerical experiment and laboratory experiments may be pressed further--program checks correspond to equipment checks and systematic errors can be induced by numerical effects or by leaving out some essential physics. Notwithstanding these problems, the study of galaxies is peculiarly satisfying because the real objects we study are so beautiful and many of the objects created in the computer are beautiful as well. There is even a thrill of sorts to watch these systems develop.

The self-consistent response of a system of particles to forces of Newtonian self-gravitation underlies a variety of astronomical problems ranging from planetary formation in the solar system through clusters of stars to galaxies and clusters of galaxies. While some of these systems display striking symmetries, most lack exact symmetry and some are quite irregular. A program to study these objects cannot presuppose any symmetry, especially if we are to understand the reasons why some objects develop and maintain their beautiful symmetries. A fully three-dimensional program is required.

Galaxies are self-consistent self-gravitating collections of stars that are held together by gravitational forces and that are prevented from collapsing by angular momentum and by random velocities (which behave in some respect like a hydrodynamic pressure). Structure, form, stability, and evolution of galaxies must be studied as a dynamical problem that is best attacked by seeing what kinds of steady-state solutions are preferred.

Two timescales show up in conventional analyses of the dynamical problem. We imagine the actual force field analyzed into a smooth field and a fluctuating part. The dynamical timescale is defined by the orbital period of typical particles moving in the smooth field. The fluctuating part arises principally from two-body interactions. The time scale over which the fluctuating part causes a particle to multiple-scatter away from the "unperturbed" orbit in the smooth field is the two-body relaxation time, and usually is simply called the relaxation time. It is the second timescale. Relaxation times in galaxies are typically around 10^{13} - 10^{15} years, as much as 1000 times the Hubble time or the age of the universe. The important processes in the dynamics of galaxies proceed on a dynamical timescale, $(G\rho)^{-\frac{1}{2}}$.

The number of particles needed in an n-body calculation is determined by the need to assure a clear separation of dynamical and relaxation timescales. The two timescales are nearly the same for particle numbers under 1000 and the ratio of the timescales is logarithmic in the number of particles. Because of the weak dependence on particle number, n-body calculations need 50,000 - 100,000 particles to ensure that two-body relaxation effects may safely be ignored. Even at 10^4 particles, the slower relaxation timescale cannot yet be ignored. Experimental checks verify that calculations with 10^5 particles are on safe ground.

The parameters of a practical calculation to study galaxies are determined by physical considerations. Calculations designed without sufficient attention to these matters are difficult to interpret and often yield misleading results.

Problems in the dynamics of galaxies that are to be attacked by n-body integrations are formulated as initial value problems. Initial conditions are unknown for true astronomical problems, and a major puzzle is to determine what kinds of initial conditions might have led to the variety of galaxies we see today. A guiding principle is that, because galaxies are more similar to each other than initial conditions could have been, final forms must be more or less independent of details of initial conditions. A second group of problems can be set up by selecting models like actual galaxies as inital conditions. Stability, shapes, and internal dynamics can be studied this way. Stable systems remain similar to the initial galaxy while unstable systems change on a dynamic timescale. Here again, a guiding principle is that real galaxies must be straightforward to mimic; they cannot require improbable or difficult-to-set-up initial conditions.

A wide variety of problems can be attacked with the code described here. Questions of the stability of galaxy models leads to studies of actual three-dimensional shapes on the one hand and to the suggestion that galaxies are actually much more massive than had been thought on the other. If so, there may be enough mass in galaxies or in clusters of galaxies to close up the universe. Mass density estimates for the universe fall short of closure by a factor 30 - 100 unless there is some unseen mass to make up the difference; massive dark halos around galaxies provide one of the more attractive possibilities. The suggestion that galaxies must have massive halos came from earlier stability studies based on n-body integrations in which the initial conditions were chosen to represent actual galaxies. A list of problems studied or under attack is included in Sec. IV. The calculation is very similar to plasma simulation codes.

II. PROGRAM DESIGN

Programs of the type described are best designed around data structures that represent the state of the problems. The same physical problem, translated into a program to run on a different processor, should be programmed differently to make use of a data structure that leads to the best program design. That data structure is dependent on the architecture of the computer on which it is to run. We estimate that about 70% of the design effort for a new large program goes into the design of the data structures. We make no claim that the program designed for ILLIAC IV could be rewritten to run decently on other computers without major redesign of the data structures. The programs described here were designed 4½ years ago, when ILLIAC IV was the only large processor available to us. It works nicely on ILLIAC.

Program design is incomplete without provision of a method to analyze the results of computations and methods for handling archival storage of output. These programs were designed around the use of motion pictures for both purposes. We return to this point in the next section.

The three-dimensional n-body program constructed for ILLIAC IV is designed to be fairly general so it can be used for a variety of problems of astronomical interest. It can handle about 10^6 particles within a cubic volume, but it is usually used with about 10^5 particles. Forces are computed in a manner that allows details down to 1/64 of the linear dimension of the configuration space to be resolved. Long-range effects are correctly handled by the force calculation.

Each particle is represented by one ILLIAC word. In addition to the configuration coordinates and velocities, 10 bits are allowed for other attributes. These attributes may be defined in any way necessary for the problem of interest. For example,one bit can designate whether the particle represents a star or a gas cloud. Other bits can be used to measure heavy element buildup in the gas or stars. This use of particle attributes leads to interesting and useful simulations to study the evolution of stellar populations in an evolving galaxy, with evident consequences for studies in nucleosynthesis.

On the computational side, the program designs are pleasing in the way they fit the ILLIAC IV architecture and utilize the parallel features of ILLIAC. It is likely that they would fit other architectures as easily; the transition to less restrictive array processors should be straightforward. The program needs a machine like ILLIAC or a modern array processor--the sheer size of the program demands a large machine and the property of the calculation to admit parallel processing makes it amenable to treatment on ILLIAC. With graphics support, this program produces useful scientific results that also appeal to the lay public. This program can give the public an example of what scientific research is all about and can help convey some of the excitement of scientific research.

Like most large n-body programs, that designed for ILLIAC IV consists of two principal parts: the potential solver, or subroutine in which the forces are calculated, and the particle-pusher, or subroutine in which the particle velocities and positions are advanced according to the forces. The potential solver makes use of densities (or the projection of the particle phase space density onto the configuration space), which are tabulated by the particle-pusher as the new velocities and positions are computed. Output summaries and generation of plot files are handled by other subroutines, as is the establishment of starting conditions.

Two-dimensional calculations can be core-contained; this is no longer possible for reasonably-sized three-dimensional calculations. Force-values and density counters must be in core along with the particle data during the particle-pushing part of the calculation. There is no need to have all the particle data available at once; data that refer to one particle can be processed to completion without reference to other particles. Particle interactions take place through the force-field. But there is not even enough room for the entire force field or density count in three dimensions.

One way to handle a three-dimensional calculation is to retain only a portion of the force field and of the density count in core at any one time. All of the particles whose configuration coordinates are within a limited region are processed before moving on to another region. Potential values and density counters are also available for this limited region. Particle data may be handled in any of several different ways. If large data file I/O is cheap, several sweeps may be made through files of particle data, selecting those particles whose coordinates match the other data in core at each pass. An alternative is to pre-arrange particle data before writing output files, to avoid need for several passes. This is the method adopted in the ILLIAC IV program because disk I/O is expensive. The program is constructed around a pattern for partitioning the data which fits the ILLIAC IV architecture naturally, Matching portions of the particle data, of the force field, and of the density count are in memory at the same time.

The cubic volume occupied by the system of particles is divided into 64 subdivisions along each edge; the force calculation returns values at the center of each subdivision. Force values at points other than the center are found by linear interpolation. Densities are determined from the count of particles in each subdivision (NGP assignment is used, but it is not required by program design). The cartesian coordinates are designated as follows: The 64 subdivisions along the x-coordinate are each assigned to one of the 64 processing elements (PE's) of ILLIAC IV; each PE handles particles whose x-coordinate is equal to the PE number. All 64 subdivisions of the y-coordinate and 8 of the 64 subdivisions of the z-coordinate are in PE memory at the same time. Eight such loadings are required to process the entire set of z-values, and thus to process the entire accessible configuration space. With nonuniform particle distributions in the configuration space, unequal numbers of particles are in volumes represented by different PE's; this leads to some unavoidable inefficiency because some PE's may have completed processing their load of particles while others still have particles left to work on. Particle data must be reassigned to the correct PE after they have moved in an integration step, and they must be placed on disk so they will be brought back in the proper band of z-values on the next integration step. This feature, along with the need for frequent access to backup storage on the ILLIAC disk, is the principal complication in the design of the particle-pushing programs. Programming is made more difficult by tight memory and by the lack of an operating system to assist with disk I/O operations on ILLIAC.

A) THE POTENTIAL CALCULATION

The potential calculation is designed to generate values of the potential at the center of each subdivision from the new density data. Potentials go to zero at infinite distance, the boundary condition that is appropriate for the gravitational problem. The calculation proceeds through Fourier transformations and the convolution theorem. This method of computing forces is necessary to avoid the n^2-dependence of exact force calculations on particle number. Correct values of the potential are returned at the tabulation-points in the center of each cell. Linearly interpolated forces are obtained in the particle-pusher by interpolation on the potential values.

Fourier transformations calculated in a computer are discrete, and represent periodic functions. The potentials generated represent periodically replicated density distributions. They can be made to represent the gravitational potential of an isolated system by doubling the period in each spatial coordinate, with a modification of the dependence on distance of the potential around a single particle (the potential must go to zero beyond double the initial interval). This is the procedure followed in the ILLIAC program.

The potential calculation utilizes the architecture and parallelism of ILLIAC IV fully. All PE's are on all the time. Some 1552 words of the 2048-word PE memory are used for data manipulation in the potential calculation.

B) THE PARTICLE-PUSHER

The part of the program that handles the actual integration (or particle-pushing) is modular for readability, to simplify debugging, and to facilitate program development. As much as possible, the overall supervision, the portions that relate to moving particle data data to and from the ILLIAC disk, and the portions in which the actual processing of particle data is carried out in separate modules.

Like the main program, these modules are designed around a certain allocation of PE memory. To a lesser degree, they are also designed around a certain mapping of particle data into a 64-bit ILLIAC word. The major design feature of the particle data is that all data that refer to a certain particle (7 fields) are packed into a single ILLIAC word.

The actual integration proceeds by means of a time-centered leapfrog scheme. Forces are obtained by interpolation of potentials from neighboring grid-points. All 27 neighboring values are used in a quadratic interpolation of the potentials.

C) PROGRAMMING

This problem was written and debugged by 1 person. Writing and debugging required about 1 calendar year. Because of other commitments a reasonable estimate is about 1/2 to 3/4 of a man-year. Both design and checking were straightforward extensions from previous experience with 2-dimensional (point particles on a plane) programs. It was essential to be near the machine and in an environment where others were programming for the same machine in order to have ready access to help in learning how to interpret dumps, idiosyncrasies of the compilers and job submission procedures, etc. The programs were written in CFD, a FORTRAN-like complier written at Ames. This complier does not hide the machine architecture.

III. SCIENTIFIC RESULTS

A) EXPERIMENTS RUN

Nearly 200 distinct numerical experiments have been run in the past three years. These cover a variety of problems in galaxy dynamics. The major groupings of experiments are listed to give some idea of the range of problems studies. Fuller descriptions are being published in the astronomical literature.

1. Collapse of initially spherical rotating configurations. A two-parameter space was sampled: initial angular velocity of rotation and initial velocity dispersion. Some of the models passed through intermediate ring stages, but the rings were short-lived nonaxisymmetrically unstable forms. Other models passed through short-lived sheet structures. All systems adopted a steady-state barlike form rotating end-over-end in space. These experiements showed that bars are robust dynamically stable forms and strongly suggest that the three-dimensional forms of elliptical galaxies must be barlike.

2. Rapidly rotating axisymmetric configurations. Eliptical galaxies have conventionally been thought to be oblate axisymmetric objects flattened by rotation, rather than the barlike objects suggested by our experiments. The checks in this set of experiments show that stellar systems do not flatten as much by rotation as had been expected, even though some of our models rotate more rapidly than had been thought possible. This makes the case for the prolate (bar-like) objects stronger because rotation cannot account for the observed shapes of elliptical galaxies, quite apart from stability considerations.

3. Collapse of nonspherical rotating configurations. This sequence was run to test the sensitivity of the short-lived intermediate forms to lack of symmetry in the initial state. The sheets were quite sensitive, but rings could tolerate $\sim$ 10% asymmetries. All of these models ultimately formed bars as well.

4. Internal dynamics of a bar. This experiment used the bar at the end of one of the experiments in (1) above as the initial state in a study of the dynamical properties of a bar in an attempt to understand the peculiar stability of bars. This led to the discovery of a surprisingly large number of periodic orbits within the bar, and of a forward fluid streaming within the rotating bar pattern.

5. Colliding galaxies. Two stable self-consistent configurations were projected toward each other to study the response of fully self-consistent systems in these circumstances. These experiments led to the discovery of coherent flows within each galaxy in response to the force field of the other, which leads to larger energy transfer within a collision than had previously been expected. The galaxies contract momentarily at close passage.

6. Anisotropic models. Several experiments have been run in which models maintain nonspherical form by means of anisotropic "pressures". Thus far, none of these models has achieved a true steady state, but they are close.

7. Galaxy model in a bath of stars. These experiments are designed to study some peculiar galaxies (multiple-core cD's) in which several density condensations seem to survive within a larger diffuse background. Such configurations are difficult to understand dynamically. The experiments verify that such dense "cores" tend to diffuse on a dynamical timescale in the presence of a sea of low-density stars.

8. Protogalaxies. For these experiments, particles represent either gas or stars.

The experiment starts from a purely gaseous configuration. Stars form from the gas according to preassigned rules. Heavy element synthesis is a consequence of stellar evolution, and is taken into account in these models. Supernova outbursts stir up macroscopic motions in the remaining gas. These experiments are still in early stages but already they show a possible mechanism for formation of double radio sources symmetrically placed around a galaxy. The radio sources result from gas ejected from the protogalaxy by energy from supernova outbursts, which drives a pair of oppositely-directed jets.

9. Planetary formation. Gravitational instabilities in particulate matter left orbiting the sun in the early solar nebula have been confirmed by experiments in which our cubic volume is pictured as being in orbit around the sun. Particles aggregate into planetesimals as a result of these instabilities. These experiments are in progress.

10. Early universe. The origin of irregularities in the very smooth early universe at "recombination" is poorly understood. Experiments are planned in which growth rates, unstable modes, shapes left when instabilities reach finite amplitude, and so on, are studied.

B) ANALYSIS METHODS

This catalogue of projects underlines the need to consider means of analyzing results as part of the program design. These programs were designed with graphics as the primary analysis tool. Motion pictures are routinely made as our primary output and archival storage. We normally make motion pictures of temporal development as seen from three orthogonal view directions. Other view directions or special coordinates, velocities, and particle attributes for about 2000 particles are recorded in a plot file at each integration step and these files are used for motion pictures, to view results on CRT displays, or for numerical analyses.

We normally have little idea what to expect in an experiment. Motion pictures are a happy choice because unexpected results are more easily appreciated visually than by almost any other method. Even the fact that something unexpected is happening can be missed without such a powerful and general method. While production and quality control problems are often annoying and turnaround can be impossibly long (we've experienced 6 months!" the power of the method, the discovery of unexpected results, and the beauty of the pictures makes it worth the pain.

To stress the power, we mention several discoveries made in these experiments that would not have been found by other methods. (1) The appearance of sheets in the collapses of spheres. (2) The discovery of the remarkable stability of barlike forms. (3) The discovery of the coherent responses in galaxy collisions. (4) The discovery of oppositely directed jets in the protogalaxy problem. (5) The discovery of forward streaming motions within rotating bars. In each case, the coherent picture of changing configuration, easily afforded by motion pictures, is an essential part of the discovery.

Motion pictures are an essential part of this entire project. They are not simply an appealing way of presenting results that we could equally well have found by other means. They are the tool by which the important discoveries of this project have been made. The fact that discoveries made in these experiments tend to relate to properties other than those which the experiments were designed to study (e.g., radio sources when studying galaxy formation) underlines their unexpected nature and the power of graphic presentation.

It is a pleasure to thank Margaret Covert for assistance with the operation of computer programs and in handling the data generated. Computations reported were carried out on the ILLIAC IV computer at the Institute for Advanced Computation. Funds for partial support of this study have be allocated by the NASA-Ames Research Center, Moffett Field, California, under Interchange No. NCA2-OR108-902. R.H.M. received partial support from NSF grant AST 76-14289.

PARTICLE SIMULATION ON THE VAP

W. E. Drummond and B. N. Moore
Austin Research Associates
1901 Rutland Drive
Austin, Texas 78758

ABSTRACT

A report will be given of the status of hardware and software development on the VAP. The VAP is designed as a floating point vector array processor of considerable power and will be able to execute large vector programs at high speed in a stand-alone mode. Other design features are multiple large fast data memories with independent data paths to a pipelined arithmetic unit. Four of these memories are vector (serial) memories with a maximum size of 8 million words. There is also a scalar (ram) memory with a total of 1 million words. Practical operation at speeds of 12 mflops will be possible. A software package is under development which will eventually make it possible to program at the Fortran level. Applications to plasma simulation will be discussed.

INTRODUCTION

Austin Research Associates is developing a floating point vector array processor, the VAP. This development was originally motivated by the need for an inexpensive high speed vector processor for large-scale plasma simulations. However, the architecture of the VAP provides an extraordinary degree of flexibility in vectorizing algorithms encountered in the solution of physical problems and, as a result, the VAP should have a fairly wide applicability.

The principal attributes of the VAP are:

1. It executes large vector programs at high speed.

   For plasma simulation problems, it is expected to be approximately three times faster than a CDC-7600.

2. It has multiple, fast, data memories with independent data paths to the pipelined arithmetic units. Four of these memories are vector (serial) memories and one is a fast scalar (ram) memory. The initial configuration will have a total of 1.5 million words of fast memory and can be easily expanded to a much larger total memory.

   For fully electromagnetic 2-1/2 D plasma simulation problems, the initial configuration handles 25,000 cells and 200,000 particles.

3. Programming is carried out using a subset of standard Fortran statements plus a few additional statements unique to vector programming.

4. It is inexpensive--less than 4 percent of the cost of a Cray I or a Star.

In the following sections, the organization of the VAP memories and data flow logic will be described, together with certain features of the VAP software and utility packages.

MACHINE DESCRIPTION

The CPU of the VAP is a modified Floating Point Systems, Inc. AP-120B array processor. The AP-120B is a high-speed synchronous processor with a cycle time of 167 nanoseconds and executes one instruction per cycle. The instruction word provides the capability of overlapping 10 independent operations in each instruction, i.e., 10 independent instructions per cycle. The floating point arithmetic units consist of a pipelined multiplier and a pipelined adder, each of which can produce one result per cycle. Thus, the maximum execution rate for floating point operations is 12 megaflops.

To write and debug optimized code for the AP-120B requires assembly language programming which is extremely tedious because of the overlapping of multiple operations on each instruction. In addition, because of data path and memory conflicts, even carefully written assembly language code does not make very efficient use of the arithmetic pipelines. Finally, there is only one data memory which is limited to a maximum of a million words. Even with these restrictions, however, we have written plasma simulation codes on the AP-120B which execute at approximately the same speed as on the CDC-7600. Thus, the AP-120B is a cost-effective processor for many applications.

The VAP was designed to make use of the many attractive features of the AP-120B, while at the same time removing the memory size and data path conflict restrictions. The resulting hardware configuration lends itself to vector processing and the development of a vector Fortran compiler removes the need for assembly language coding. With this vector Fortran compiler, programming can be carried out as easily as on a standard Fortran processor and programs execute approximately three times faster than on a CDC-7600.

The principal modifications to the AP-120B involve the addition of four high-speed data and control paths to the CPU, together with the expanded hardware logic to facilitate the flexible use of these additional data paths. The four high-speed data paths are connected through controllers to four high-speed ram memories which are used as vector (serial) memories. The use of ram memories avoids the latency problems associated with CCD or bubble memories.

The resulting VAP functional diagram is shown in Figure 1 and more details of the array processor section are given in Figure 2. In summarizing machine features and capabilities, it is convenient to distinguish between those characteristics of the conventional AP mode of operation given in Table 1 and the extended VAP mode given in Table 2. Under software control, the VAP can operate either as a vector array processor or in a conventional AP mode.

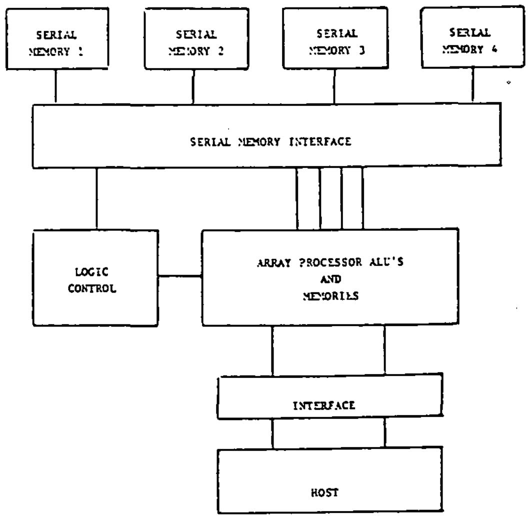


Fig. 1. VAP Machine Organization

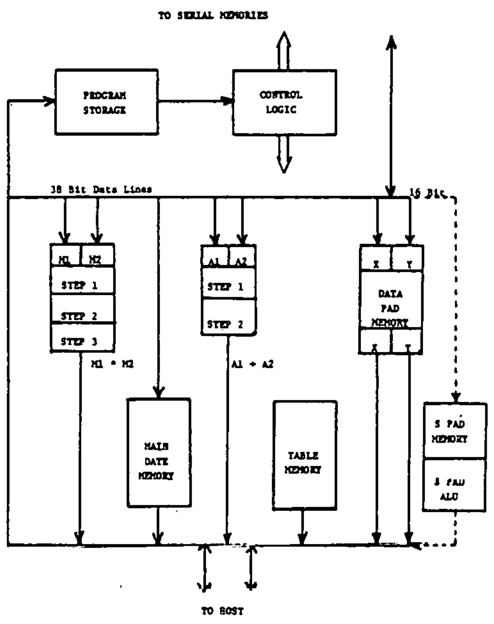


Fig. 2. Details of Array Processor CPU.

Table 1. AP mode features.

---

* 167 nsec cycle time.
* Independent pipelined floating adder and multiplier.
* Pipelined access to as much as 512K words of ram data memory at rates up to 6 million words/second.
* Sixteen 16-bit integer scratch registers with associated ALU.
* 2K table memory rom.

For large vector programs, the execution speed is primarily limited by the maximum rate from the data memories to the ALU. The addition of the four fast data memories increases the data rate for the VAP to 30 million words per second as compared to 6 million words per second in the AP mode.

Table 2. VAP features.

---

* All features of conventional AP available.
* 4 (n x 64K) serial memories configurable under program control as inputs and outputs of arithmetic units.
* Access rates of 6 million words/second to each memory giving total data flow rate of 30 million words/second.
* Setup time for vector operations ≈ 4 microseconds.

The additional control logic incorporated in the VAP takes advantage of the flexibilities provided by a mixture of vector and scalar memories so that scatter/gather operations which, in the past, were not thought to be vectorizable, are, in fact, easily vectorizable. For plasma simulation problems, the scatter/gather operations of interpolation and allocation are thus materially speeded up.

SOFTWARE

Parallel development of software and hardware is being undertaken in order that hardware design options may be realistically evaluated as they become apparent. The earliest possible useful production from the machine will also be obtained. VAP software items can be classified as support, e.g., compiler, assembler, linker, etc., or as utilities. Typical utilities include commonly used vector arithmetic operations, as well as specialized utilities for plasma simulation problems. Figure 3 is a block diagram indicating the stages required to convert VAP Fortran source code into an execution module. All of the support software items appear in Figure 3. All of the compiling, assembling, and linking is done on the host computer and the complete binary execution module is shipped from the host to the VAP for stand-alone execution. Results and diagnostic data can be returned to the host during execution.

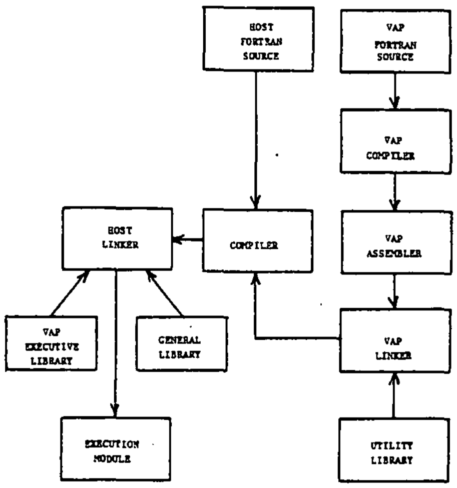


Fig. 3. Support software.

Programming is done principally in terms of a subset of standard Fortran statements with a few modifications unique to vector operations. Variables are declared as either vector or scalar variables and then used in the usual Fortran syntax.

For example, if B, C & D are vectors of the same length, located in different vector memories, the Fortran statement

$$A = B * C + D$$

multiplies each element of B times the corresponding element of C and adds the product to the corresponding element of D, to produce the resulting vector A, which is stored in the remaining vector memory.

Scatter/gather operations make use of additional symbols but the same Fortran syntax. E.g., if B, C & J are vectors located in different serial memories, the Fortran statement

$$A = B * C + M\langle J\rangle$$

multiplies each element of B by the corresponding element of C and adds the product to the contents of the scalar memory at the address specified by the corresponding element of J to give the resulting vector A, which is stored in the fourth vector memory. This scatter/gather vector operation executes at the same speed as the pure vector operation discussed above.* Table 3 lists the execution rate of typical Fortran operations.

Table 3. Execution rates for VAP Fortran statements and utilities.[a]

Pure Vector Operations	
1. $A + B * C$	6 Megaflops
2. $A = (B \pm C)$	6 Megaflops
3. $A = B * C \pm D$	12 Megaflops
4. $A = (B \pm C) * D$	12 Megaflops
Scatter/Gather Vector Operations	
1. $A = B * C \pm M\langle J\rangle$	12 Megaflops
2. $M\langle J\rangle = M\langle J\rangle + A * B$	4 Megaflops
Vector Utilities	
1. A = SQRT(B)	1.5 Megaflops
2. A = 1/B	1.5 Megaflops

[a]In this table, A, B, C and J are vectors located in different vector memory. $M\langle J\rangle$ is the contents of the scalar memory location whose address is the corresponding element of the vector J.

*Other scatter/gather operations, e.g., scatter/gather operation No. 2 in Table 3 may run more slowly for two reasons: The scalar memory is accessed twice for the operations on each element; and if adjacent elements of J have the same value, i.e., if the same memory location is addressed for two consecutive elements, A critical element in any vector processor is the setup time for vector operations. For the VAP, this setup time is between three and four microseconds. Since the basic VAP cycle time is 167 ns, this means that the setup time for any vector operation consumes approximately 20 cycles. Since the most common vector operations execute one vector element per machine cycle, the overhead time, as a fraction of the execution time, is simply 20 divided by the number of elements in the vector. For vectors of length 200, the setup time is thus 10 percent of the execution time. For particle simulation problems, the typical vector length is 2,000, and thus setup time amounts to only 1 percent of the execution time. As a result, the utilization of the arithmetic pipelines approaches 100 percent in the VAP.

APPLICATIONS

Although the VAP is being developed because it is needed for a rather specific problem, it will be capable of quite general application. Some problems for which the VAP will be useful are listed in Table 4. Consideration of the 2-1/2 D plasma simulation problem will illustrate some of the features of the VAP. A 2-1/2 D fully-electromagnetic, fully relativistic e-beam simulation code has been in production on the AP for some time and it will be the first major code to be implemented on the VAP. An estimated performance comparison is given in Table 5. The VAP can handle larger field arrays and particle tables because of the serial memory capacity, and the availability of optimized Fortran operations and efficiently coded vector utilities makes larger pieces of the code run at the 12 megaflop rate. With the VAP Fortran compiler, programs will also be more easily modified and debugged.

Table 4. Applications.

- * 2-1/2 D and 3 D plasma simulation.
- * Hydrodynamic and magnetohydrodynamic problems.
- * Simulation of diode operation.
- * General 2 D and 3 D partial differential equations.

OUTLOOK

An operating prototype of the VAP with reduced data path width and skeleton serial memory should be available by the end of the year to perform tests of the design. If all goes reasonably well at that point, it is anticipated that a fully operational machine will be working by the middle of next year.

Most of the software will be available before the prototype VAP is operational, and work will proceed on actual code development.

---

the write to this memory location from the first of these elements would not be completed before the read of that same memory location for the next element. To guard against this possibility, the utility has been slowed. For plasma simulation codes, a special utility has been written for this operation which executes at 6 megaflops.

Table 5. Comparison of the capabilities of the AP mode and the VAP mode for a 2-1/2 D fully-electromagnetic, fully-relativistic, particle push program.

	AP	VAP
Number of particles	32K	200K
Number of cells	4K	25K
Execution time per particle[a]	81 μsec/particle	21 μsec/particle[b]
Ease of programming	Difficult	Same as Standard Fortran

[a]For purposes of comparison, standard Fortran programming of the CDC-7600 gives an execution time of about 80 microseconds per particle and standard Fortran programming of the Cray I initially achieved an execution time of approximately 12-1/2 microseconds per particle. However, more recent hand-optimized coding on the Cray I has led to a significant reduction in this push time. (Private communication - D. Forslund)

[b]This result is not a measured result. However, since the processor is a synchronous processor, it is believed to be accurate.

Dup

A VECTORIZED FOKKER-PLANCK PACKAGE FOR THE CRAY-1*

M. G. McCoy, A. A. Mirin, J. Killeen
Magnetic Fusion Energy Computer Center
Lawrence Livermore Laboratory
P. O. Box 5509
Livermore, CA 94550

ABSTRACT

A program for the solution of the time-dependent, two dimensional, nonlinear, multi-species Fokker-Planck equation is described. The programming is written such that the loop structure is highly vectorizable on the CRAY FORTRAN Compiler. A brief discussion of the Fokker-Planck equation itself is followed by a description of the procedure developed to solve the equation efficiently. The Fokker-Planck equation is a second order partial differential equation whose coefficients depend upon moments of the distribution functions. Both the procedure for the calculation of these coefficients and the procedure for the time advancement of the equation itself must be done efficiently if significant overall time saving is to result. The coefficients are calculated in a series of nested loops, while time advancement is accomplished by a choice of either a splitting or an ADI technique. Overall, timing tests show that the vectorized CRAY program realizes up to a factor of 12 advantage over an optimized CDC-7600 program and up to a factor of 3.65 over a non-vectorized version of the same program on the CRAY.

INTRODUCTION

In the simulation of magnetically confined plasmas in which the ions or electrons are non-Maxwellian and where a knowledge of the distribution functions is important, kinetic equations must be solved. In both mirror and tokamak confinement devices non-Maxwellian plasmas may be present. This may be due to the presence of extensive loss regions in velocity space or to the presence of monoenergetic neutral beams. The kinetic equation to be solved is the Boltzmann equation with Fokker-Planck Coulomb collision terms.

This nonlinear partial differential equation, which describes the evolution of the distribution functions of all charged species in the plasma, has seven independent variables (three spatial coordinates, three velocity coordinates and time). Consequently, a number of simplifications must be introduced into the equation to allow any present day computer to solve the problem. One can reduce the number of independent variables to three by neglecting spatial dependence and by assuming azimuthal symmetry of the distribution functions about the direction of the magnetic field. With these simplifications Rosenbluth, MacDonald and Judd(1) succeeded in expressing the equation in a form suitable for solution on a computer.

Subsequently, a number of computer programs have been developed(2),(3) which solve this equation. These Fokker-Planck codes are frequently incorporated in larger codes which simulate additional physical processes such as quasi-linear diffusion due to radio frequency heating(4) and spatial diffusion of the plasma(5).

A problem encountered in these programs has been the long run times required to solve the Fokker-Planck equation, and this problem is aggravated in studies for which it is necessary to solve for the operator at a large number of spatial points or when very high resolution is required.

With the arrival of the CRAY-1, it has become possible to gain a full order of magnitude savings in computer time with respect to a program optimized for the

CDC-7600 by casting the equations in a tractable form and applying some vectorization techniques. The program is compiled on the CRAY using CFT. Provided that careful attention is paid to "do loop" coding, this program provides an example of the capability of the compiler to vectorize coding and improve performance. A package, available to the community, has been written which solves the Fokker-Planck equation and has been incorporated in a number of programs.(3),(4),(5)

THE FOKKER-PLANCK EQUATION

The Boltzmann equation for the distribution function of each plasma species is given by

$$\frac{\partial f_a}{\partial t} + \underline{v} \cdot \frac{\partial f_a}{\partial \underline{x}} + \frac{\underline{F}}{m_a} \frac{\partial f_a}{\partial \underline{v}} = \left(\frac{\partial f_a}{\partial t}\right)_c \qquad (1)$$

where f_a is the distribution function of species type "a", $\underline{F}$ is a force field and $\left(\frac{\partial f}{\partial t}\right)_c$ is the Fokker-Planck operator representing the rate of change of the distribution function due to Coulomb collisions.

With the simplifying assumption that the Fokker-Planck operator depends only on the velocity, $\underline{v}$, and not upon spatial position, $\underline{x}$, the operator as derived by Rosenbluth et al.(1) has the form

$$\left(\frac{\partial f}{\partial t}\right)_c = \Gamma_a \left\{ -\frac{\partial}{\partial v_i}\left(f_a \frac{\partial h_a}{\partial v_i}\right) + \tfrac{1}{2} \frac{\partial^2}{\partial v_i \partial v_j}\left(f_a \frac{\partial^2 g_a}{\partial v_i \partial v_j}\right)\right\} \qquad (2)$$

where the usual summing conventions over repeated indices "i" and "j" are to be used and where $\Gamma_a = 4\pi Z_a^4 e^4/m_a^2$. The "Rosenbluth potentials" are written as

$$g_a = \sum_b \left(\frac{Z_b}{Z_a}\right)^2 \ln \Lambda_{ab} \int f_b(\underline{v}')\,|\underline{v} - \underline{v}'|\,d\underline{v}' \qquad (3)$$

$$h_a = \sum_b \frac{m_a + m_b}{m_b} \left(\frac{Z_b}{Z_a}\right)^2 \ln \Lambda_{ab} \cdot \int f_b(\underline{v}')\,|\underline{v} - \underline{v}'|^{-1}\,d\underline{v}'. \qquad (4)$$

The Coulomb logarithm which depends on both interacting species is

$$\ln \Lambda_{ab} = \ln \left\{ \left(\frac{m_a + m_b}{m_a + m_b}\right) \frac{2\alpha c \lambda_D}{e^2} \cdot \sup \left(\frac{2E_k}{m_k}\right)^{1/2}_{a,b} \right\}^{-1/2}$$

where $\alpha = 1/137$ is the fine structure constant, $\lambda_D = \sqrt{E_e/6\pi n_e e^2}$ is the Debye length, E_k is the energy of species k, and n_e is the electron density.

If we assume further that the distribution functions f_a are azimuthally symmetric about the direction of the magnetic field, the formulation given above may be recast as a two dimensional problem when written in spherical polar coordinates (v,θ,ϕ), where v is the speed, θ is the angle between the velocity vector and the magnetic field, and ϕ is the azimuthal angle. Equation (2) then becomes

$$\frac{1}{\Gamma_a}\left(\frac{\partial f_a}{\partial t}\right)_c = \frac{1}{v^2}\frac{\partial G_a}{\partial v} + \frac{1}{v^2 \sin\theta}\frac{\partial H_a}{\partial \theta}, \qquad (5)$$

where

$$G_a = A_a f_a + B_a \frac{\partial f_a}{\partial v} + C_a \frac{\partial f_a}{\partial \theta} \qquad (6)$$

$$H_a = D_a f_a + E_a \frac{\partial f_a}{\partial v} + F_a \frac{\partial f_a}{\partial \theta} \qquad (7)$$

The coefficients A_a, B_a, C_a, D_a, E_a and F_a satisfy

$$A_a = \frac{v^2}{2}\frac{\partial^3 g_a}{\partial v^3} + v\frac{\partial^2 g_a}{\partial v^2} - \frac{\partial g_a}{\partial v} - v^2\frac{\partial h_a}{\partial v} - \frac{1}{v}\frac{\partial^2 g_a}{\partial \theta^2} + \frac{1}{2}\frac{\partial^3 g_a}{\partial v \partial \theta^2} - \frac{\cot\theta}{v}\frac{\partial g_a}{\partial \theta} + \frac{\cot\theta}{2}\frac{\partial^2 g_a}{\partial v \partial \theta} \qquad (8)$$

$$B_a = \frac{v^2}{2}\frac{\partial^2 g_a}{\partial v^2} \qquad (9)$$

$$C_a = -\frac{1}{2v}\frac{\partial g_a}{\partial \theta} + \frac{1}{2}\frac{\partial^2 g_a}{\partial v \partial \theta} \qquad (10)$$

$$D_a = \frac{\sin\theta}{2v^2}\frac{\partial^3 g_a}{\partial\theta^3} + \frac{\sin\theta}{2}\frac{\partial^3 g_a}{\partial v^2\partial\theta} + \frac{\sin\theta}{v}\frac{\partial^2 g_a}{\partial v\partial\theta}$$

$$- \frac{1}{2v^2\sin\theta}\frac{\partial g_a}{\partial\theta} + \frac{\cos\theta}{2v^2}\frac{\partial^2 g_a}{\partial\theta^2} - \sin\theta\frac{\partial h_a}{\partial\theta} \tag{11}$$

$$E_a = \sin\theta\left[-\frac{1}{2v}\frac{\partial g_a}{\partial\theta} + \frac{1}{2}\frac{\partial^2 g_a}{\partial v\partial\theta}\right] \tag{12}$$

$$F_a = \frac{\sin\theta}{2v^2}\frac{\partial^2 g_a}{\partial\theta^2} + \frac{\sin\theta}{2v}\frac{\partial g_a}{\partial v}\,. \tag{13}$$

As suggested by Rosenbluth et al.(1), the "Rosenbluth potentials" and the distribution functions themselves may be represented by expansions in Legendre polynomials. For this purpose we let

$$f_b(v,\theta,t) = \sum_{j=o}^{\infty} V_j^b\,(v,t)\;P_j\;(\cos\theta). \tag{14}$$

where

$$V_j^b(v,t) = \frac{2j+1}{2}\int_{-1}^{+1} f_b\;(v,\cos\theta,t)$$

$$\cdot\; P_j(\cos\theta)\;d(\cos\theta) \tag{15}$$

The expansions for the potentials are

$$g_a(v,\theta,t) = \sum_{j=o}^{\infty}\sum_b\left(\frac{Z_b}{Z_a}\right)^2 \ln\;\Lambda_{ab}$$

$$\cdot\; B_j^b\;(v,t)\;P_j(\cos\theta) \tag{16}$$

and

$$h_a(v,\theta,t) = \sum_{j=o}^{\infty}\sum_b\left(\frac{m_a+m_b}{m_b}\right)\left(\frac{Z_b}{Z_a}\right)^2$$

$$\cdot\;\ln\Lambda_{ab}\;A_j^b(v,t)\;P_j(\cos\theta) \tag{17}$$

with coefficients

$$A_j^b = \frac{4\pi}{2j+1}\left[\int_0^v \frac{(v')^{j+2}}{v^{j+1}}\,V_j^b(v',t)\,dv'\right.$$

$$\left.+\int_v^{\infty}\frac{v^j}{(v')^{j-1}}\,V_j^b(v',t)\,dv'\right] \tag{18}$$

$$B_j^b = -\frac{4\pi}{4j^2-1}\left[\int_0^v \frac{(v')^{j+2}}{v^{j-1}}\right.$$

$$\cdot\left(1-\frac{j-1/2}{j+3/2}\,\frac{(v')^2}{v^2}\right)V_j^b(v')\,dv'$$

$$+\int_v^{\infty}\frac{v^j}{(v')^{j-3}}\left(1-\frac{j-1/2}{j+3/2}\,\frac{v^2}{(v')^2}\right)$$

$$\left.\cdot\;V_j^b(v')\,dv'\right]\;. \tag{19}$$

It is convenient to define four functionals

$$M_j(w)\;(v) = \int_v^{\infty} w(y)\;y^{(1-j)}\;dy \tag{20}$$

$$N_j(w)\;(v) = \int_o^{v} w(y)\;y^{(2+j)}\;dy \tag{21}$$

$$R_j(w)\;(v) = \int_v^{\infty} w(y)\;y^{(3-j)}\;dy \tag{22}$$

$$E_j(w)\;(v) = \int_o^{v} w(y)\;y^{(4+j)}\;dy\;. \tag{23}$$

Using these functionals, Eqs. (18) and (19) become

$$A_j^b = \frac{4\pi}{(2j+1)}\left[v^{-j-1}\;N_j(V_j^b)\right.$$

$$\left.+\;v^j\;M_j\;(V_j^b)\right] \tag{24}$$

$$B_j^b = \frac{4\pi}{2j+1}\left\{\frac{1}{(2j+3)}\left[v^{-j-1}\;E_j\;(V_j^b)\right.\right.$$

$$\left.+\;v^{j+2}\;M_j\;(V_j^b)\right]$$

$$-\frac{1}{(2j-1)}\left[v^{1-j}\;N_j(V_j^b)\right.$$

$$\left.\left.+\;v^j\;R_j\;(V_j^b)\right]\right\} \tag{25}$$

Evaluation of the coefficients defined in Eqs. (8)-(13) requires derivatives of the "Rosenbluth potentials". These may be obtained through term by term differentiation of Eqs. (16)-(17). To obtain derivatives with respect to v requires differentiation of the right hand side of Eqs. (24)-(25). This is done analytically.

The resulting expressions are

$$\frac{\partial A_j^b}{\partial v} = \frac{4\pi}{(2j+1)} \left[j\, v^{j-1}\, M_j(V_j^b) - (j+1)\, v^{-j-2}\, N_j(V_j^b) \right] \quad (26)$$

$$\frac{\partial B_j^b}{\partial v} = \frac{4\pi}{(2j+1)} \left\{ \frac{1}{(2j+3)} \left[(j+2)\, v^{j+1}\, M_j(V_j^b) - (j+1)\, v^{-j-2}\, E_j(V_j^b) \right] - \frac{1}{(2j-1)} \left[j v^{j-1}\, R_j(V_j^b) - (j-1)\, v^{-j} N_j(V_j^b) \right] \right\} \quad (27)$$

$$\frac{\partial^2 B_j^b}{\partial^2 v} = \frac{4\pi}{(2j+1)} \left\{ \frac{(j+1)(j+2)}{(2j+3)} \left[v^{-j-3}\, E_j(V_j^b) + v^j\, M_j(V_j^b) \right] - \frac{j(j-1)}{(2j-1)} \left[v^{-j-1}\, N_j(V_j^b) + v^{j-2}\, R_j(V_j^b) \right] \right\} \quad (28)$$

$$\frac{\partial^3 B_j^b}{\partial^3 v} = \frac{4\pi}{(2j+1)} \left\{ \frac{1}{(2j+3)} \cdot \left[j(j+1)(j+2)\, v^{j-1}\, M_j(V_j^b) - (j+1)(j+2)(j+3)\, v^{-j-4}\, E_j(V_j^b) \right] - \frac{1}{(2j-1)} \left[j(j-1)(j-2)\, v^{-j-3}\, R_j(V_j^b) - (j+1)(j)(j-1)\, v^{-j-2}\, N_j(V_j^b) \right] \right\} . \quad (29)$$

Derivatives with respect to θ are also done analytically.

SOLUTION PROCEDURE

The numerical finite difference solution to the problem consists of two separate parts: the first is concerned with obtaining the coefficients A_a through F_a (Eqs. (8) through (13)) while the second is devoted to the actual time advancement of the Fokker-Planck operator.

COMPUTATION OF COEFFICIENTS

Since the structure of each of the six coefficients is quite similar, we will, for simplicity, consider the detailed evaluation of only one of them, namely C_a (see Eq. (10)).

Letting

$$\tilde{C}_j^b\,(v_k,\theta_i) = -\frac{1}{2v_k}\, B_j^b(v_k)\ \frac{\partial}{\partial\theta} P_j(\cos\theta_i) + \frac{1}{2}\frac{\partial}{\partial v} B_j^b\ (v_k)\ \frac{\partial}{\partial\theta} P_j(\cos\theta_i), \quad (30)$$

we may, using Eqs. (10) and (16), express C_a in terms of the various $\tilde{C}_j^b$ coefficients as

$$C_a(v_k,\theta_i) = \sum_b \ell n\, \Lambda_{ab} \left(\frac{Z_b}{Z_a}\right)^2 \cdot \sum_{j=o}^{M} \tilde{C}_j^b\ (v_k,\theta_i) \ . \quad (31)$$

Note that we have truncated the Legendre series at M+1 polynomials and have introduced velocity and theta meshes $\left\{v_k\right\}_{k=1}^{kmax}$ and $\left\{\theta_i\right\}_{i=1}^{imax}$. Coefficients $\tilde{A}_j^b$ through $\tilde{F}_j^b$ may be similarly defined and expressed.

The expressions for the coefficients $\tilde{A}_j^b$ through $\tilde{F}_j^b$ are complicated (e.g. Eq. 30) and these coefficients must be evaluated at all of the mesh points. In fact, the computation of these coefficients represents a high percentage of the total number of arithmetic operations required; consequently, it is necessary to calculate these coefficients as rapidly as possible. This is accomplished through a series of nested "do-loops" which the CFT compiler vectorizes.

The coefficients $\tilde{A}_j^b(v_k,\theta_i)$ through $\tilde{F}_j^b(v_k,\theta_i)$ are functions of the four indices, "b", "j", "k" and "i", and efficiency considerations dictate that the species index "b" and the Legendre index "j" form the outer-most "do-loops", since "b" and "j" are never large. Within these two loops one must evaluate Eq. (30) for all (v_k,θ_i) mesh points. The procedure may be outlined as follows:

(1) Calculate the Legendre coefficients $\{V_j^b(v_k,t)\}_{k=1}^{kmax}$ (Eq. (15)). This involves the calculation of kmax 1-D integrals over θ, and may be calculated in two nested loops with the outermost loop over θ ("i"). This will permit compiler vectorization of the entire procedure, since the inner loop over v ("k") is not recursive.

(2) Evaluate the functionals $\{M_j(V_j^b)(v_k), N_j(V_j^b)(v_k), R_j(V_j^b)(v_k), E_j(V_j^b)(v_k)\}_{k=1}^{kmax}$ (See Eqs. (20) - (23)). This is accomplished in two separate loops over "k". For the functional $N_j(V_j^b)(v_k)$, in the first loop we calculate the temporary array TEM(k) = $V_j^b(v_k)\ v_k^{(2+j)}\ \Delta v_k$, where Δv_k is a mesh increment, while in a second separate loop we add together these temporaries to form the functional $N_j(V_j^b)(v_k)$. This second loop does not vectorize since it is recursive.

(3) Determine the coefficients $\{B_j^b(v_k), \frac{\partial}{\partial v}B_j^b(v_k), \text{etc}\}_{k=1}^{kmax}$ (Eqs. (24)-(29)). This is accomplished through a single vectorizable loop over "k".

(4) Determine $\{\tilde{A}_j^b(v_k,\theta_i)$ through $\tilde{F}_j^b(v_k,\theta_i)\}_{i,k=1}^{imax,kmax}$. It is evident from Eq. (30) that the evaluation procedure is vectorizable. We choose the theta ("i") loop on the outside, since it is normally much smaller than the velocity ("k") loop. The Legendre polynomials and their derivatives are time-independent and are stored quantities. As the coefficients $\tilde{A}_j^b - \tilde{F}_j^b$ are calculated, they are simultaneously summed over "j" since it is the sum, $\sum_{j=0}^{M} \tilde{C}_j^b$, that is required in Eq. (30).

(5) Evaluate the contribution of the bth species to the Fokker-Planck coefficients $A_a(v_k,\theta_i,t)$ through $F_a(v_k,\theta_i,t)$ for all species "a" (see Eq.(31)). This is accomplished within the two outer loops over "b" and "j" and within a third loop over "a". The procedure easily vectorizes.

TIME ADVANCEMENT

The second part involves the numerical integration of the finite difference analog of the equation

$$\frac{1}{\Gamma_a}\frac{\partial f}{\partial t} = \frac{1}{v^2}\frac{\partial G_a}{\partial v} + \frac{1}{v^2 \sin\theta}\frac{\partial H_a}{\partial v} \qquad (5)$$

Both operator splitting and ADI methods have been used to solve this equation. Consider the splitting scheme

$$\frac{1}{\Gamma_a}\frac{\partial f_a}{\partial t} = \frac{1}{v^2}\frac{\partial G_a}{\partial v} \qquad (32)$$

$$\frac{1}{\Gamma_a}\frac{\partial f_a}{\partial t} = \frac{1}{v^2 \sin\theta}\frac{\partial H_a}{\partial \theta} \qquad (33)$$

Equation (32) is discretized as follows:

$$\frac{f_{k,i}^{n+1} - f_{k,i}^{n}}{\Gamma_a \Delta t} = \frac{A_{k+1,i}^{n} f_{k+1,i}^{n+1} - A_{k-1,i}^{n} f_{k-1,i}^{n+1}}{2v_k^2 \Delta v_k} + \frac{1}{v_k^2}\left[B_{k+\frac{1}{2},i}^{n}\frac{\left(f_{k+1,i}^{n+1} - f_{k,i}^{n+1}\right)}{\Delta v_{k+\frac{1}{2}}} - B_{k-\frac{1}{2},i}^{n}\cdot\frac{\left(f_{k,i}^{n+1} - f_{k-1,i}^{n+1}\right)}{\Delta v_{k-\frac{1}{2}}}\right] + \frac{1}{2v_k^2\Delta v_k}\left[C_{k+1,i}^{n}\cdot\frac{\left(f_{k+1,i+1}^{n} - f_{k+1,i-1}^{n}\right)}{2\Delta\theta_i} - C_{k-1,i}^{n}\cdot\frac{\left(f_{k-1,i+1}^{n} - f_{k-1,i-1}^{n}\right)}{2\Delta\theta_i}\right]. \qquad (34)$$

The coefficients $B_{k\pm\frac{1}{2},i}^{n}$ are simple averages of $B_{k,i}$ and $B_{k\pm1,i}$, and the Δv's and Δθ's are mesh increments. The subscript "a" has been dropped.

This equation may be written in the form

$$-A_{k,i}^{n} f_{k+1,i}^{n+1} + B_{k,i}^{n} f_{k,i}^{n+1} - C_{k,i}^{n} f_{k-1,i}^{n+1} = D_{k,i}^{n}$$

Employing a standard technique for solving tridiagonal systems[6], the problem is solved recursively. We have

$$f_{k,i}^{n+1} = E_{k,i} f_{k+1,i}^{n+1} + F_{k,i} \qquad (35)$$

where

$$E_{k,i} = A_{k,i} \Big/ \left(B_{k,i} - C_{k,i}\, E_{k-1,i}\right) \qquad (36)$$

$$F_{k,i} = \left(D_{k,i} + C_{k,i}\, F_{k-1,i}\right) \Big/ \left(B_{k,i} - C_{k,i}\, E_{k-1,i}\right) \qquad (37)$$

with $F_{1,i}$, $E_{1,i}$, and $f^{n+1}_{kmax,i}$ determined through boundary conditions.

The use of two dimensional matrices for E and F permits vectorization of the calculation. Clearly since the arrays E and F are large, if there were no intention of vectorizing the coding, one dimensional arrays of length sup {kmax,imax} could be used in place of E and F, thereby saving storage. This is the case in the 7600 version of the program.

The calculation on the CRAY proceeds as follows:

(1) Determine the matrices $\left\{E_{k,i}, F_{k,i}\right\}^{imax,kmax}_{i,k=1}$. This is accomplished in two nested loops with index "k" on the outside and "i" on the inside. This ordering is forced if one wishes to vectorize the inner loop since E and F are defined recursively on index "k".

(2) Calculate $\left\{f^{n+1}_{k,i}\right\}^{kmax,imax}_{k,i=1}$ (See Eq.(35)). This too is vectorized by placing the loop over "k" on the outside since $f^{n+1}_{k,i}$ is determined recursively in "k".

A similar procedure, with loops reversed, is employed for the second half of the splitting procedure. An ADI scheme can also be used, but requires a little more time due to the need to evaluate the explicit terms.

It must be noted here that any comparison between two programs, one optimized for the CRAY-1 and the other for the CDC-7600, must take into account the different sizes and types of memories of the two machines - in particular the fact that most of the 7600 memory, LCM, is slow access. While it is possible to optimize a 7600 program with STACKLIB commands, and efforts have been made to do so, one is still forced constantly to swap back and forth between SCM and LCM. Furthermore, one is often inhibited from attempting more STACKLIB or vector operations on the 7600 due to the increased memory requirements in SCM needed for vectorization. For instance, on the 7600 the arrays E and F defined in Eqs. (36) and (37) are one dimensional. To add two more arrays of length kmax by imax would be prohibitive. This eliminates any possibility of STACKLIB vectorization on the 7600 of this particular solution procedure, further increasing the advantage of the CRAY-1 over the 7600.

TIMING RESULTS AND CONCLUSIONS

In order to determine the efficiency of the vectorized program, a number of timing tests for various mesh resolutions were performed. Comparisons were carried out between the vectorized program on the CRAY-1 (CRV in column two of TABLE-I below), the same program on the CRAY-1 with the vector option turned off (CRNV below), and the STACKLIB optimized 7600 program (denoted by 7600 below). In all of these examples, Fokker-Planck equations for the distribution functions of two charged species were time-advanced in the presence of three background Maxwellian species. A value of M=4 was used for each of the two primary species, whereas M=0 sufficed for the background Maxwellians.

The immediate conclusion is that the CRAY vectorized program achieves at least an order of magnitude advantage over the 7600 version and, depending on mesh size, a factor of from 2.86 to 3.65 over the nonvectorized CRAY version. Separate timings were carried out for the calculation of the Fokker-Planck coefficients and the time-advancement of the Fokker-Planck operator. Of general interest is the marked gain in efficiency which results from the relatively simple procedure of vectorizing the operator splitting or ADI section of coding. As much as a factor of 4.4 has been achieved over the non-vectorized version and a factor of 14 over the 7600 version.

The effects of vectorization can be readily seen in TABLE-II, where the computer time per mesh point is given. Note that the efficiency steadily increases with increasing mesh size up to a resolution of 64 x 45. There is a small loss of efficiency at 65 x 45 due to the increased overhead in vectorization. Overall peak efficiency is reached at 64 x 45 and little speed is gained at higher resolutions.

The Fokker-Planck equation is a fairly typical example of the type of complicated nonlinear partial differential equation being solved today. The procedure employed to optimize the solution of this equation basically consists of organizing the loop structure so that the compiler can vectorize the code efficiently. It appears likely that similar order of magnitude improvements can be achieved with the use of no more than CRAY FORTRAN in most programs which solve equations of this type.

Table 1. CRAY-1 vs. CDC-7600 Improvement Factors

MESH (kmax x imax)	TYPE	SUBROUTINE	FACTOR OVER 7600	FACTOR OVER CRNV
(32 x 13)	CRNV	Time Adv.	3.16	1.
"	CRNV	F-P Coefs	3.44	1.
"	CRNV	Both	3.36	1.
"	CRV	Time Adv.	10.03	3.17
"	CRV	F-P Coefs	9.49	2.76
"	CRV	Both	9.63	2.86
(46 x 19)	CRNV	Time Adv.	3.22	1.
"	CRNV	F-P Coefs	3.39	1.
"	CRNV	Both	3.34	1.
"	CRV	Time Adv.	11.71	3.63
"	CRV	F-P Coefs	10.09	2.97
"	CRV	Both	10.53	3.15
(64 x 46)	CRV	Time Adv.	14.15	4.40
"	CRV	F-P Coefs	11.41	3.37
"	CRV	Both	12.18	3.65

Table 2. CRAY-1 Computation Time Per Mesh Point

MESH (kmax x imax)	Time (sec) Per mesh pt.
32 x 13	1.62×10^{-5}
46 x 19	1.37×10^{-5}
64 x 45	1.09×10^{-5}
65 x 45	1.15×10^{-5}
115 x 49	1.10×10^{-5}

REFERENCES

1. M. N. Rosenbluth, W. M. MacDonald and D. L. Judd, The Physical Review, Second Series, Vol. 107, No. 1 (1957), 1.

2. J. Killeen, A. A. Mirin, M. E. Rensink, Methods in Computational Physics, Vol. 16, (1976) 389-432.

3. A. A. Mirin, Lawrence Livermore Laboratory Report, UCRL-51615 Rev. 1 (1975).

4. R. W. Harvey, J. C. Riordan, J. L. Luxon, K. D. Marx, "Studies of Current Due to RF Induced Runaway in the DIIA Lower Hybrid Experiment" presented at the Annual Meeting on Theoreitcal Aspects of Controlled Thermonuclear Research, Mount Pocono, April 18-20, 1979.

5. A. A. Mirin, J. Killeen, K. D. Marx and M. E. Rensink, Journal of Computational Physics, Vol. 23, No. 1, (1977), 23.

6. R. D. Richtmyer and K. W. Morton, Difference Methods for Initial Value Problems, John Wiley and Sons, New York, (1967), 198.

*Work performed under the auspices of the U.S. Department of Energy by the Lawrence Livermore Laboratory under contract number W-7405-ENG-48.

THE CRAY-I AND MHD STABILITY STUDIES IN TOKAMAKS

J. Manickam
Princeton Plasma Physics Laboratory
Princeton University
Princeton, NJ 08544

ABSTRACT

The stability of tokamak plasmas in reactor-like configurations to linerarized magnetohydrodynamic modes is of great interest from the theoretical and practical point of view. Analytic studies can give general dependencies in various approximations. Numerical studies are necessary to describe the behavior of plasmas in realistic configurations of practical interest, and require large high speed computers. With the advent of advanced computers, such as the CRAY-1, such a study is now feasible. This paper reviews the history of the computational study of MHD modes and describes the implementation of one such code on the CRAY-I. It describes the impact of the increased memory and speed on the computational program to study the dependence of these instabilities on various parameters of interest.

INTRODUCTION

The aim of this paper is to review the impact of advanced computer systems, specifically, the CRAY-I on the computational study of the MHD - Stability of Tokamaks. In order to do this, it is important to first review the background of the computational problem. This is not meant to be an exhaustive review, which may be found elsewhere.[1, 2] Here we intend to provide a flavor of the problems associated with the computational study of MHD instabilities, and describe some of the interesting techniques used to overcome them. We then proceed to discuss the impact of the CRAY-I on one particular code.

In the first section we briefly describe the physics of the problem and define a physical model. In Sec. 2, we establish the numerical model and outline the main features of the code. In Sec. 3, we discuss the implementation of the code on the CRAY and compare it with the implementation on the CDC 7600. Section 4 contains the results in the context of their impact on the computational program to study the MHD Stability problem. In Sec. 5, we present a summary and observations on the future role of advanced scientific computers in the stability studies of tokamaks.

1. THE PHYSICAL MODEL

In the quest of achieving viable fusion reactors, the tokamak approach is considered to be one of the more promising. The tokamak is essentially a toroidal device, with a strong externally generated toroidal magnetic field. The magnetic field lines are given a helical twist by a poloidal magnetic field generated largely by the current flowing in the plasma in the torus. Figure 1 shows the geometry of a tokamak in schematic form. The plasma in such a device is subject to a host of possible instabilities, however stable regimes can be found, where the instabilities do not exist. Study of the nature of the instability often gives us a clue on how we might be able to control or supress it. Of these instabilities the most obvious and dangerous ones are the gross MHD instabilities, associated with balance of the various forces acting on the system. The complex geometry makes this a difficult device to study analytically. It is common to resort to approximations based on various ordering schemes. For example, one considers an ordering in aspect ratio, the ratio of major to minor radius. Another parameter used for ordering is β, the ratio of material to magnetic pressure supported by the system. These give analytic tools of some value, but we still lack an accurate description of realistic systems. Tokamaks of practical interest are of small aspect ratio, typically between 3 and 5. At these values toroidal effects appear, that have no analogue in the cylindrical limit. Further from the reactor point of view, the highest possible β is preferable. In fact an important aim of a computational program to study the MHD behavior of tokamaks, would be to determine the dependence of β_C, the largest stable β, on various parameters, such as the geometry, and the pressure and current profiles. Such a study would then permit the design of an optimal tokamak from the MHD stability point of view.

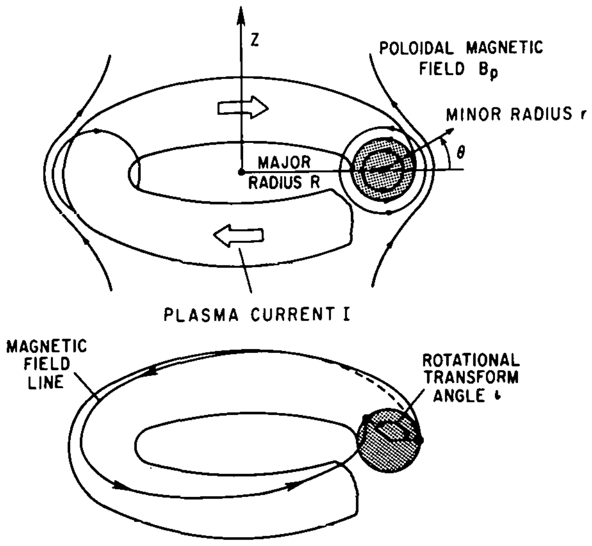


Fig. 1 A schematic diagram of a tokamak. The upper half shows the plasma current flowing clockwise, the toroidal field is opposed to it, and the lower half of the figure shows the resulting magnetic field line. The safetv factor $q = 2\pi/\iota$.

For numerical studies in tokamak systems, it has been shown that a relatively simple model, with the plasma treated as a perfectly conducting fluid, ignoring dissipative effects, such as resistivity, viscosity etc., is adequate to describe these instabilities. In such a model, known as the Ideal MHD model, we describe the system through the equations,[3]

$$\rho \frac{dv}{dt} = -\nabla p + J \times B$$

$$\frac{\partial B}{\partial t} = -\nabla \times E$$

$$E = -v \times B$$

$$\frac{\partial \rho}{\partial t} = -v \cdot \nabla \rho - \rho \nabla \cdot v$$

$$\frac{\partial p}{\partial t} = -v \cdot \nabla p - p \nabla \cdot v \tag{1}$$

these are essentially the Maxwell's equations, ignoring the displacement current, equations of continuity and motion. The fluid behavior also requires an equation of state and of course the necessary boundary conditions. These equations are usually linearized using a perturbation expansion about the stationary equilibrium defined by,

$$\nabla p = J \times B$$

$$J = \nabla \times B$$

$$\nabla \cdot B = 0 \tag{2}$$

The linearized equations of motion for the displacement vector ξ are given by,

$$\rho^o \frac{\partial^2 \xi}{\partial t^2} = \nabla(\xi \cdot \nabla p^o + \gamma p^o \nabla \cdot \xi) + (\nabla \times B^o) \times [\nabla \times (\xi \times B^o)] + \{\nabla \times [\nabla \times (\xi \times B^o)]\} \times B^o \tag{3}$$

the quantities with the superscripts are equilibrium quantities.

There are two commonly used techniques to solve these equations. One is basically the time advancement of the linearized equations of motion of the perturbed quantities. The second method is essentially a variational method based on an energy principle.

The time advancement method has been used extensively with considerable success.[4, 5, 6]

This method faces a formidable obstacle in the multiplicity of time scales associated with the linear MHD spectrum. These range over eight to ten orders of magnitude. Different methods have been used to solve this problem with considerable success. One technique involves eliminating the fast time scale analytically, generally accomplished by considering various limiting forms of the MHD equations. Another interesting technique has been to average over the fast time scale numerically by the use of a partially implicit scheme, on a dynamic grid. This method has the virtue of preserving phenomena on the fast time scale. These time dependent methods have the added advantage of being easily extended into the non-linear phase. For a more complete description of this approach, refer to the work of Carreras and Hicks in these proceedings.

The second approach uses the Lagrangian associated with the linearized perturbations, about an equilibrium configuration to formulate an eigenvalue problem. Reference 7 gives a fine comparison of three major codes of this class.[8, 9, 10] We describe here, the implementation of one of them,[8] the PEST (Princeton Equilibrium, Stability and Transport) code, on the CRAY-I.

2. THE NUMERICAL MODEL

In the Lagrangian approach, we seek normal modes of the system. We denote the displacement vector as ξ, then the Lagrangian L is given by

$$\xi(r, t) = \xi(r)e^{i\omega t}$$

$$L = \omega^2 K(\xi, \xi^*) - \delta W(\xi, \xi^*) \qquad (4)$$

where K represents kinetic energy, and δW the potential energy associated with the perturbation. We adop the Galerkin approach to the more general Rayleigh-Ritz procedure, in which the ξ's are expressed as a linear superposition of a subset of a complete set of functions.

$$\xi = \sum_m a_m \Phi_m$$

Then the variational calculation is reduced to the determination of the eigenvalues and eigenfunctions of the matrix eigenvalue problem,

$$\sum_m [\omega^2 \langle\Phi^*_{m'}|K|\Phi_m\rangle - \langle\Phi^*_{m'}|\delta W|\Phi_m\rangle]\, a_m = 0 \qquad (5)$$

This procedure has several nice features. It assures convergence from above, and therefore will not give instability in an essentially stable configuration, the numerical approximations involve only square integrable functions, so it avoids singularities. The choice of expansion functions is governed by the boundary conditions, e.g., the kinetic energy norm must remain finite, the normal component of the perturbed magnetic field must be continuous at the plasma-vacuum interface and vanish at the vaccum wall. For the expansion functions, Φ, we use Fourier series in the poloidal direction and finite elements in the radial direction.

The PEST code is composed of three parts, an equilibrium section which determines the equilibrium by solving the Grad-Shafranov equation obtained from Eqs. (2) above, which is essentially an elliptic partial differential equation for the poloidal magnetic flux, Ψ

$$x \frac{\partial}{\partial x} \frac{1}{x} \frac{\partial}{\partial x} \Psi + \frac{\partial^2 \Psi}{\partial_z 2} = 2\pi \times J_\phi \qquad (6)$$

This is solved by using double cyclic reduction in a form quite similar to the usual application to Poisson's equation in cylindrical coordinates. This in itself constitutes a major part of the problem. Figure 2 shows a typical configuration for a plasma with an essentially circular cross-section. The next step is to map the equilibrium flux function to a flux coordinate system, better suited to the decomposition of the expected eigenvectors. This mapping section is the second major part of the PEST package. Finally the matrix elements of the potential and kinetic energy are computed, and the matrix eigenvalue problem, Eq. (5) is posed. The matrices are real, symmetric and banded, a consequence of the Hermitian nature of δW, the symmetry, and the use of finite elements. This is solved either by a direct Gaussian elimination scheme to obtain the entire eigenvalue spectrum, if the rank of the matrix is small enough to fit into the available memory, or by an inverse iteration scheme, designed to determine only the lowest eigenvalues, which are in fact the most interesting. Extensive post processing facilities are available to examine the resulting eigenvectors.

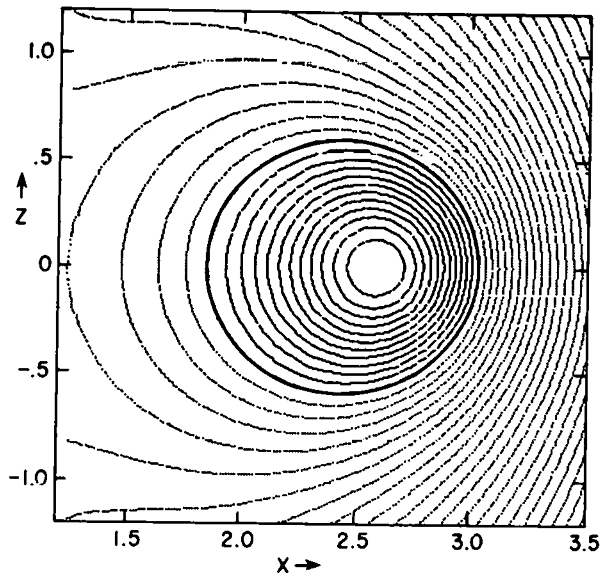


Fig. 2 A contour plot of the poloidal flux at equilibrium, showing a cross section of the torus. The dark solid line shows the position of the vacuum vessel.

3. THE IMPLEMENTATION ON THE CRAY

The PEST package is in an overlayed format on the CDC 7600. The three overlays consist of the three parts described in Sec. 2, the equilibrium, mapping and stability. Thus a single run covering all three parts can be executed in one job step. This is particularly convenient in simplifying the file management, as the data is in large part passed between overlays through disk, and the CPU requirements indicate that a batch type production run is most appropriate. Memory requirements are set by the computational grid dimensions in each part of the program. The appropriate figures are a 65 × 65 mesh for the equilibrium, mapping onto a 97 × 128 mesh in the mapping section. In the stability section the relevant parameters are the number of radial finite elements, which can be up to half of the number of surfaces, 97, and the number of Fourier modes permitted, typically about 30. The resulting eigenvalue problem involves matrices of rank ranging from 1500 to 4500. With this set of parameters, the code essentially uses all available memory on the CDC 7600. Running times are typically 5, 7, and 15 minutes, respectively, for the three parts. In practice it is necessary to make several stability runs in order to fit a convergence formula and extrapolate to the true eigenvalue. Thus a typical calculation to determine one such point would require about 75 minutes. To determine the dependence of a single instability mode on any given parameter would require several such runs, and in practice there are several parameters which play an important role in determining the stability of a given configuration. Thus it is impractical to attempt a serious parameter survey. Fortunately the CRAY provides, through its increased memory and speed a practical solution for this problem.

On the CRAY we found it convenient to break up the code into its three components. This was dictated in large part to the convenience of debugging, without having to worry about complications of the overlay structure. Since implementation, the dramatic change in CPU requirements has eliminated the original purpose of the overlay structure. The next major consideration was to improve the vectorization. The PEST code does not easily lend itself to vectorization, as say a time dependent code would. However there is room for optimization. This has been accomplished largely by increasing buffer lengths in the inverse iteration package for determining the unstable eigenvalues, eliminating logical test statements within inner loops, consolidating coding from functions and subroutines into single subroutines where possible, and reordering the internal storage to take advantage of the increased availability of memory and therefore reduce I/O. The increased memory of the CRAY permits finer computational meshes. The equilibrium can now be obtained on a 129 × 129 mesh, the mapping can used a 145 × 128 mesh, and the resulting matrices are of rank up to 6500. Running times on the CRAY with the modifications mentioned above are significantly lower. The equilibrium, mapping and stability sections, now require, 1, 1, and 1.5 minutes for comparable parameters respectively. Further, convergence improves with the increased dimensionality and the CPU requirements for one converged point drops to about 8 minutes from the 75 minutes on the CDC 7600. We discuss the impact of these changes in the next section.

4. IMPACT OF THE CRAY-I

In this section we discuss the impact of the CRAY on the study of Ideal MHD instabilities in tokamaks. Such a study would in large part reflect a parameter survey, with the aim of optimizing the stability of some idealized configuration. In practice there are several different kinds of instabilities, each requiring a slightly different computational approach. The main features identifying the instabilities are the toroidal mode number n, the helicity of the poloidal magnetic field expressed as a ratio of the toroidal magnetic field, through a quantity termed as the safety factor q. The poloidal mode numbers of interest then lie around m = nq. n takes values 0, 1, 2, 3,... etc., while q lies between 1 and 5 in configurations of interest. The ability to represent high m's is limited by the mesh size to be about 40. The coupling of the modes requires inclusion of several m's on either side of the dominant m. Thus with moderate q, we see that the largest n allowed is about 5 or 6. Independent analytic theory,[11] inspired in part by earlier studies from the PEST code for n = 3, has revealed that the high n limit is more restrictive for an important class of modes called ballooning modes. The analytic theory is for $n = \infty$, which leaves a large gap of intermediate values of n, which are at the present time difficult to study. In addition to n and q, the boundary conditions are important in determining the nature of the instability. Figure 3, shows an example of a particularly virulent n = 1, instability. The intricate structure, of the displacement pattern shown emphasizes the need for a fine mesh to resolve the details of the mode. This is only an m = 3 mode.

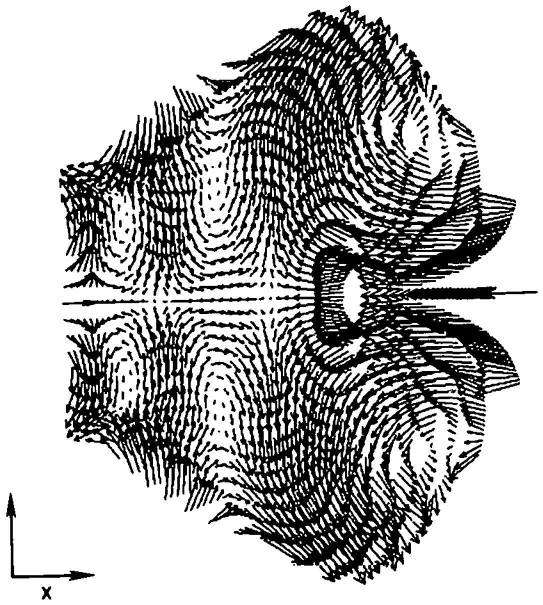


Fig. 3 A free surface kink instability for n = 1, q = 3. The arrows indicate the projection of the displacement vector field, onto the x-z plane. Note the complicated structure of the mode, which requires careful resolution with a fine computational mesh.

In transferring the PEST code to the CRAY, the major areas of change have been in the ability to use finer meshes, and the dramatically reduced CPU times. The importance of the grids is in determining the accuracy and variety of modes that can be study. In particular, near the marginal point, greater accuracy is needed and the singular behavior of certain modes requires finer meshes. This resulted in a moderate boost in the value of n that can be examined, and in the accuracy of determining marginal points. These are important effects but are not quite as dramatic as the impact of the reduced CPU requirements. The change of CPU requirements has made it possible to conduct extensive parameter surveys which have been hitherto impossible. In particular, at this point in time we have studied and understand the influence of various geometrical effects on the stability of the configuration.[12] This has in turn influenced the design of future tokamaks. The effects of other parameters, such as the current and pressure profile are much more difficult, and are now being studied. Further, it is now routine practice to use the code to examine specific configurations of interest for their stability properties. In this last context the code is being used as a design tool by the engineers and experimentalists.

5. SUMMARY AND OBSERVATIONS

We have reviewed the computational study of Ideal MHD instabilities in tokamaks. The main conclusion is that the nature of the problem is such that, while it is possible to use a machine such as the CDC 7600, the CRAY is really better suited. Thus the earlier studies on the 7600 should be viewed as preparation of the code. The numerical techniques are not particularly different from those on smaller machines, except that with the increased memory, and more important, speed of the CRAY, this field of study has matured. The discussion in Sec. 4 shows that even the CRAY is not adequate to simulate some of the instabilities. Different numerical approaches are being studied to complete the range of possible instabilities that are represented.

In this discussion we have restricted ourselves to consideration of linerarized ideal MHD effects only. This represents one important part of the physics of tokamaks. For completeness of an MHD study, we would have to introduce non-ideal effects and extend the study into the non-linear phase. This would best be done in a time dependent model. We have earlier alluded to the time scale problem that has to be overcome. Extension to three dimensions to account for non-axisymmetric effects adds another major degree of difficulty. Such codes are being developed, their effectiveness in realistic configurations remains to be demonstrated. In this area, even computers of the CRAY-I class are inadequate, and at best serve as a developing tool, even as the CDC 7600 served to develop the linearized ideal codes. Future, more powerful computers will be needed to fill the need in this area.

Finally we recognize that these MHD codes carry us through only a part of the time evolution of a plasma discharge in a tokamak. They consider the equilibrium, and the evolution of gross MHD instabilities from the linear to the non-linear phase. There remain other major areas of consideration. For example the slow evolution of the system between quasi-equilibria due to dissipation, is the subject of transport codes. These codes are also advancing, and are now capable of studying the two-dimensional evolution. With the increased speed of stability codes, we might consider coupling the two together. In practice one might advance the transport code to find a quasi-equilibrium and then examine its stability and continue. Such codes are under consideration. Efforts are also underway to couple transport effects into a time advancement MHD code. We should point out that this will not present the ultimate code to describe all the physics of the tokamak. Indeed given the complexity of the system and the various processes that need to be considered, such a code remains a near impossibility. However the future of MHD studies of tokamaks remains a promising and exciting field. The continued development of advanced computer systems assures continued progess in our understanding in this very important area of plasma physics.

REFERENCES

[1]R. C. Grimm and J. L. Johnson in Comp. Phys. Comm., 12, (1976) 45.

[2]J. A. Wesson in Nuclear Fusion 18, (1978) 87.

[3]I. B. Bernstein, E. A. Frieman, M. D. Kruskal, and R. M. Kulsrud in Proc. Roy. Soc., A244, (1958) 17.

[4]G. Bateman, W. Schneider, and W. Grossman in Nuclear Fusion 14, (1974) 669.

[5]A. Sykes and J. A. Wesson in Nuclear Fusion 14, (1974) 645.

[6]S. C. Jardin, J. L. Johnson, J. M. Greene, and R. C. Grimm in J. of Comp. Phys., 29, (1978) 101.

[7]M. Chance, et. al., in J. of Comp. Phys. 28, (1978) 1.

[8]R. C. Grimm, J. M. Greene, and J. L. Johnson in Meth. of Comp. Phys., Ed. J. Killeen, 16, (1976) 253.

[9]D. Berger, R. Gruber, and F. Troyon in Proc. of 2nd European Conference on Comp. Phys. (1976) Paper C3.

[10]W. Kerner and H. Tasso in Plasma Physics and Controlled Nuclear Fusion Research 1, (1974) 475.

[11]Dobrott, et. al., in Phys. Rev. Lett. 39, (1977) 943.

[12]A. Todd, et. al., in Nuclear Fusion 19, (1979) 743.

ATMOSPHERIC MODELLING

THIS PAGE

WAS INTENTIONALLY

LEFT BLANK

A VECTORIZED THREE-DIMENSIONAL OPERATIONAL TROPICAL CYCLONE MODEL

Rangarao V. Madala
Naval Research Laboratory
Washington, D.C. 20375

and

Simon Chang
JAYCOR
Alexandria, VA 22304

ABSTRACT

A three-dimensional numerical model to predict the intensification and movement of tropical cyclones is under development at the Naval Research Laboratory using TI-ASC* computer.

The physics of the model includes latent heat released in convective and non-convective clouds. The atmospheric boundary layer is parameterized using generalized similarity theory.

A newly developed fully vectorisable time integration scheme, split-explicit, is used to integrate the governing equations. In this scheme all the dependent variables are initially expanded in terms of the natural eigen modes of the model. The spectral equations governing the eigen modes are integrated using a time step which varies with each mode. These modes are then recombined at regular intervals of time to obtain the required solution. Use of this method has enabled us to reduce the computing time requirements by a factor of four compared to the conventional explicit schemes.

For a horizontally staggered 51x51x7 grid network with a horizontal resolution of 60 km, each computational cycle of the model requires 3.85 seconds when run in vector mode and 15.25 seconds when run in scalar mode.

INTRODUCTION

Operational forecast of tropical cyclones with numerical models has been moderately successful. One of several severe restrictions of operational models is the time requirement within which the model computation must be complete. Therefore, operational models either have poor spatial resolution or crudely parameterized physics. Current operational models of tropical cyclone which have limited skill on storm track forecast, have no skill on intensity forecast.

It is generally believed that the storm track cannot be determined solely by the mean flow. There are important interactions between the tropical cyclone and the mean flow. The internal structure of the tropical cyclone is an important factor in the interactions. To have a realistic model structure becomes a prerequisite for a good forecast model. Such models should have good spatial resolution and parameterized physics comparable to some research models.

With the development of the advanced computing systems, operational models with adequate spatial resolution and physics are feasible. One such model is now being developed at the Naval Research Laboratory.

With the vectorization of the code and a new temporal integration scheme, called split-explicit method, an efficiency of 16 is achieved compared with the same model using leapfrog method run on scalar machine. This efficiency enables the

*Mention of a commercial product does not imply endorsement.

final version of the NRL tropical cyclone to meet the Navy's requirement in an operational mode.

We will present in the following section governing equations, parameterized physics and computational aspects of the present version of the model. Model structure and some preliminary results from a series of hindcast will be discussed. And finally, we discuss the future development of the model.

GOVERNING EQUATIONS AND MODEL PHYSICS

The governing equations include the primitive conservation equations for horizontal momentum, mass, enthalpy, and water vapor. The system of equations is hydrostatic, a normalized pressure (sigma) is the vertical coordinate (Phillips, 1957).

The model physics include the subgrid-scale horizontal mixing, the cumulus convection, the large scale precipitation, and the subgrid-scale vertical mixing due to surface friction.

The subgrid scale horizontal mixing is parameterized by a kinematic eddy coefficient. This coefficient consists of a constant part and a part linearly dependent on wind speed (Anthes et al, 1971). This form of eddy coefficient yields suitable mixing in the initial as well as the mature stages.

The cumulus convection is parameterized following Kuo's (1974) method. Conditional instability and the boundary layer convergence of water vapor initiate the convection. Partitioning of heating and moistening depends on the relative humidity of the air column. The vertical distribution of heating is a function of the conditional instability. Large scale precipitation occurs when the air reaches saturation in large scale lifting.

The surface friction is parameterized based on a generalized similarity theory, in which logarithmic-linear profiles in the surface layer are "matched" into the mixed layer (Chang and Madala, 1979). Universal functions involved are formulated following Yamada (1976). Charnock's formula is used to compute the roughness of the ocean surface.

GRID NETWORK AND INTEGRATION SCHEME

The model atmosphere from P = Ps (surface) to P = 0 is divided into seven sigma layers. All prognostic variables (such as u, v, T, q) are defined at the center of each layer, all diagnostic variables (such as $\dot{\sigma}$ and ω) are defined at the boundary of each layer. The momentum points and mass points are staggered in horizontal directions following Arakawa scheme C.

The center-in-space finite differencing method is used for spatial differencing. Mass, momentum, and enthalpy in the model are conserved in finite differencing form.

For the temporal integration, a newly developed split-explicit method (Madala, 1979) is used. In this method, all the dependent variables are initially expanded in terms of eign modes of the model. The linearized spectral equations governing the eigen modes are integrated in leapfrog fashion using a time step which varies with each mode. Those modes are then recombined at regular intervals of time to evaluate the forcing function and nonlinear effects. The overall computing time are reduced by a factor of four compared with the conventional explicit methods.

There are 51 x 51 horizontally staggered grid points and seven vertical layers in current version of the model. The uniform horizontal resolution is 60 km.

INITIALIZATION PROCEDURE

A static initialization based on the nondivergent component of observed wind field is used. The procedures are as follows:

(a) Vorticity is computed from the observed wind.

(b) A Poisson equation is solved to determine the stream function. Nondivergent wind field is obtained from the stream function.

(c) A second Poisson equation is solved to determine the surface pressure based on the non-divergent wind.

(d) A three-dimensional Poisson equation is solved for the geopotential or temperature fields over the domain of the model.

Additional restriction on the lateral boundary is added in steps (a) and (b) to ensure there is no net mass inflow or outflow at all levels. In above, a direct fast elliptic equation solver (Madala, 1978) is used to solve the Poisson equations.

STRUCTURE OF THE MODEL TROPICAL CYCLONE

It is generally believed that the forecast of the paths of tropical cyclone is not just simply the transport of isolated vortices by mean flow. There are significant interactions between the tropical cyclones and the mean flow. The internal structures of the storm have great influence in these interactions. Thus, to have a realistic model structure becomes one prerequisite of a good forecasting model.

To examine the model structure at quasi-steady state, we integrate the model from a hypothetical asymmetric initial vortex embeded in a tropical atmosphere with zero large scale mean flow on an f-plane over 28.5 C ocean water. The model tropical cyclone, after the initial dissipation stage, and the development stage reaches quasi-steady state at about 60h. At 72h simulating time the minimum central pressure is 980 mb and the maximum wind is 40 m s^{-1}.

The surface pressure analysis shows an axisymmetric low pressure center with concentrated pressure gradients near the center (Figure 1).

Figure 2 shows the wind vectors and isotachs in the lowest layer of the tropical cyclone where the inflow and convergence is the strongest. The wind field is characterized by a mostly axisymmetric, cyclonic circulation with high wind velocities near the storm center. At the storm center, the velocity is nearly zero. The maximum wind speed occurs to the northeast and the southwest of the center and is believed to be associated with the strong convective band.

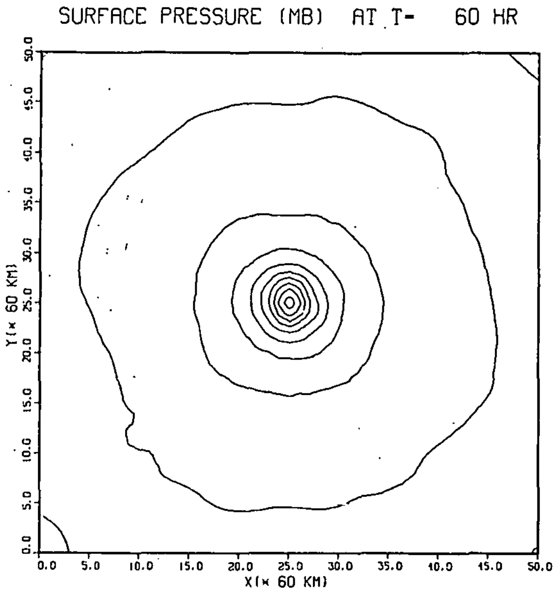


Fig. 1. Analysis of the surface pressure of a stationary tropical cyclone. The interval between isobars is 4 mb. The outmost circle is the 1016 mb isobar, the minimum pressure is 981 mb.

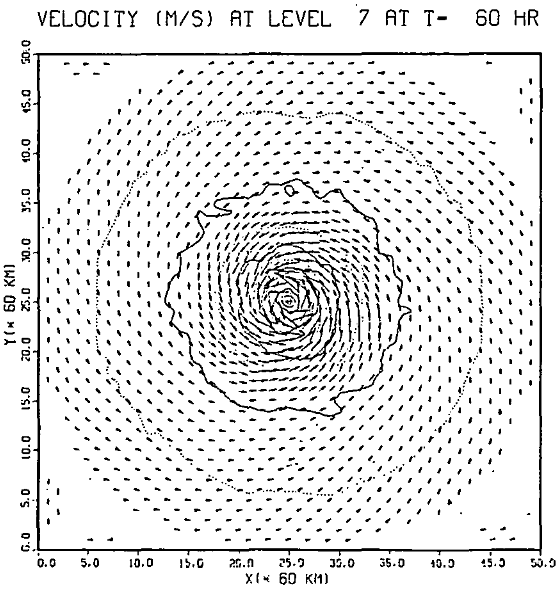


Fig. 2. Analysis of the isotach and the vectors of the surface wind field. The interval between isotachs is 5 m s^{-1}. The maximum wind is 41 m s^{-1}.

Figure 3 shows the wind field at the outflow level just beneath the tropopause. Contrary to wind field at the inflow level, the wind field at this level is highly asymmetric, in agreement with observation. It features weak cyclonic circulation near the center and anticyclonic with high speed circulation at large radii. The high anticyclonic wind speeds are caused by the conservation of angular momentum in the outflow supported by the conservation of angular momentum in the outflow supported by the warm core.

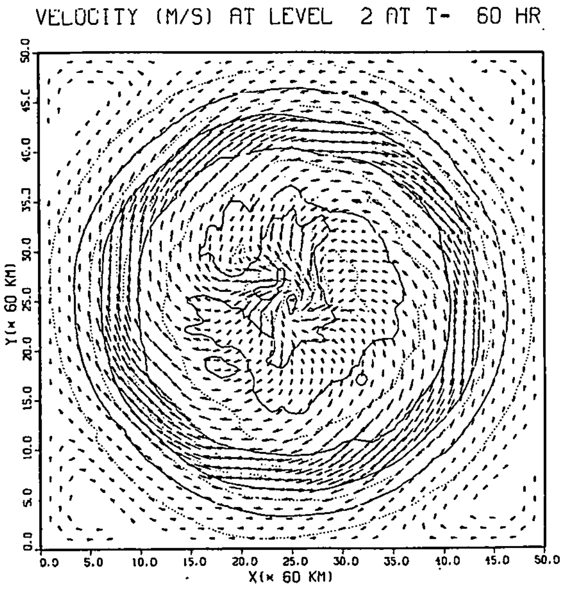


Fig. 3. Same as Figure 2 except at the outflow level.

The warm core character of the storm is depicted by the temperature change from the initial condition on an east-west cross-section through the storm center (Figure 4). The warm core, with maximum of 9°C at near 300 mb, is caused by the exceed of the diabatic heating over the adiabatic cooling. The cooling on top of the tropopause is due to the forced assending in a stably stratified environment and the geostrophic adjustment there.

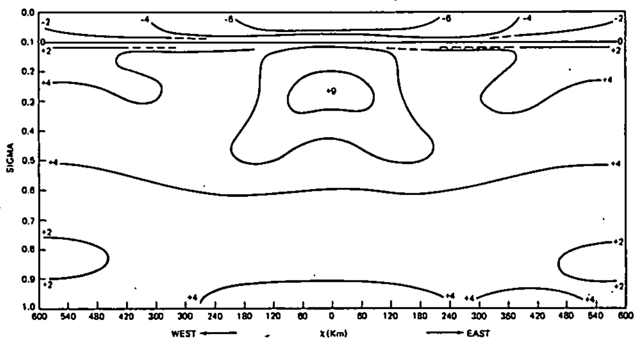


Fig. 4. The temperature change (^{o}C) on a eastwest cross-section through the storm center.

The structure of the eye and the eyewall is illustrated in Figure 5 by the relative humidity (RH) on the same cross section as in Figure 4. The RH field features (1) a very dry eye region due to the descending motion, (2) relatively moist eyewalls due to convection, (3) very moist inflow due to sea-surface evaporation, (4) very moist outflow layer, and (5) relatively dry troposphere outside the eyewalls due to general descending motion.

In general, the structure of the model tropical cyclone is very realistic as compared with the observations.

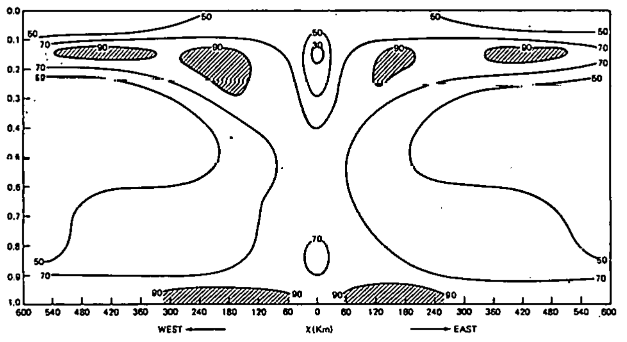


Fig. 5. Same as Figure 4 except for the relative humidity (%).

HINDCASE PERFORMANCE

A series of real data experiments have been carried out on ten typhoon cases in the 1976 and 1977 seasons to test the forecast skill of the model. Two current operational models, the Movable Fire Mesh Model (MFM) of the National Meteorological Center and the One-Way Interaction Model (OI) of the Fleet Numerical Weather Prediction Center, are used for comparison.

Table 1 lists the averaged vector position error (VPE) for these ten typhoon cases. It is clear that the present model has much less vector position errors at 24 and 36 hr. The deterioration of the forecast of our model at 48 hr is caused by the ad hoc boundary conditions of our model as the typhoons move close to the boundaries. Unlike the mesh of the other two models in comparison, the mesh in the present model is not moveable and is not nested into a coarse grid. An effort to implement a moveable nested grid work is now being undertaken.

Table 1. Averaged Vector Position Error (n.m.)

	12h	24h	36h	48h
MFM	63	130	158	200
OI	81	123	143	163
NRL	66	91	112	217

The persistance of our model forecast is illustrated by the standard deviations in the VPE for the 10 typhoon cases (Table 2).

Table 2. Standard Deviations of the Vector Position Errors

	12h	24h	36h	48h
MFM	109	221	257	318
OI	102	196	194	261
NRL	83	113	133	245

SUMMARY

A three-dimensional numerical model of tropical cyclone is under development at the Naval Research Laboratory. This model, when fully developed, will satisfy the Navy's needs in operational forecast of tropical cyclone.

As shown, the present version of the model is capable of producing a realistic structure of tropical cyclone. A series of test runs shows the model has greatly reduced the averaged vector position errors in storm track forecast. Combining the utilization of a new temporal integration scheme and the vectorization of the computer code, an efficiency over conventional models using conventional computers of 16 is achieved. A 72 hr integration of the present version requires 50 min. of CPU time on a TI-ASC.

Before becoming fully operational, there are several areas in the model that need to be improved. One of the most important improvements will be the implementation of a moveable nested gridwork. It is planned to construct a grid network that has spatial resolutions of 20, 60, and 180 km. The two inner grids will be able to move with the tropical cyclone so that the center of vortex is never far away from the center of the two finer grid networks. Such a nested grid work will give good enough resolution at the finest grid for the detailed structure near the storm center while providing good interaction at the coarser grid with the synoptic scale flow.

ACKNOWLEDGEMENTS

We thank Mr. Richard Hodour of the Naval Environment Prediction Research Facility for running the real data tests. Mrs. Jane Polson typed the manuscripts.

The development of the model is supported by grants ONR RR033-02-44, NAVAIR 9F52-551-792, and NRL N00173-78-C-421.

REFERENCES

Anthes, R. A., J. W. Trout, and S. L. Rosenthal, 1971: Comparisons of tropical cyclone simulations with and without the assumption of circular symmetry. Mon. Wea. Rev. 99, 759-766.

Chang, S. W., and R. V. Madala, 1979: Use of similarity theory to parameterize the PBL of tropical cyclone. NRL Technical Memorandum.

Kuo, H. L., 1974: Further studies of the parameterization of the influence of cumulus convection on large scale flow. J. Atmos. Sci., 31, 1232-1240.

Madala, R. V., 1978: An efficient direct solver for separable and non-separable elliptic equations. Mon. Wea. Rev., 106, 1735-1741.

Madala, R. V., 1979: Computationally efficient time integration methods. Fourth Conference on Numerical Weather Prediction of the American Meteorological Society, October 1979, Silver Springs, Maryland.

Phillips, N. A., 1957: A coordinate system having some special advantages for numerical forecasting. J. Meteor., 14, 184-185.

Yamada, T., 1976: On the similarity functions A, B, and C of the planetary boundary layer. J. Atmos. Sci, 33, 781-793.

IMPLEMENTATION OF VECTORIZING TECHNIQUES ON THE CDC-STAR-100 FOR SPEED ENHANCEMENT OF GLAS GCM

Lawrence Marx
Sigma Data Services Corporation
c/o NASA/Goddard Space Flight Center
Code 911, Building 22
Greenbelt, Maryland 20771

ABSTRACT

The Goddard Laboratory for Atmospheric Sciences nine-level global general circulation model with variable horizontal grid resolution has been converted to run on the CDC STAR 100 at NASA Langley Research Center. Vectorization programming has been employed to achieve enhanced computational performance including: (i) dynamic equivalencing; (ii) a generalized development of vector expansion and masking techniques for dealing with cloud distributions and other scalar dependent processes; (iii) the use of high-speed vector kernel functions. These program enhancements have resulted in an overall 4.5 speed factor improvement over the same code run on the Amdahl 470V/6 at the Goddard Modeling and Simulation facility.

THE USE OF THE CRAY-1 IN SIMULATING HAIL GROWTH

C. M. Berkowitz
Battelle, Pacific Northwest Laboratory
P. O. Box 999
Richland, WA 99352

ABSTRACT

A two-dimensional (x,z) Monte Carlo model of the hail growth zone was used to investigate the effects of updraft tilt and width on hail production. To allow for selection processes necessary to resolve the difference between the concentration of hailstone embryos and the concentration of hailstones, the growth of a large number of embryos had to be simulated. Development of this model on a CDC-7600 computer required extensive tape and mass storage buffer operations to model the many growing particles. By modifying the program to run on the newly available Cray-1 computer, most of the I/O operations were no longer necessary, and computation times were greatly reduced. This allowed for a more extensive investigation than would otherwise have been possible.

INTRODUCTION

The large number of computations required to simulate the growth processes of hailstone embryos have greatly restricted the investigation of hail development through the use of mathematical models. By use of fairly sophisticated I/O software, the CDC-7600 computer could be used to model such interactions in simple two-dimensional (x,z) models. With the advent of such systems as the Cray-1, cloud physicists now can spend more time on modeling the physics, and less time having to develop software processes which circumvent memory and speed limitations of standard computers.

At least two categories of physical processes have to be identified when describing the growth of hailstones. There is, of course, the growth of the individual embryo, which is described by collection and heat budget equations. And, of equal importance, there are processes resulting from the presence of other embryos within the cloud, competing with each other for the available liquid water required for growth.

Development of the simple kinematic flow model of hailstone growth which is described in this paper began on a CDC-6400 at the University of Arizona, where extensive use was made of time-consuming mass storage read and write statements. Work on the 6400 was, however, directed primarily at developing a working FORTRAN code which modeled the microphysics of individual particles. Only one embryo per time step could be processed, so there was effectively no depletion of the liquid water field and no interaction among developing hailstones.

The second phase in the development of the hail growth zone model was done on the NCAR CDC-7600, a larger, much faster machine. At NCAR, hundreds of groups of embryos could be simulated per time step and processed, although core requirements still required buffering intermediate blocks of data to disk, which would then be buffered back later in the program when they were needed for further computations. Using a series of such buffer manipulations, it was possible to run the model for some of the more simple cases of interest, and barring hardware problems, obtain results within a day or two.

Even in the 7600 though, core limitations forced development of a hail growth model consisting of two programs; one to simulate the microphysics, and a separate code to actually evaluate the data. The first program produced a tape containing the size, density, and coordinates of hailstones that were outside the model grid. A second program would perform an assortment of statistical tasks on this first data set.

The hail growth model was probably one of the first programs to be modified for operational use on the NCAR's Cray-1 system. Using the Cray, the hail growth program and the analysis program could not only be combined into one deck (all of the development was done in batch mode) but no tapes or disk space was required to circumvent the considerable memory overflow problems encountered when running the Hail Growth Analysis Packet on the 7600. The turnaround time went from days to minutes, making refinements in the code for the microphysics orders of magnitude easier to develop than they had been previously.

Given a computer of infinite memory capacity and speed, each hailstone could be individually modeled. Until such descendents of the Cray-1 are developed though, cloud physicists must be content with processing groups of hailstones that are defined by a common size, temperature, density, location, and which contain a specified number of particles per unit volume. However, because any one particle within these groups is treated in exactly the same fashion as any other particle, they will all compete for liquid water on an equal basis--this is a result of the grouping process, and has no physical justification.

In a series of experiments to investigate the results of this numerical competition, Young[1] found that for a 60 m by 60 m set of grids (defined by a homogeneous liquid water content), the maximum number of particles that could be contained in a group was $100/m^3$. With a concentration greater than $100/m^3$ per group, numerical competition was found to significantly decrease the resulting mean hailstone size of particles in a group.

By limiting the group concentration to $100/m^3$ and specifying a 40 m x 40 m grid that was assumed to be 1 m deep, up to 160,000 particles per grid could compete for the liquid water. Using the Cray-1, up to 70,000 groups per time step (spread throughout the model) could be processed.

REVIEW OF THE PHYSICAL PROBLEM

THE HISTORY OF A HAILSTONE

Observations of Knight and Knight[2] suggest that the majority of hailstone embryos in northeast Colorado are graupel; that is, ice crystals which have collected enough super-cooled water drops to obscure the crystals' original structure. These embryos are thought to begin their development as hailstones within a convective cell in updrafts that are just strong enough to buoy them up but not so strong that they are lifted out the top of the cloud. As the embryos grow, they are thought to enter regions of stronger updrafts having higher liquid water content (between 3 and 7 g/m^3) than the weaker updrafts. Because liquid water is a prerequisite to hailstone growth, which is primarily an accretion process, temperatures in this zone of strong updrafts must be less than -40°C to avoid homogeneous freezing which, in turn, would effectively remove liquid water available to the growing embryos. An upper temperature limit of approximately -20°C can also be specified for this zone of strong updrafts since warmer temperatures are more likely to result in "soft" or spongy hail, which would probably melt before reaching the ground.

This highly simplified conceptual model of embryos developing initially in a zone of weak updrafts (called the Embryo Formation Region, or EFR) and undergoing most of their growth in a zone of strong updrafts having more liquid water and a fairly well defined temperature range (called the Hail Growth Zone, or HGZ) is consistent with observations made during the National Hail Research Experiment (NHRE)[3,4], and the Alberta Hail Study (ALHAS)[5]. It can apply to short-lived convective systems by adopting a time sequence for the transition from EFR to HGZ. It can also apply to a longer lasting supercell with a fairly steady-state circulation. In this latter case, the EFR and HGZ would be spatially contiguous.

THE HEAT BUDGET OF A HAILSTONE

There are a few studies in cloud physics which do not draw heavily on the fundamental concepts of classical thermodynamics. The theoretical development of hailstones is no exception, and a heat budget approach is the basis of a quantitative theory of hailstone growth first formulated by Ludlam[6] and reviewed briefly here.

The temperature of a hailstone will be determined by a balance between four factors. First, sensible heat will be lost to the environment by the hailstone. Assuming a spherical particle of radius r_h, a constant thermal conductivity of air of K, and a factor to account for the anisotropy of the temperature field the particle is falling through, b, then the heat transfer rate, dQ_s/dt, will depend on the temperature gradient,

$$\frac{dQ_s}{dt} = 4\pi b r_h{}^2 K \frac{dT}{dr} \quad . \qquad (1)$$

By assuming a constant heat transfer rate, and evaluating dT/dr at r_h, this integrates to

$$\frac{dQ_s}{dt} = -4\pi b r_h K \; (T_\infty - T_h) \quad , \qquad (2)$$

where T_∞ is the environmental temperature, and T_h is the hailstone's surface temperature.

Similar arguments for the mass flux away from the hailstone can be used to derive an expression for the water loss by sublimation with resultant cooling:

$$\frac{dQ_m}{dt} = L_s \; Dc(\rho_{s,h} \cdot \rho_\infty) 4\pi r_h \quad , \qquad (3)$$

where L_s is the latent heat of sublimation, D is the diffusivity of vapor in air (assumed constant), c is a mass ventilation factor, and the ρ's refer to the saturation vapor density of water at the hailstone's surface ($\rho_{s,h}$) and the environmental vapor density (ρ_∞).

Heat is gained by the accretion process, whereby supercooled water droplets (occurring with number density χ) freeze onto the hailstone, releasing their latent heat of fusion, L_f. This heat source is described by a collection equation for collector particles of radius r_h, moving with speed V_h, colliding with smaller particles of size category j and having a velocity V_j:

$$\frac{dQ_a}{dt} = L_f \; \pi r_h{}^2 \; (V_h - V_j) \; \chi_j \qquad (4)$$

where it has been assumed that all particles in the sweep path of the hailstone are collected.

The last term in this heat budget expression is one describing the exchange in sensible heat between the hailstone and the accreted droplets,

$$\frac{dQc}{dt} = \pi r_h^2 \; (V_h - V_j) \; \chi_j \; (T_h - T_j) \; C_w \; . \qquad (5)$$

These four expressions involve temperature as a function of heat transfer rate, fall speed (also a function of size and air density, among other factors), liquid water content and latent heats (which are functions of particle temperature). The growth of a hailstone from a small graupel particle is obviously related to the environmental characteristics and cloud height, which will be continually changing. Reasonable values for the ambient temperature, vapor density, and liquid water field must be simulated, in addition to vertical and horizontal winds. Hailstones that develop in an environment where

$$\frac{dQ_s}{dt} + \frac{dQ_m}{dt} > \frac{dQ_a}{dt} + \frac{dQ_c}{dt} \qquad (6)$$

(sensible heat loss + sublimal cooling > heat released by freezing of droplets + sensible heat transfer by droplets)

will loose heat fast enough to allow solid ice to form on the surface, producing a hailstone with a different density than one where the heat budget is described by a reversed inequality sign in Eq. (6).

With so many interacting factors, it is difficult to see how a population of hailstones can be treated as a homogeneous group. This, of course, necessitates individual processing of embryos, which in turn necessitates use of computers such as the CDC-7600 and Cray-1.

COMPETITION AMONG HAILSTONES AND STOCHASTIC PROCESSES

Early computer models of hail growth[7] have assumed smooth trajectories of particles moving through the hail growth zone that are basically determined by the initial size and location of the embryo. With the addition of a perturbation component to the mean velocity of a particle, a unique trajectory for a given size particle beginning growth at a particular location no longer exists. This mixing can move larger particles from low, more favored trajectories. Therefore, analysis of model results must be done in a statistical manner, and hence the need for a second program to interpret the results of the microphysical model on the CDC-7600.

Competition for available liquid water has been suggested as a mechanism for producing the interesting phenomenon of "size-sorting". By simulating depletion of cloud water by hailstones, Young[8] found that a high embryo concentration would result in a "most favored trajectory" for hailstone growth that was quite low and short in length when compared to "least favored trajectory" that would be higher and longer, but would allow a hailstone to encounter less liquid water due to depletion by many lower trajectories. In this situation, large embryos would have a greater chance of developing to hailstones since they would have a more likely chance of following a lower trajectory. This process would result in the larger embryos falling out first--a phenomenon Young called "negative size sorting". With low embryo concentrations, a higher trajectory would be the "most favored", since there would be little depletion of liquid water; thus, smaller embryos would have a tendancy to produce bigger hailstones and these hailstones would fall out further from the EFR by virtue of their longer path; this has been labelled "positive size sorting".

Of importance here is that size-sorting and "most favored trajectory" are a result of the stochastic processes and liquid water depletion. These two features, Young concludes, must be included among the interactive effects of hailstone growth.

MODEL RESULTS AND CONCLUSION

By modeling heat budgets, perturbation velocities and hailstone interaction (via competition for liquid water), the effect of updraft width and tilt on hailstone size and total mass was investigated. Quantitative values of size and mass were derived, but because of the extreme complexity of the actual physical system under consideration, with all its many feedback processes, direct comparison of model results with field observations is difficult at best, and more probably impossible at this stage of hail growth models. The results are most useful in helping to define isolated mechanisms of hail growth.

The tilt (tangent of the updraft angle from the vertical) and width of the updraft were found to have significant effects on the total hail mass produced and on the sizes of hailstones. Because a very broad updraft could buoy a developing hailstone up longer than a narrow one, and an updraft that was more strongly tilted would carry a hailstone through an updraft core faster than a more vertical one, it was not surprising to find that the maximum and average hailstone size increased linearly with the ratio of updraft width to tilt, W/T (Fig. 1). A similar linear relationship between hail mass produced by the HGZ and W/T was also found for $W/T \leq 5$ (Fig. 2). By plotting the cumulative frequency of hailstone size for assorted ratios of W/T (Fig. 3), the model was found to decrease the total number of hailstones for values of $W/T > 2$. However, since the mean hailstone size continues to increase with W/T, the effects of increasing hailstone size and decreasing hailstone numbers appear to balance, producing a roughly constant hail mass. Without simulating depletion of the liquid water field by hailstones of assorted sizes, such results would not occur, and the hailstone mass and size would continue to increase with W/T values.

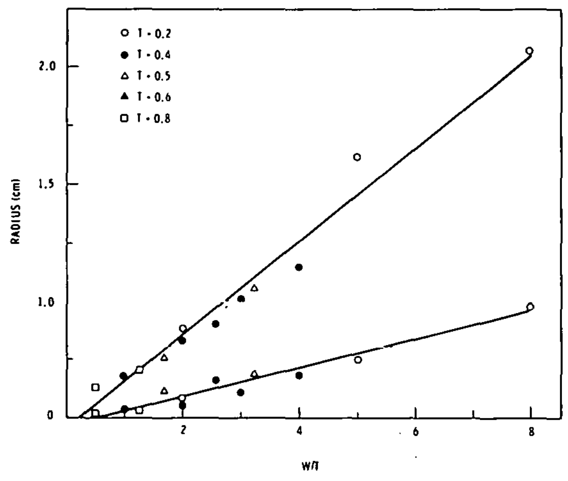


Fig. 1. Maximum and average hailstone radii as a function of the ratio of updraft width to tilt (W/T). Lines of best fit for maximum and average radii are also shown ($r^2=0.97$ and 0.95 respectively). Maximum updraft strength was 25 m/s in all cases.

Sulakvelidze[9] has proposed that the largest hailstones to be produced by the HGZ will be those having fallspeeds equal or greater than the maximum updraft velocity. By modeling a tilted flow field and including random horizontal displacements, hailstones appear to be able to pass through the updraft core without ever achieving a balance with the maximum updraft. Should this "tunneling" phenomenon be found to exist, then nomograms forecasting maximum hailstone size based on updraft strength, and temperature at which the maximum updraft occurs[9,10,11] would require the addition of updraft tilt and width in order to be complete.

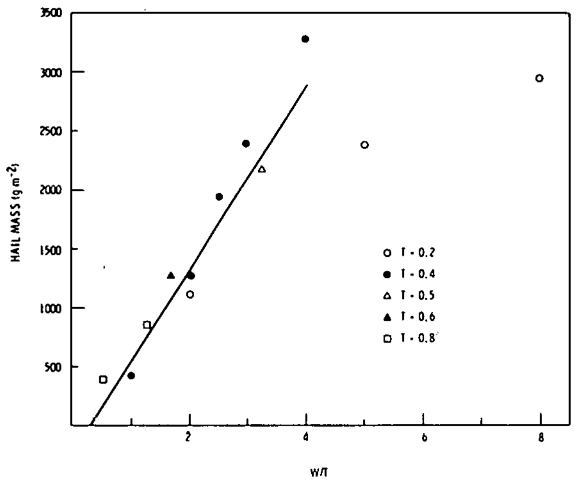


Fig. 2. Total hail mass per m^2 deposited on the ground as a function of the ratio of updraft width to tilt (W/T). The total hail mass is the amount of hail that would be left on the ground by a storm which had passed overhead, moving at 10 m/s. The total mass demonstrates a linear dependence on W/T for W/T < 5 (r^2=0.96).

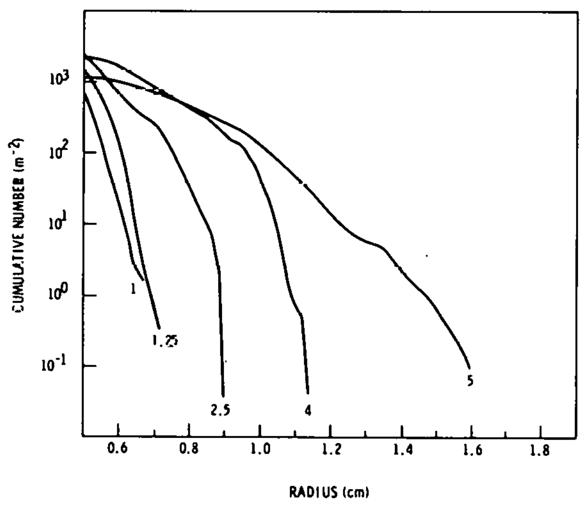


Fig. 3. Cumulative number frequencies for five different combinations of updraft width and tilt. All cases had a maximum updraft of 25 m/s.

With the memory and speed of the Cray-1, it may be possible to extend the two-dimensional model to a third dimension, allowing for recirculation of hailstones. Also, an updraft profile that is time and height dependent can now be considered, as can an even more complete treatment of microphysical processes. Such refinements, which are only now possible, would greatly facilitate studies into hailstone trajectories, feedback mechanism between hailstones and updrafts, and the feasibility of cloud seeding for hail suppression.

ACKNOWLEDGEMENT

Funding for the development of the variable updraft Monte Carlo model was provided by NSF Grant SUB NCAR 55004 under the National Hail Research Program. Current application and related research was conducted on the MAP3S Scavenging Program at Pacific Northwest Laboratory and is supported by the U.S. Department of Energy under contract EY-76-C-06-1830.

The author would like to thank Dr. Bryan Scott for his useful comments on the original draft of this paper.

Dr. Ken Young of the University of Arizona was principle investigator for NSF Grant Sub NCAR 55004, under which this work was done. Dr. Young's support and guidance are very much appreciated.

REFERENCES

1. Young, K. C., 1978b: On the role of mixing in promoting competition between growing hailstones. J. Atmos. Sci., 35:2190-2193.

2. Knight, C. A., and N. C. Knight, 1970: Hailstone embryos. J. Atmos. Sci., 27:659-666.

3. Browning, K. A. and G. B. Foote, 1976: Airflow and hailgrowth in supercell storms and some implications for hail suppression. Quart. J. Roy. Meteor. Soc., 102:499-533.

4. Browning, K.A., J.C. Fankhouser, P.J. Chalong, P.J. Eccles, R.G. Straoch, F.H. Merrem, D.J. Musil, E.L. May and W.R. Sand, 1976: Synthesis and implications for hail growth and hail suppression. Structure of an evolving hailstorm. NHRE Tech. Rep. 71/1, Il/22.

5. Chisolm, A.J., 1973: Alberta Hailstorm: Part I. Radar case studies and airflow models. Meteor. Monog. 36:1-36.

6. Ludlam, F.H., 1958: The hail problem. Nubila, 1:1-12.

7. Musil, D. J., 1970: Computer modeling of hailstone growth in feeder clouds. J. Atmos. Sci., 27:474-482.

8. Young, K. C., 1978a: A numerical examination of some hail suppression concepts. Meteor. Monog., 38:195-214.

9. Sulakvelidze, D. K., N. Sh. Bibilashvili, and V. F. Lapcheva, 1965: Formation of Precipitation and Modification of Hail Processes. Program for Scientific Translations, Jerusalem, Israel, 208 pp.

10. Diebert, R. J., 1976: Alberta hail project field program, 1975: Alberta weather modification Board, Three Hills, Canada. 67 pp.

11. Dennis, A. S., and D. J. Musil, 1973: Calculations of hailstone growth and trajectories in a simple cloud model. J. Atmos. Sci., 30:278-288.

DEVELOPMENT OF A STAR-100 CODE TO CALCULATE A TWO-DIMENSIONAL FAST FOURIER TRANSFORM

Jay Lambiotte
NASA
Langley Research Center
MS/125
Hampton, VA 23665

ABSTRACT

This paper describes the development of a computer code to perform a two-dimensional fast Fourier transform (2-D FFT) for real data on the STAR-100. Since the 2-D transform can be computed by performing successive 1-D transforms, the code has been built around an existing STAR 1-D FFT subroutine. Much of the complexity of this development effort has resulted from the STAR hardware requirements for vectors to be sufficiently long and to reside in contiguous memory locations, from the need to compute the transform of large data sets which can exceed the available central memory of the STAR, and from the desire to take advantage of the real property of the data. These particular requirements are discussed and STAR-100 timing results are presented.

INTRODUCTION

The need to compute a two-dimensional discrete Fourier transform (DFT) arises in a variety of applications such as data analysis, image processing[1], and spectral approaches to solving partial differential equations[2]. This paper describes the author's effort to develop a two-dimensional fast Fourier transform (2-D FFT) for the STAR-100 vector processing computer. The 2-D FFT can be viewed as successively performing the 1-D FFT of each of the rows of data from a grid followed by the 1-D FFT of the resulting column information. Consequently, if a 1-D FFT code is available, the 2-D code is conceptually simple. However, when one considers the vector processing characteristics of the STAR-100, the task becomes more complicated, especially in light of the computational requirement to transform both row and column information. In addition, the desire to minimize the CPU time for a given computation can, at times, be in conflict with the desire to minimize the amount of page faulting by the virtual memory system of the STAR-100 and vice-versa.

The code, FFT2DR, described here is a compromise between these two objectives. It is primarily directed at minimizing the CPU time for small grids which require no paging (up to 128 × 128) but with enough virtual memory considerations to permit problems 4 to 16 times that number of points to be computed with a moderate amount of paging. The extra software complexity does not increase the CPU time more than a few percent. Alternative vectorizations, which could significantly reduce the paging at the expense of increasing the CPU time, will be mentioned.

This paper will first examine the equations and computational requirements for computing a 2-D FFT. Then the specific effects of the STAR-100 hardware on these requirements are examined. FFT2DR is described next followed by results of some numerical experiments with the code and suggestions for alternative vectorization.

2-D DFT EQUATIONS

The 2-D Discrete Fourier Transform can be expressed as

$$F_{j,k} = \frac{1}{\sqrt{MN}} \sum_{p=0}^{N-1} \sum_{q=0}^{M-1} f_{p,q} \, W_N^{pj} \, W_M^{qk} \qquad (1)$$

where

$\tilde{f}_{p,q}$ are complex data defined for $p = 0, 1, .., N-1$ and $q = 0, 1, .., M-1$.

$W_M = e^{2\pi i/M}$ and $W_N = e^{2\pi i/N}$

$F_{j,k}$ are the complex transform of $f_{p,q}$ and are defined for $j = 0, 1, .., N-1$ and $k = 0, 1, .., M-1$.

The inverse discrete Fourier transform (IDFT) can similarily be expressed as

$$f_{p,q} = \frac{1}{\sqrt{MN}} \sum_{j=0}^{N-1} \sum_{k=0}^{M-1} F_{j,k} W_N^{-pj} W_M^{-qk}$$

with the obvious changes in the given definitions. It is easy to see[1] that Eq. (1) can be evaluated in the two steps:

$$\bar{F}_{p,k} = \frac{1}{\sqrt{M}} \sum_{q=0}^{M-1} f_{p,q} W_M^{qk} \qquad (2)$$

for all p,k

and

$$F_{j,k} = \frac{1}{\sqrt{N}} \sum_{p=0}^{N-1} \bar{F}_{p,k} W_N^{pj} \qquad (3)$$

Now, for each of the N values of p, Eq. (2) is a 1-D DFT of the pth row of grid values, $f_{p,q}$, and for each of the M values of k, Eq. (3) is a 1-D DFT of the kth column of grid values of $\bar{F}_{p,k}$. Consequently, the computation to be done involves N independent FFT's over the row of data followed by M independent FFT's over the columns of intermediate data.

When the input data are real, the storage and computation can be cut in half using the following well-known procedures[3]:

PROCEDURE 1

Given two real sets of data x_j, y_j, $j = 0, 1, .., N-1$ with transforms X_j, Y_j, $j = 0, 1, .., N-1$, let $u_j = x_j + iy_j$. Then use a standard FFT subroutine to compute $U_j = X_j + iY_j$. Given the fact that x is real if, and only if, X is conjugate even ($X_j = \bar{X}_{N-j}$), X and Y can be recovered through the equations

$$X_j = \frac{1}{2}(U_j + \bar{U}_{N-j}) \qquad (4)$$

and

$$Y_j = \frac{1}{2}i(\bar{U}_{N-j} - U_j) \qquad (5)$$

Since X, Y are conjugate even, only X_j, Y_j, $j = 0, 1, ..,(N/2) + 1$ need be stored or computed.

PROCEDURE 2

Similarly, given two complex data sets, X and Y, which are conjugate even, the IDFT can be computed by forming for $j = 0, 1, .., N/2$

$$U_j = X_j + iY_j \qquad (6)$$

$$\bar{U}_{N-j} = X_j - iY_j \qquad (7)$$

After the IDFT of U,

$$x_j = \text{Real}\ (u_j) \qquad (8)$$

$$y_j = \text{Imag}\ (u_j) \qquad (9)$$

PROCEDURE 3

Consequently, given a 2-D array of real values with M rows and N columns, the following steps can be used to compute the 2-D FFT assuming M and N are even:

(1) Perform[3] M/2 1-D complex FFT's of length N letting row i be the real part of a complex data set U_j and row i + 1 be the imaginary part for i = 1, 2, .., M/2.

(2) Recover the M transforms using Equations (4) and (5) storing only the first N/2 + 1 components of each transform.

(3) Perform (N/2) + 1 independent FFT's of size M from the columns of transformed data.

The inverse 2-D FFT can be computed by:

(1) Perform (N/2)+1 1-D complex inverse FFT's of length M over the columns of

transformed data.

(2) Using Eqs. (6) and (7) on the rows resulting from step (1), form M/2 complex data sets of size N to be inverse transformed.

(3) Perform M/2 FFT's of size N.

(4) Recover the real data using Eqs. (8) and (9).

CONSIDERATIONS FOR THE STAR-100

The programming considerations for Procedure 3 become more complex when one considers the architecture of the STAR-100. The characteristics of the STAR-100 most relevant to coding a 2-D FFT are:

CPU SPEED IS A FUNCTION OF VECTOR LENGTH

The CPU efficiency on STAR increases as a function of vector length due to the startup time associated with each vector operation. A vector operation, op, of length N has a startup time, S_{op}, and result rate, α_{op}, related to the total time, T_{op}, by

$$T_{op} = S_{op} + \alpha_{op} * N \qquad (10)$$

where all times are in units of the STAR-100 40 nanosecond minor cycle. For the vector multiplication[4], $S_* = 159$ and $\alpha_* = 1$. For the addition, $S_+ = 69$ and $\alpha_+ = \frac{1}{2}$. The vector length obviously has a significant effect on the overall result rate. For example, a vector addition of length 60 generates 15 million results per second, one of length 3600 generates 48 million results per second and in the limit as the vector length increases, 50 million results per second can be achieved. From this it is clear that a vectorized FFT algorithm that has vector lengths proportional to only one dimension of a grid or the other will not be particularly fast.

Two STAR subroutines exist at NASA's Langley Research Center on the STAR Math Library to perform 1-D FFT's. The first, Q4FFTV, is designed to perform the FFT of one, or a few, long data sets.[5] Its vectors are proportional to N, the size of the data set. The other subroutine, Q4FFORMS, assumes there are M independent FFT's to do and achieves average vector lengths[5] of $\frac{1}{2}$ M $\log_2$ N. Table 1 demonstrates the difference in the two subroutines. For the present application Q4FFORMS is superior. It assumes that the M independent data sets of N complex values are stored in 2*M*N contiguous locations as shown here for M = 3:

$$\left[X_0(R), X_0(I), Y_0(R), Y_0(I), Z_0(R), Z_0(I), X_1(R), X_1(I), \ldots, Z_{N-1}(I)\right]$$

where $X_j(R)$ is the real component of the j+1 element to be transformed and $X_j(I)$ is the corresponding imaginary component.

Table 1. Comparison between Q4FFTV and Q4FFORMS

		CPU times (microsecs) per transform for M transforms of size N	
M	N	Q4FFORMS	Q4FFTV
1	64	3170	1080
32	64	147	430
256	64	59	430

VECTOR CONTIGUOUS LOCATION REQUIREMENT

The vector arithmetic operations on the STAR-100 require the source operands to be in contiguous locations. Consequently, while Q4FFORMS can do the FFT's of the rows of a 2-D grid stored columnwise in the computer, it cannot transform the columns unless the matrix of data values is rearranged so that it is stored consecutively by rows. Consequently, a matrix transpose subroutine is used to do the data movement. As will be observed later, this is not an insignificant cost, being about 1/3 of the FFT time for the cases run. The transposes could be eliminated by using Q4FFTV to transform the column data since that software requires the data for each transform to be stored consecutively. This alternative was rejected because Q4FFORMS is enough faster than Q4FFTV for the problem sizes of interest to more than make up for the extra transpose time.

VIRTUAL MEMORY CONSIDERATIONS

The STAR-100 has a virtual memory architecture. At Langley Research Center, there are 8 large pages of central memory, each page being 65536 64-bit words long. In principle, the virtual memory system can be used to do calculation on problems whose storage requirements exceed the central memory capability. However, there are examples of algorithms, or entire codes, which must be reworked considerably if they are to avoid excessive data movement (paging) to and from central memory.

The subroutine Q4FFORMS has been found to have excessive paging if M*N > 45,000 ≡ MAX. This relatively small value of MAX occurs because:

(1) Temporaries of length M*N are required for vector operations of that length.

(2) The programmer gave the most emphasis in his design to reducing the CPU time; for example, the implementation avoids the bit reversal at the expence of an extra storage array the size of the input array.

In order to permit the code to calculate 2-D FFT's of larger size, a partitioning strategy is incorporated. If (K-1)*MAX ≤ M*N < K*MAX, K separate steps are performed. At each of the K steps, the largest number of rows that Q4FFORMS can accommodate with no paging are moved to a temporary array. The temporary array is then transformed. Similarly, the smaller number of transformed rows are manipulated (e.g., the transposes taken, the two complex transforms recovered from the single complex set, etc.) prior to being inserted into the large output array. With this approach the only paging which occurs is during the movement from the input array to the buffer and possibly from the buffer to the output array. Even this amount of paging, however, is significant so that it is desirable to keep the number of steps K to a minimum.

MAXIMUM VECTOR LENGTHS

The maximum vector length allowable in the STAR-100 hardware is 65,535. From the standpoint of CPU efficiency this is an unimportant restriction and rarely occurs. But, on occasion, it gives rise to added programming complexity. In this case the code was originally written without thought to the vector length restriction. There are portions of the code in which the operations on MP independent transforms of size N involve vectors of length 2*MP*N, meaning that instead of MP*N < 45,000 (the restriction for paging), the code, as originally written, had the vector length restriction MP*N < 32,767. This had the effect of increasing the partitions needed and, hence, the paging. The code has been reworked to divide each major step into the minimum number of substeps necessary to keep the vector lengths below 65,536.

CODE DESCRIPTION

This section describes FFT2DR assuming an input array A contains N columns of real data, M entries per column. Figures 1 and 2 will be referenced frequently. All arrays are stored consecutively by columns. If a block in either figure has an extra vertical line on the left, then the dimensions given refer to complex data entries. These figures do not show the additional logic required to ensure the vector lengths do not exceed 65535.

FORWARD TRANSFORM

There exists the choice to evaluate the 1-D FFT of either the row or column data first. Since Q4FFORMS is ideally structured to transform 2*MP rows of A as MP complex FFT's of size N with no data rearrangement, the rows of A will be transformed first.

F1. The code determines the number of blocks, KK, that A must be partitioned into. Each block contains MR = 2*MP rows of data such that MP*N < MAX. Then for each block of rows, Steps F2 thru F7 are performed.

F2. In subroutine TRBLK, the N columns of MR values each are transferred to the buffer array C. This transfer can be accomplished via a DO Loop performing N vector to vector transfers of length MR. However, the TRANSMIT INDEXED LIST instruction in the STAR has a C-bit option which allows groups of elements from A to be moved to C. Here each group is taken to be one of the N columns. The time for this instruction is given as 73 + N(56 + MR/2). This is superior to the DO Loop approach which would require N(91 + MR/2) plus the overhead for the loop and descriptor generation. All paging occurs during the steps involving TRBLK. Note that since A is stored by columns, each page on which A resides is referenced.

F3. The MP complex transforms of size N are performed by Q4FFORMS. The value of MP has been chosen so that no paging is required during this step. In order to rearrange the complex data so that the second sequence of 1-D transforms can be performed, the row-stored complex data must be transposed.

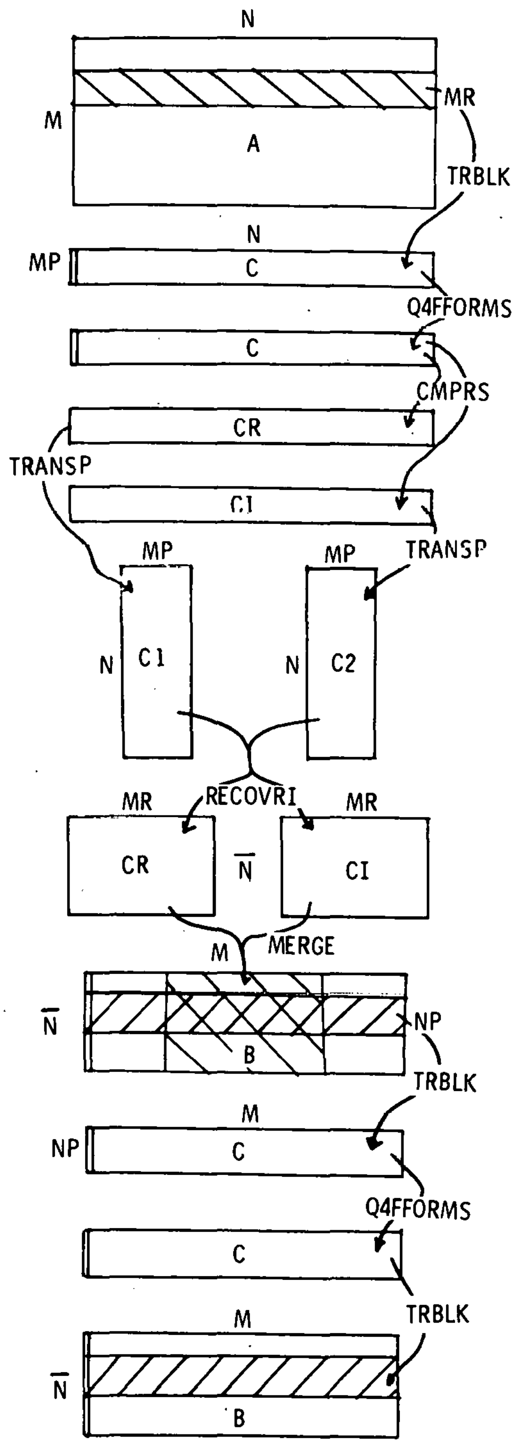


Fig. 1. Forward Transform

F4. To facilitate both the transpose procedure and the recovery of the transforms of the two real data sets, the real and imaginary components are compressed into arrays CR and CI, respectively, in subroutine CMPRS. The real or imaginary part can be obtained using the STAR COMPRESS instruction with a bit vector which alternates 1 and 0. Each will be of length MR*N.

F5. The arrays CR and CI are transposed using subroutine TRANSP which makes repetitive use of the STAR instruction which transposes an 8x8 array. The transposed arrays are stored in the upper and lower halves of C, referred to as C1 and C2 in Fig. 1.

F6. The MR complex transforms are recovered from the MP computed transforms in RECOVRI using Eqs. (4) and (5). The vectorization involves vector additions of length N/2 and vector multiplications of length N. The vector REVERSE instruction is also used to reverse the components of $\bar{U}$ as required in Eq. (4) and Eq. (5). The first (N/2) + 1 (denoted $\bar{N}$ in Fig. 1) components of each of the MR transforms are stored back into CR and CI.

F7. The real and imaginary parts of the transform are merged together in subroutine MERGE using the STAR MERGE instruction with the alternating bit vector. These are vectors of length $\bar{N}$*MR. They are merged into the $\bar{N}$ × M complex output array B.

This completes all the operations on a particular block of MR rows. After all blocks are processed, the 1-D FFT's in the other direction must be computed.

F8. Since the transformed data from the rows of A has been transposed and now resides in columns of B, it is the rows of B which must be transformed. Because of the conjugate even property, there are only $\bar{N}$ such transforms of size M to be performed. Again, FFT2DR determines the number of transforms, denoted NP, to be included in a block and transfers the NP*M complex elements to C in TRBLK. The vector lengths are 2*NP.

F9. The NP complex transforms of size M are computed by Q4FFORMS and stored back into C.

F10. The columns of C are stored back into the original portions of the columns of B using the TRANSMIT LIST INDEXED instruction with the G-bit option in a manner similar to the step F2.

Having completed all blocks of B, the forward transform is complete.

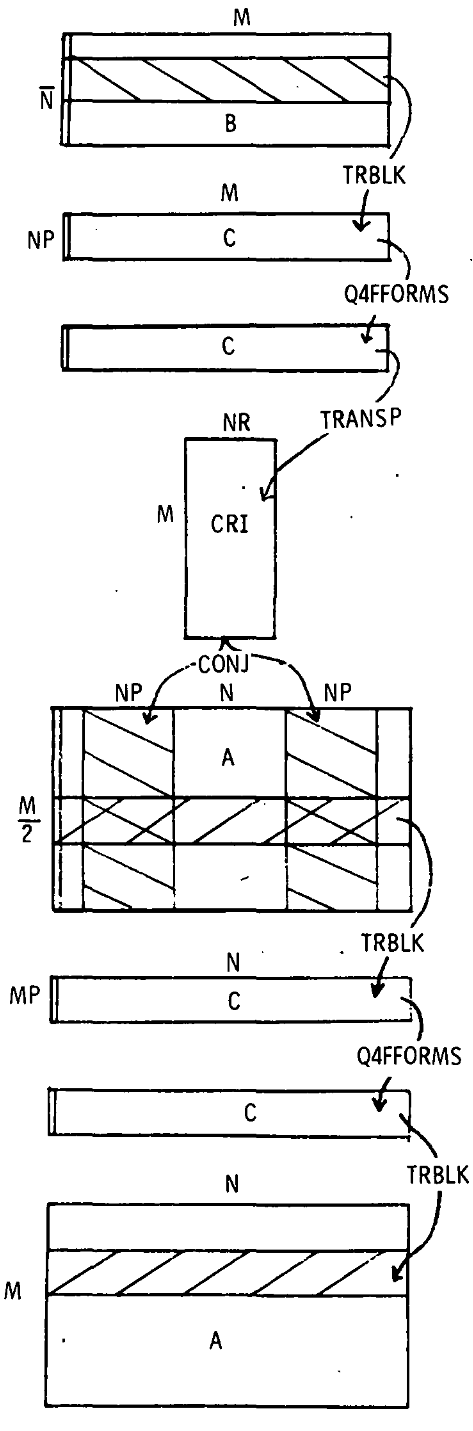


Fig. 2. Inverse Transform

INVERSE TRANSFORM

The inverse transform procedure incorporates much of the software already described. It expects an $\bar{N}$ by M complex array B as input. The program flow is illustrated in Fig. 2.

I1. FFT2DR computes the number of blocks, KK, and the number of complex transforms per block, NP.

I2. TRBLK is called to move the NP complex data sets from B to C.

I3. The NP inverse FFT's of size M are computed in Q4FFORMS.

I4. The array C is transposed using TRANSP into the M × NR array CRI which is equivalent to the space CR and CI occupy. Here NR = 2 × NP.

I5. The array CRI now contains the data in a form that allows the computation indicated in Procedure 2 by Eqs. (6) and (7). Initially, assume NP = $\bar{N}$. Then, for j = 1, 2, .., NP the (2j-1) column of C contains the real part of the j^{th} component of each of the $\bar{N}$ inverse transforms just computed. The 2j column contains the corresponding imaginary parts. The subroutine CONJ uses each pair of columns of C as vectors to compute, first, the j^{th} component of each of the M/2 complex sets (one such complex set is denoted U in Eq. (6)) and then the $(N-j)^{th}$ component (see Eq. (7)). If only a subset of the N transforms are being performed, then the components being computed are only a portion of the total of N and are inserted into B as shown in Fig. 2. The vectorization in CONJ uses vectors of length M/2 or M to compute each of the M/2 complex quantities. An alternating patterned bit control vector is required since the complex result vector has real and imaginary parts interspersed. After all KK blocks of B have been processed, B contains M/2 complex data sets of size N which have been combined as in Procedure 2.

I6. The final steps require only performing the M/2 1-D FFT's after moving a block of MP complex data sets from B into C (see Fig. 2). The resulting MP transforms are then moved to the corresponding part of A.

I7. The resulting real data is recovered using Eqs. (8) and (9). However, the storage used means that the real data set X is a row of A and y is the next row of

A. If the inverse transform on B is performed with no filtering or manipulation of B after the forward transform, the resulting matrix A is the original input data.

RESULTS

A number of cases were executed on the STAR-100 and are summarized in Table 2. The FFT time is, as expected, the most dominant as it requires 53% of the time for a 64x64 grid and the percentage increases for larger grids. The transpose time is also significant as it requires approximately 20% of the total time. The extra work required to recover the two transforms in RECOVRI and to combine the two complex data sets to be inverse transformed in CONJ appears to be well worth the effort. As M and N increase this extra work becomes less important. There are two reasons for this: first, for M = N, the computation in these two subroutines is $O(N^2)$ whereas the overall requirements are $O(N^2 \mathrm{Log}_2 N)$ and secondly, since the vector lengths are O(N), increases in the problem size by a factor of 2 do significantly increase the vector result rate. Q4FFORMS, in contrast, increases its share of CPU time as problem size increases and for essentially the same two considerations: first, its part of the total computation dominates that total more and more, and second, its vector lengths are sufficiently large for the problems shown that increases in vector length are not noticeably increasing the result rate. A close estimate for the FFT times in Table 2, based on published timing[5], is given by $.08*(N+M) + .00014 * N * M * \mathrm{Log}_2(N * M)$. Twice this estimate is then an overestimate to do the entire computation.

As previously mentioned, paging occurs in TRBLK when the data base is so large that it cannot reside fully in memory. As each block of partial columns are transferred between A and C, the entire A array is referenced. It is for this reason that it is important to keep the number of partitions of A to a minimum. For example, setting MAX = 40000 caused KK = 4 for the 512 x 512 grid and resulted in 183 page faults instead of 139.

ALTERNATIVES

It is clear that for the larger problem sizes the paging time (approximately .3 second/page fault) dominates the overall time. There are alternative approaches that will reduce the paging at the expense of increasing the CPU time:

a. Write a more memory conscious FFT subroutine.

Q4FFORMS could be rewritten to perform the bit reversal at completion of the FFT processing. This would reduce the intermediate storage and increase the size of

Table 2. FFT2DR statistics[a] for M × N array of real values

M	64	128	128	256	256	512	512
N	64	64	128	128	256	256	512
Total time (millisecs)	31.6	51.6	90.4	164.0	307.7	670.0	1393.0
TRBLK (%)	7.1	7.1	6.3	6.1	5.7	6.0	5.9
Q4FFORMS (%)	53.0	56.9	55.9	58.5	58.7	61.7	63.7
COMPRESS + MERGE (%)	3.5	3.8	4.2	4.6	4.6	4.3	4.1
TRANSP (%)	19.8	17.5	20.9	19.5	21.2	20.7	20.1
RECOVRI + HERMCON (%)	15.5	13.9	12.6	10.9	9.7	7.3	6.1
No. of partitions	1	1	1	1	1	2	3
No. of page	0	0	0	5	16	50	139

a - The table entries are the total for a forward transform and an inverse transform

MAX. It is also possible to write FFT software which is specifically designed for out-of-memory problem sizes[7] so that no partitioning of A or B is performed at all.

b. Eliminate the B array.

The array B could be eliminated by storing the transformed data back into a slightly enlarged A. This would have the effect of reducing the paging at each step by making the working set smaller. In addition, study shows that the number of calls to TRBLK would be reduced by 1/3. However, since all data is stored columnwise (when considered as a matrix of input data), twice as many transpose operations would be required, significantly increasing the CPU time.

Very little increase in CPU time has resulted from the efforts to generalize the code to the extent shown in Figs. 1 and 2. The only penalty suffered by the partitioning is that when none is required, there is no need to transfer the data from A or B to C as in TRBLK. From Table 2, approximately 7% could be saved. Logic to recognize this situation is currently being implemented.

An alternative which was considered during the code design, but rejected, was computing the FFT of a real data set[3] of size N as the FFT of a complex set of size N/2. It was rejected because the savings in CPU time to perform M FFT's of size N/2 as opposed to M/2 of size N is only minimal (approximately 33% for the M = N = 64 case and less for larger values of M and N). In addition, storage incompatibilities with Q4FFORMS, and more complex pre- and post-processing would offset the modest gains in FFT time.

SUMMARY AND CONCLUSIONS

The STAR-100 code, FFT2DR, which performs a two dimensional FFT of real data, has been described and analyzed. It is shown that for an M × N grid of real data, the CPU time in milliseconds for a forward transform followed by the inverse transform is bounded by $2*(.08(M+N) + .00014 * M * N * LOG_2(M * N))$. The code has been designed primarily to minimize the CPU time but a partitioning strategy has been included to prevent the virtual memory system from "thrashing" during the FFT calculation for larger problems (M*N > 90,000). This strategy adds only approximately 10% of overhead CPU time and while it does permit problems of any size to be done without thrashing, there are still unacceptable amounts of paging for some problem sizes of interest. Alternatives are discussed which will both increase the maximum size of a problem which can be done with no paging at all and will reduce the total amount of paging once it occurs.

While it is both desirable and aesthetically pleasing to write a code for a virtual memory system which will handle the large problems efficiently but at no cost to the smaller ones, it is frequently difficult to do. Persons using a code such as FFT2DR to do 64x64 transform thousands of time do not want the overhead of the partitioning regardless of how relatively small its cost. Persons wanting to do a 1024x1024 transform consider an extra second or two of CPU time insignificant if some other approach will substantially decrease the paging. It appears that even with a virtual memory system it is still necessary, for some types of problems, to have different codes to handle different problem sizes.

ACKNOWLEDGEMENT

The author would like to acknowledge the many useful discussions with Dr. George Ioup of the University of New Orleans about the 2-D FFT and its computation.

REFERENCES

[1]Andrews, H. C., Computer Techniques in Image Processing, Academic Press, 1970.

[2]Fornberg, B., J. Comp. Phy. 25, 1 (1977).

[3]Cooley, J. W., Lewis, P. A., and Welch, P. D., J. Sound Vib. 12, 315 (1970).

[4]Control Data Corporation, STAR-100 Instruction Execution Timing Manual, Pub. 60440600, Arden Hills, Minn. 55112.

[5]Korn, D. G. and Lambiotte, J. Jr., Math. of Comp. 33, 977 (1979).

[6]Lambiotte, J. Jr., NASA TM X-3512 (1977).

[7]Singleton, R. A., CACM 10, 647 (1967).

GENERAL SCIENTIFIC COMPUTATION

THIS PAGE

WAS INTENTIONALLY

LEFT BLANK

Dup

IMPACT OF ADVANCED SYSTEMS ON LMFBR ACCIDENT ANALYSIS CODE DEVELOPMENT

F. E. Dunn and J. M. Kyser
Reactor Analysis and Safety Division
ARGONNE NATIONAL LABORATORY
9700 South Cass Avenue
Argonne, IL 60439 U.S.A.

ABSTRACT

In order to investigate the ability of an advanced computer, using currently available software, to handle large LMFBR accident analysis codes, the SAS3D code has been run on the NCAR CRAY-1. SAS3D is a large code (56,000 Fortran cards) using many different physical models and numerical algorithms, no one of which dominates the computing time. Even though SAS3D was developed on IBM computers, remarkably little effort was required to run it on the CRAY-1. Making limited use of the CRAY-1 vector capabilities, it runs a factor of 2.5 to 4 times faster on the NCAR CRAY-1 than on the ANL IBM 370-195. With minor modifications, an additional 20-30% speed improvement on the CRAY-1 is achieved. In the current process of completely re-writing SAS3D to make SAS4A, much of the coding is being vectorized for the CRAY-1 without sacrificing IBM, CDC 7600, or UNIVAC performance and portability. An initial SAS4A test case runs a factor of 7.1 faster on the CRAY-1 than on the IBM 370-195. On either computer, this SAS4A case runs appreciably faster than a corresponding SAS3D case, indicating that there can be significant benefits from using vectorizable coding, even on a non-vector computer. It appears that even though the one-dimensional models in SAS3D strain the capacity of ANL's current computers, an advanced computer such as a CRAY-1 would make it feasible to replace many 1-D models with 2-D or 3-D models.

INTRODUCTION

The SAS series of computer codes[1,2,3,4] are used to analyze hypothetical accidents in Liquid Metal Cooled Fast Breeder Reactors (LMFBRs), as well as Gas Cooled Fast Reactors (GCFRs). All of the existing codes in the SAS series, except for the original SAS1A, have been capable of representing a reactor core with a quasi-three-dimensional treatment which uses coupled one-dimensional models to approximate the real three-dimensional system. The current representation is adequate for many cases; but for some phenomena, local two-dimensional or three-dimensional effects occur which can not be handled adequately with the current SAS one dimensional models. More detailed two- or three-dimensional models would probably require significantly more computer time than the current models. Even with the current models, a detailed whole core analysis with the SAS3D code can strain the capacity of the current IBM 370-195 and IBM 3033 computers at Argonne National Laboratory (ANL); indicating that it may be desirable or necessary to consider the use of more advanced computers if more detailed models are needed. Therefore, an effort was undertaken to address two main questions. First, how much effort would be required to get the SAS3D code, which is the current production version in the SAS series, to run on an advanced computer? Would extensive re-writing of the code be necessary? Second, since the next SAS code, SAS4A, is currently being written from scratch, as opposed to adding new modules onto SAS3D, can SAS algorithms and coding be modified so as to vectorize on a vector machine without sacrificing IBM, CDC 7600, or UNIVAC performance or portability? Under a grant from the National Center for Atmospheric Research, computer time on the NCAR CRAY-1 computer was available for this project.

SAS CODES

GENERAL DESCRIPTION

The SAS codes have been developed

to analyze the initiating phases of hypothetical accidents in LMFBRs or GCFRs. SAS3D starts with steady-state calculations to determine the initial conditions in the reactor, usually normal operating conditions. Then transient accident calculations are made for a user-specified event, such as loss of power to the primary coolant pumps, or insertion of reactivity at a user specified rate.

In an LMFBR or a GCFR, the reactor core contains long, narrow fuel pins, which are steel tubes (cladding) containing fuel pellets plus a gas plenum above or below the fuel to hold the gaseous fission products released during irradiation of the fuel. The fuel pins are arranged in hexagonal arrays within fuel subassemblies, with coolant flowing in the axial direction between the pins. The subassemblies have steel, hexagonal shaped outer duct walls. The ducts are somewhat longer than the fuel pins to allow room above and below the pins for flow orifices and instrumentation. Typically, there are 217 fuel pins, each 1/4 inch in diameter and about 8 feet long, in a fuel subassembly which is about 12 feet long. There are between 75 and a few hundred subassemblies in a reactor core.

The geometry used by SAS3D to represent the reactor core consists of a number of "channels", where each channel represents a fuel pin and its associated coolant. Usually a SAS channel is used to represent a subassembly or a group of similar subassemblies. In this case, the fuel pin represents an "average" pin in the subassembly. Coolant flow in a SAS channel is only in the axial direction, and heat flow in a pin is calculated only in the radial direction. Because of the very large length-to-diameter ratio of the fuel pins, axial heat conduction within the fuel pin or the coolant is negligible. The coolant removes heat by convection.

Each SAS channel is divided into a number of axial nodes. Typically about 20 axial nodes are used to represent the fueled part of the pin, and another 10-15 nodes are used to represent the rest of the subassembly. At each axial node in the fuel section, about 10 radial nodes are used in the fuel and 3 radial nodes are used in the clad. One radial node is used for the coolant, and one or two radial nodes are used for the "structure", which represents both the subassembly duct wall and the wrapper wires or grid spacers which keep the pins in position. Above and below the fueled section, only a few radial nodes are used.

With this channel treatment, SAS3D calculates steady-state and transient temperatures in the fuel, clad, coolant, and structure. Sodium boiling; clad melting, relocation, and freezing; fuel melting, relocation, and freezing; and interactions between molten fuel and liquid coolant are calculated. Also, the stresses and strains in fuel pins prior to pin failure, the amount of fission product gas in the fuel and its contribution to fuel relocation after pin failure, and the pressures and flow rates of the coolant in the core and around the primary coolant loop are calculated.

ADVANTAGES AND LIMITATIONS OF THE SAS CHANNEL REPRESENTATION

One big advantage of the SAS channel representation is that it provides a detailed, three-dimensional representation of the reactor core even though only one-dimensional equations are solved; and the solution of one-dimensional equations is usually much faster than the solution of multi-dimensional equations. Detailed radial and axial temperature profiles in the fuel pin are obtained by solving a one dimensional radial heat transfer equation at each axial node. Coolant temperatures and flows are obtained by solving one dimensional equations for the axial direction. Fuel and cladding relocation are calculated in the axial direction. Different subassemblies or groups of subassemblies can be represented by different SAS channels, providing detailed axial descriptions at a number of radial and azimuthal locations in the core.

SAS3D only accounts for limited coupling between SAS channels, but this corresponds to the limited interaction between subassemblies in a reactor. The duct walls prevent coolant flow between subassemblies, except that the subassemblies all receive their coolant from a common inlet plenum and all discharge their coolant into a common outlet plenum. The common inlet and outlet plenums are accounted for in SAS3D. The different parts of the core are coupled neutronically, and this neutronic coupling is accounted for in the SAS3D neutronics calculations. There is some heat flow between the duct walls of adjacent subassemblies, whereas SAS3D uses an adiabatic boundary at the outside of the SAS "structure". Accounting for heat flow between subassemblies by computing heat transfer between the structures in different SAS channels could be done in a fairly straight-forward manner without changing the basic SAS channel representation.

The main limitation of the SAS channel representation is that it does not account for any differences between fuel pins or coolant sub-channels within a subassembly. In reality, there are often power skews across a subassembly, the coolant sub-channels next to the duct walls are somewhat larger than those in the interior of the subassembly, and the duct walls have heat capacity and tend to act as limited heat sinks during a transient. The net result is that the interior of a subassembly is usually hotter than the edge, and sometimes one side is hotter than another.

The use of an "average pin" representation in SAS3D tends to average-out many of the variations within a subassembly and often gives good results. For instance, predictions of the boiling model in SAS3D usually agree reasonably well with the results of multiple-pin boiling tests.[5,6] On the other hand, it is often necessary to account for radial incoherence within a pin bundle to obtain satisfactory agreement with clad melting and re-location experiments.[7] A really adequate treatment of fuel relocation probably also requires accounting for radial incoherence within a subassembly.

POSSIBLE IMPROVEMENTS AND COMPUTER LIMITATIONS

Because of the above mentioned limitations of the current SAS3D "single pin" representation of a subassembly, it would be desirable to have a "multiple pin" representation in which a subassembly is represented by a number of pins or pin groups with connected coolant channels.

One problem with developing multiple pin models for whole-core accident analysis is that even the one-dimensional single pin models in SAS3D strain the capacity of the current ANL computers. A 33 channel SAS3D case can take 6 hours of computer time on the IBM 370-195 or IBM 3033. It also requires about 3 megabytes of memory on an IBM computer. Even a 1 channel case requires about 800 kilobytes of memory. On a CDC 7600 the same 33 channel case would require about 3 hours and almost 400,000 words of LCM storage. These computer times are almost entirely CPU times, since SAS3D does relatively little I/O in a big run. If multiple-pin models were used, the running times would probably increase at least linearly with the number of pins used per subassembly, and the increase may be proportional to the square of the number of pins.

Some computer codes already exist for treating some aspects of intra-subassembly incoherence. Because of computer limitations, these codes are usually limited to treating single subassemblies instead of whole cores, and they are limited in the phenomena that they treat. For instance, both the COBRA-3 code[8,9] and the COMMIX-1 code[10] can compute detailed coolant temperature distributions within a subassembly. Either steady-state or pre-voiding transient calculations can be made by these codes. On the IBM 370-195 it is estimated that COMMIX-1 would require about 2.5 hours to compute the steady-state temperatures in all of the coolant sub-channels of a 217 pin subassembly. COBRA-3 can take 20 minutes to calculate steady-state coolant temperatures for 12 coolant sub-channels.

Because of the above mentioned computer limitations, it is unlikely that many pin (217 pin) models will be used in whole-core LMFBR accident analysis codes in the near future, even if significantly faster computers are used. On the other hand, a reasonable increase in computer speed would make it feasible to use "few pin" models for whole-core analysis. A well developed model using 2-5 pin groups to represent a subassembly would probably be quite adequate for most purposes.

NUMERICAL ALGORITHMS AND CODING ASPECTS

SAS3D is a relatively large code, and SAS4A will be larger. The source deck for SAS3D contains about 56,000 FORTRAN cards. The main reason that SAS3D is large is that it contains a number of separate, but coupled, modules for computing different aspects of an accident: heat transfer, coolant flow, fuel pin mechanics, sodium boiling, clad relocation, fuel relocation, fuel-coolant interactions, and neutronics.

The equations solved by these modules are all different, but there are some aspects that are common to most of the modules. Finite differencing in both space and time is used. A number of discrete nodes are used to obtain spatial variations, and the transient time behavior is obtained using discrete time steps. Typically, the algorithms are set up to determine the conditions at each node at the end of a time step, starting from known conditions at the beginning of a time step. In general, the equations solved are non-linear, although they are linearized across a time step. Semi-implicit or fully implicit schemes are often used, leading to the simultaneous solution of linear equations with coefficients that are re-calculated each time step. The resulting matrices are banded, often tri-diagonal. The calculation of the coefficients usually takes longer than the actual solution of the matrix equations. A significant amount of computer time is used in obtaining physical and thermal properties as a function of temperature, and sometimes as a function of pressure or other variables, by linear interpolation from tables or by the evaluation of numerical correlations.

Since the SAS channels are only loosely coupled, SAS3D works on one channel at a time, completing a time step for one or more modules for one channel before going on to the next channel. The arrays used for each channel are stored in a few "data packs". The data packs for a channel are moved into working memory while the code is doing the calculations for that channel, and then moved out to a storage area. Thus, every time step the data packs for each channel are moved into and out of the working area a few times, and each code module is entered at least once for each channel in which the module is active. They are often 1000 or more time steps in a run.

On an IBM or CRAY-1 computer, the storage area is in main memory. On a CDC 7600, working memory is in SCM, and the storage area is in LCM. There are about 9000 words per channel in the data packs. In principal the storage area could be on disk, but in practice it is best to have the whole calculation core-contained. Storing either coding overlays or data packs on disk adds tremendous amounts of I/O time and increases the total running time by about an order of magnitude.

One important aspect of SAS3D is that no one small area of the code accounts for the bulk of the computing time, and no one subroutine accounts for more than about 15% of the total time on the IBM 370-195. The computing time is spread through many modules and many subroutines. Therefore, dramatic improvements in running time can not be obtained by improving a single algorithm or a single subroutine. In order for the code as a whole to run well, a large number of subroutines must each run well.

CODE PORTABILITY

SAS3D is currently being used by many organizations in the U.S and abroad on CDC, IBM, and UNIVAC computers. Therefore, it was written with portability in mind. ANSI-standard FORTRAN[11] is used almost entirely, and machine-dependent features are avoided. The few machine-dependent features that are required are mainly isolated in a few separate subroutines.

The UPDAT Code. One feature that contributes to both the portability and the maintainability of SAS3D is the use of the UPDAT code to modify the SAS3D source files. UPDAT, which was written at ANL by R. George, has many of the features of CDC's UPDATE code,[12] such as inserting and deleting cards, and inserting COMDECKS. In the SAS3D source, the COMMON blocks are listed only once, and the COMDECK feature is used to insert the common blocks in each subroutine where they are needed. UPDAT is also used to make corrections or modifications to the code.

Programs with features similar to those of UPDATE have been available on IBM computers, but the IBM codes and CDC's UPDATE use different directives and require different input. The UPDAT code, which was written in FORTRAN, runs on IBM, CDC, and UNIVAC computers, and all versions use the same input. Therefore, the same UPDAT input deck can be used to modify or correct the IBM, CDC, and UNIVAC versions of the code.

RUNNING SAS3D ON THE CRAY-1

Considering the size of the SAS3D code, it was relatively easy to get the code running on the NCAR CRAY-1. A few routines known to be machine dependent had to be modified, but the modifications were straight-forward. Also, the Cray Fortran Compiler[13] (CFT) would not compile a few statements, but these were mainly cases of violating the ANSI FORTRAN standards.

The actual process of putting SAS3D on the NCAR computer and running it was all done from ANL using a remote batch terminal. The main steps in this process were as follows.

1. A FORTRAN source tape was written at ANL and sent to NCAR. This tape contained 3 files. The first file was the source for the UPDAT program. The second file contained the SAS3D COMMON blocks, and the third file was the SAS3D source file.

2. The UPDAT program was compiled after the necessary modifications to the UPDAT source were made using the system UPDATE utility.

3. UPDAT was used to insert the common blocks into the SAS3D routines and to make modifications to the SAS3D source.

4. The resulting SAS3D source code was compiled, and both the source file and the object file were stored in permanent datasets on the CRAY-1 disks.

5. Sample SAS3D cases were run, and the results were compared with IBM and CDC results.

The CDC version of the export version of UPDAT was sent to NCAR. Some changes to this code were required to get it to run on the CRAY-1. Frist, some machine-dependent constants had to be changed to account for differences in word length and data representation. The CDC 7600 uses ten 6 bit characters per 60 bit word, whereas the CRAY-1 uses eight 8 bit characters per 64 bit word. Fortunately these constants were all set at the same place in DATA statements, so it was easy to change them. Second, in one spot the CRAY-1 version required the FORTRAN function SHIFTR instead of the SHIFT function used in the CDC version. Third, the UPDAT directive *END happens to correspond to a NCAR control card, so in UPDAT this directive was changed to *EEND by changing one DATA statement.

The CDC 7600 version of SAS3D was sent to NCAR. This version is overlayed, with a special overlay routine that stores overlays in LCM rather than on disk. Since the NCAR CRAY-1 has plenty of main memory, and no LCM, the CRAY version of SAS3D was not overlayed. Thus, all of the OVERLAY and CALL OVERLAY cards were removed, as well as subroutine OVERLAY.

Another known machine-dependent aspect was the data pack storage area and the routines that store and retrieve data packs. On the CDC 7600, the routines READEC and WRITEC, written in COMPASS, are used to store blocks of data in LCM when they are not being used, and to put them back in SCM when they are needed. On the ANL IBM machines, the data packs are stored in main memory, using specially optimized FORTRAN versions of READEC and WRITEC. On the CRAY-1, the data packs are stored at the end of blank common, using simple FORTRAN versions of READEC and WRITEC. The CRAY FORTRAN compiler (CFT) automatically vectorizes these versions of READEC and WRITEC. Since the CRAY loader loads coding from the bottom of memory up, with blank common at the end of the coding, and since I/O buffers start at the top of memory and work down, any unused memory is between the end of blank common and the bottom of the I/O buffers. Therefore, the effective length of the data pack storage area at the end of blank common can be set at run time by specifying the total job memory size to correspond to the size of the problem being run.

Another machine-dependent aspect of SAS3D is the timing routine TLEFT. For the CRAY-1, a TLEFT routine that sets TLEFT to 1,000,000-100.*SECOND(1.0) was used, where SECOND is the elapsed CPU time. This version gives the correct timing of various parts of the code, but it does not give an accurate warning when the code is approaching a time limit.

Some accumulated SAS3D modifications were also incorporated into the CRAY-1 version. These modifications are minor items that correct some known non-standard usages in the code.

The first attempt to compile SAS3D on the CRAY-1 turned up only 6-8 FORTRAN errors. One error was in a DATA statement in subroutine RESTAR. The IBM version of this statement uses a 4H specification, the CDC version uses 10H, and the CRAY version requires 8H. The other errors were all cases of non-standard separators in FORMAT statements. The CRAY compiler does not allow two consecutive comma separators or ,/, in a FORMAT statement. Apparently the IBM and CDC systems ignore spurious commas in FORMAT statements.

After SAS3D compiled, the first attempt to run a case turned up one last problem. Subroutine WRITEI is a multiple-entry routine, even though it should have been written originally as two separate single-entry routines. The CRAY compiler uses an IBM-type convention for passing arguments to multiple entry points, whereas the CRAY version of SAS3D started as a CDC version, with a different treatment of multiple entry points.

After the entry point problem was corrected, a number of SAS3D cases have been run successfully without encountering any additional problems.

<u>Linear Interpolation Routines</u>

Although the Cray Fortran Compiler will automatically vectorize part of the SAS3D coding, the FORTRAN coding in three linear interpolation routines, INTIRP, INTERP, and INTRP, will not vectorize. These routines provide an area in which moderate improvements in CRAY performance can be achieved with relatively little effort, since they are small routines that account for a moderate fraction of the total running time.

INTIRP scans a table of Y as a function of X. It obtains the result Y_1, corresponding to the input value X_1, by linear interpolation between the appropriate table values. INTERP is the same as INTIRP, except that INTERP is passed an extra parameter, IFUEL, the fuel type; and the tables used by INTERP contain entries for each fuel type, i.e., the Y array has two subscripts, Y(J, IFUEL). INTRP takes a whole array of input variables, $X_1(I)$, and an array of fuel types, IFUELI(I), and it computes a whole array of output results, $Y_1(I)$.

Timing studies on both the IBM 370-195 and the CDC 7600 have shown that INTIRP and INTERP spend more time scanning the tables to find the appropriate table entries than they do in the actual interpolation. INTIRP and INTERP always start scanning from the start of the table. INTRP starts scanning at the start of the table for the first variable in the input array. For later values in the imput array, INTRP starts scanning the table at the location where it found the previous value.

The table scanning loop in these routines was written in a somewhat convoluted manner in order to get into loop-mode on the IBM 370-195, since significant speed improvements are often achieved in loop-mode. A simple DO loop containing an IF statement that jumps out of the loop when the appropriate table location has been found will not run in loop-mode on the 195. The logic required to achieve loop-mode on the IBM machine degrades the performance on CDC and CRAY computers, both of which will run the simple DO loop version as a simple in-stack loop.

The Cray Fortran Compiler uses only scalar instructions to compile either the convoluted scanning loop or the simple scanning loop in the interpolation routines. Therefore, CAL verisons of these routines were written to use the vector compare instructions on the CRAY-1. Also, the CAL version of INTRP performs the actual interpolation calculations in vector mode.

<u>Timing Results for Interpolation Routines.</u> For timing purposes a simple driver program was written to call the interpolation routines with the proper arguments. The tables used for this program had a length of 20, which is typical of the tables used in SAS3D. An array of 12 values of $X_1(I)$ was used, and these values were distributed fairly evenly over the tables. The values used for IFUEL were 1,1,1,1,2,2,2,2,3,3,3,3. For timing INTIRP an inner DO loop in the driver made 12 separate calls to the routine using the appropriate values for X_1 and IFUEL. For timing INTRP, one call was made to obtain an array of 12 results. In order to obtain running times large enough to measure, an outer DO loop was used to call INTRP 1000 times or to execute the inner loop for INTIRP 1000 times. Thus, the measured times are for 1000 calls to INTRP or 12,000 calls to INTIRP. Since INTERP is very similar to INTIRP, it was not timed.

Table 1 lists the running times measured for these routines. The simple DO loop for scanning does somewhat better than the original coding on any computer. One call to INTRP with an array of 12 values takes less time than the corresponding 12 calls to INTIRP, partly because subroutine linkage overhead accounts for a moderate fraction of the total INTIRP time, and partly because of the more efficient table scanning in INTRP. The FORTRAN versions of these routines run a factor of 2-2.5 times as fast on the CRAY-1 as on the IBM computers. The use of the vector compare instructions, as well as generally tighter coding, in the CAL version improves the CRAY speed by an additional factor of 3-4.

Table 1. Timing results for interpolation routines.

Computer, Compiler	CPU time (seconds) for 12,000 results		
	INTRP	INTIRP	
		original coding	simple scanning DO loop
IBM 370-195 FTH,[a] OPT=2	.165	.366	.320
IBM 3033 FTH, OPT=2	.160	.305	----
CDC 7600 FTN4,[b] OPT=2	.121	.224	.165
CRAY-1 CFT	.067	.178	.127
CRAY-1 CAL	.021	----	.032

[a]IBM's Fortran H compiler
[b]CDC's Fortran Extended, Version 4 compiler, as implemented on the Lawrence Berkeley Laboratory Computers.

SAS3D TIMING RESULTS

Timing comparisons for the SAS3D code are complicated by two factors. First, there is no one small section of the code that accounts for the great bulk of the computing time. It is necessary to run the whole code, or a significant fraction of the code, in order to get meaningful timing comparisons. Second, there are many types of cases in which it is not possible to get exactly the same results on different computers. In many cases, differences in running times between computers are due to both differences in computer speeds and differences in computational paths.

Three different SAS3D cases were run on the CRAY-1. The first case was a limited case which exercised only part of the code, but it was a case for which the same computed results are achieved on all computers, so that exact timing comparisons are meaningful. This was the first 300 time steps of a 1-channel low power boiling case (LOWBLA). This run was terminated before boiling initiation, so it only tested the pre-boiling parts of the code.

The second case was a more extensive 1-channel case: 1000 time steps for channel 1 of a 33-channel CRBR transient undercooling case (1-channel test). This case gets into sodium boiling, clad relocation, and fuel relocation (SLUMPY). The results obtained on different computers for this case were not identical, but they were quite similar; and the computational paths were quite similar.

The third case was the standard SAS3D 3-channel test case (3-channel test). This case was run mainly to test the code rather than to get timing comparisons. This case tests most of the options in the code. It is an extremely touchy case with an appreciable amount of positive feed-back. Any small deviation, due to factors such as round-off error, tends to grow as the run progresses, and it is not possible to get the same results for this case on different computers. Even the IBM 370-195 and the IBM 3033 give different results for this case.

Table 2 gives the running times on various computers for these cases. Also, given in parentheses are the relative speeds, with the IBM 370-195 speed defined as 1 for each case.

Table 2. SAS3D timing comparisons.

Computer	CPU time, seconds (Relative speed, 1/CPU time)		
	LOWBLA, 1 channel, 300 steps, no boiling	1 channel test, 1000 steps	3-channel test[a]
IBM 370/195	44.1 (1.0)	333.3 (1.0)	740.0 (1.0)
IBM 3033	43.8 (1.01)	309.1 (1.08)	----- ----
CDC 7600	19.4 (2.27)	162.9 (2.05)	453.9 (1.63)
CRAY-1 CFT[b]	11.8 (3.73)	129.6 (2.57)	186.2 (3.97)
CRAY-1 CAL[c]	9.7 (4.55)	95.5 (3.49)	----- -----

[a]Timing comparisons are not very meaningful for the 3-channel test case.
[b]All-Fortran version.
[c]CAL versions for three interpolation routines, everything else Fortran.

SAS3D runs slightly faster on the IBM 3033 than on the 195, about twice as fast on the CDC 7600 as on the 195, and a factor of 2.5 to 4 times as fast on the CRAY-1 as on the 195.

The only area in which performance improvement for SAS3D on the CRAY-1 was investigated was the three linear interpolation routines. Use of the CAL versions of these routines led to an improvement of 20% - 35% in the overall running time of SAS3D.

SAS4A

The initial version of SAS4A contains mainly steady-state and pre-voiding transient heat transfer and coolant flow routines. Other modules are being added as they are developed, but this initial version is the only one that has been run on the CRAY-1.

There are two main differences between these SAS4A pre-voiding routines and the corresponding SAS3D routines. First, the pre-voiding SAS4A module contains routines that have been especially tailored for prevoiding transient calculations, whereas in SAS3D the corresponding routines are generalized routines that handle the whole transient. Second, the algorithms used in the pre-voiding routines were modified somewhat to promote vectorization. This did not require major changes in algorithms; mainly it involved re-ordering of calculations and sometimes the saving of arrays of interim results. Also, some changes in programming style were required to eliminate cases where the basic algorithm allowed vectorization but programming style precluded it.

Most of the pre-voiding transient coolant calculations vectorized easily. These calculations consist mainly of obtaining coolant properties at each axial coolant node by evaluating parametric fits. The fits are all single range fits containing no branching, and properties for all coolant nodes can be calculated in parallel. Also, since typically about 30 coolant nodes are used, vector lengths of about 30 were achieved.

In cases where a complete calculation could not be vectorized, part of the calculation often could. For instance, the calculation of the coolant pressure at node J requires the value from node J-1, so the calculation could not be vectorized. In this case, the calculation of the node-to-node pressure differences would vectorize, and this calculation accounts for most of the computing time in this area. Then a small non-vectorized loop sums the differences to give the final results.

Vectorizing the heat transfer routines was more difficult. For each axial node, the temperatures at all radial nodes are solved for simultaneously by solving a tri-diagonal matrix equation. Many of the calculations used to obtain the coefficients of the matrices were vectorized, but the vector lengths were usually no longer than the number of radial nodes, which typically ranges from 4 or 5, above and below the fueled region, to about 17 in the fueled region. The tri-diagonal matrix solution itself does not vectorize in the pre-voiding mdule, although in the voiding module the corresponding matrix solution might be vectorized by solving for all axial nodes simultaneously. In the pre-voiding calculation, the computed coolant temperature at axial node J is needed to obtain some of the coefficients for node J+1; but in the voiding module, the coolant temperatures are calculated separately in the coolant routines, and in the fuel pin heat transfer routines there is no coupling between axial nodes.

SAS4A TIMING RESULTS

A non-voiding case with 1000 time steps was timed using the initial version of SAS4A. In addition to total running times, a timing distribution by subroutine was also obtained. For the timing distribution, the CFT timing trace was used on the CRAY-1, the PROGLOOK feature was used on the ANL IBM 370-195, and a combination of the SNOOPY routines plus a number of calls to SECOND was used on the CDC 7600. For this case, the steady-state initialization accounts for less than 1% of the running time, so it was mainly the pre-voiding transient routines that were timed. Tables 3 and 4 give these timing results.

These timing results indicate that SAS4A runs about six times as fast on the CRAY-1 as on the IBM 370-195; with CAL versions of the interpolation routines, the speed ratio increases to seven. The CDC 7600 version also runs appreciably faster than the IBM version, but the CRAY-1 version still runs a factor of 2.1 to 2.5 times as fast as the CDC 7600 version.

Table 3. SAS4A timing comparisons.

Computer	CPU time seconds	Relative speed 1/CPU time
IBM 370-195	43.6	1.0
IBM 3033	46.5	.94
CDC 7600	15.49	2.8
CRAY-1, CFT[a]	7.37	5.9
CRAY-1, CAL[b]	6.14	7.1

[a]All-Fortran version.
[b]CAL versions of interpolation routines, CFT for rest of code.

Table 4. Detailed breakdown of SAS4A timing.

	CPU time, seconds (percentage of total)			
			CRAY-1	
Program area	IBM 370-195	CDC 7600	CFT	CAL[a]
heat transfer, except matrix solution	9.98 (23%)	4.54 (29%)	2.01 (27%)	2.01 (33%)
matrix solution	1.81 (4%)	1.41 (9%)	.51 (7%)	.51 (8%)
interpolation routines	4.38 (10%)	3.46 (22%)	1.91 (26%)	.68 (11%)
coolant routines	2.03 (5%)	1.19 (8%)	.42 (6%)	.42 (7%)
log, exp, x^y	6.85 (16%)	1.84 (12%)	b	b
READEC, WRITEC, data pack movement	9.22 (21%)	.56 (4%)	.32 (4%)	.32 (5%)
formatted I/O	8.41 (19%)	2.38 (15%)	2.10 (29%)	2.10 (34%)
other	.92 (2%)	.11 (1%)	.10 (1%)	.10 (2%)
total	43.6 (100%)	15.49 (100%)	7.37 (100%)	6.14 (100%)

[a]CAL versions of INTRP, IINTIRP, CFT for rest of code.
[b]Included with coolant routines.

Comparisons with the LOWBLA times in Table 2 for 300 pre-voiding steps show that SAS4A runs 1000 steps in about the same time that SAS3D requires for 300 steps. About 1/3 of the SAS3D time for this case was accounted for by the DEFORM module, which has not been incorporated into SAS4A yet; but the remaining 2/3 of the time is in routines corresponding to the SAS4A routines. This indicates that the heat transfer and coolant routines in SAS4A run about twice as fast as the corresponding SAS3D routines. On the CRAY-1, the SAS4A speed improvement is greater than a factor of two.

Formatted I/O, mainly printing transient results, accounted for an appreciable fraction of the total CPU time for this case. In 6 seconds of computing on the CRAY-1, this case printed 213 pages of output. For longer runs, it will be necessary to reduce greatly the amount of print-out per second of computation.

The log and exponential functions are called by the coolant routines. On the IBM and CDC computers, and maybe on the CRAY-1 also, they account for the bulk of the time spent in the coolant routines. The coolant routines vectorized well on the CRAY-1, and this shows in the relative coolant calculation times. On the CRAY-1, the coolant routines, including the log and exponential functions, run about 21 times as fast as on the IBM 370-195, or about 7.2 times as fast as on the CDC 7600.

The data pack movement is quite a bit slower on the IBM 370-195 than on either the CRAY-1 or the CDC 7600. This reflects the relatively slow speed of the main memory on the IBM 370-195.

SUMMARY AND CONCLUSIONS

The SAS3D code was run on the CRAY-1 computer with only minor modificaitons to the code, and it ran reasonably well. On the CRAY-1, SAS3D runs 3.5 to 4.5 times as fast as on the IBM 370-195, or about twice as fast as on the CDC 7600. The IBM and CDC compilers that are used with SAS3D are highly developed compilers that produce well optimized object code, so the performance of SAS3D on the CRAY-1 is a reflection of the basic speed of the CRAY-1 hardware, as well being an indiciation that the CFT compiler produces moderately efficient object code.

Writing new pre-voiding heat transfer and coolant flow routines for SAS4A, using new algorithms and coding that would vectorize where possible, led to significant speed improvements on all three computers. The CRAY-1 version of SAS4A runs about 2.5 times as fast as the CDC 7600 version, which runs 2.8 times as fast as the IBM 370-195 version. Even the IBM version of SAS4A runs about twice as fast as the corresponding IBM version of SAS3D. Some of the improved speed of SAS4A, as compared to SAS3D, was probably due to the use of somewhat better algorithms and the elimination of some unnecessary calculations, but much of this improvement is probably due to the fact that coding that vectorizes on a CRAY-1 tends to run efficiently on an IBM 370-195 or a CDC 7600. Therefore, new coding for the SAS codes should be vectorizable where possible, even if it is not expected that these codes will be run extensively on vector machines in the near future.

The performance attained by SAS3D and SAS4A on the CRAY-1 was partly due to the ability of the CRAY-1 to use short vectors effectively. Many of the vector lengths in these codes are in the range from 10-20, and some are as small as 3 or 4. Other than block transfers, none of the vector lengths is currently greater than 48. The SAS codes probably would not perform well on a computer designed for long vectors.

The computing speed attainable on a CRAY-1 computer should make it feasible to develop "few pin" models for whole core accident analysis, if the models are carefully developed and coded so as to optimise computer performance. Such calculations may even be feasible on a CDC 7600. Even on a CRAY-1, detailed many pin (217 pins per subassembly) models would probably still be restricted to limited, single subassembly cases because of running time considerations.

ACKNOWLEDGMENTS

We would like to thank L. Rudsinski for obtaining computer time on the NCAR CRAY-1 for this project, and also for his assistance in getting SAS3D to the NCAR computer. This work was performed under the auspices of the U. S. Department of Energy.

REFERENCES

1. D. R. MacFarlane, J. C. Carter, G. J. Fischer, T. J. Heames, N. A. McNeal, W. T. Sha, C. K. Sanathanan, and C. K. Youngdahl, ANL-7607 (1970).

2. F. E. Dunn, G. J. Fischer, T. J. Heames, P. A. Pizzica, N. A. McNeal, W. R. Bohl and S. M. Prastein, ANL-8138 (1974).

3. M. G. Stevenson, W. R. Bohl, F. E. Dunn, T. J. Heames, G. Höppner, and L. L. Smith, Proceedings of the Fast Reactor Safety Meeting, Beverly Hills, CA, (1974), CONF-740401, p. 1303.

4. J. E. Cahalan, D. R. Ferguson, H. U. Wider, C. H. Bowers, L. L. Briggs, F. E. Dunn, J. M. Kyser, L. Mync, A. M. Tentner, and W. L. Wang, ANS/ENS International Meeting on Fast Reactor Safety Technology, Seattle, Washington, (1979).

5. G. Höppner, W. L. Chen, F. E. Dunn, and M. A. Grolmes, Trans. Am. Nucl. Soc., 18, 213 (1974).

6. I. T. Hwang, T. M. Kuzaz, W. W. Marr, and K. J. Miles, Trans. Am. Nucl. Soc., 28, 443 (1978).

7. M. Ishii and W. L. Chen, Trans. Am. Nucl. Soc., 28, 442 (1978).

8. D. S. Rowe, BNWL-1522-4 (1971).

9. W. W. Marr, ANL-8131 (1975).

10. W. T. Sha, ANL-7796, NUREG/CR-0785 (1979).

11. "American National Standard Programming Language FORTRAN," ANSI x3.9-1978.

12. Update Reference Manual, Control Data Corporation Publication No. 60342500, Minneapolis, Minnesota, (1978).

13. CRAY-1 FORTRAN (CFT) Reference Manual, Cray Research Publication No. 2240009, Bloomington, Minnesota, (1977).

IMPLEMENTATION OF A LINEAR SYSTEM SOLVER

James G. Sanderson
Los Alamos Scientific Laboratory
P. O. Box 1663
Los Alamos, New Mexico 87544

ABSTRACT

We will discuss the use of a line relaxation scheme to solve a five point difference approximation to the radiation diffusion equation. The matrix solver has been vectorized for the CRAY-1. We will also discuss the use of a column convergence test and column iteration decision to speed computation.

In addition, the implementation of a multigrid routine in one of two dimensions is mentioned.

INTRODUCTION

We seek a solution to the radiation diffusion equation over a region containing numerous materials and moving material boundaries. One approach [1] leads to the implicit difference scheme

$$K_1T_{ij}^{(n+1)} - C_1T_{ij-1}^{(n+1)} - C_2T_{ij+1}^{(n+1)} - C_3T_{i+1j}^{(n+1)} - C_4T_{i-1j}^{(n+1)} = K_1T_{ij}^{(n)} \qquad (1)$$

where n represents the time step, T the fourth power of the temperature, and C_i and K_i are the coupling coefficients. In matrix form (1) is the usual symmetric five point difference matrix under the natural ordering.

Various methods exist to solve (1). In [2] Buzbee, et. al., describe the use of successive line over relaxation (SLOR). They choose to adopt an odd-even ordering of lines. Hence the tridiagonal systems associated with odd (even) lines are mutually independent and can be solved simultaneously. By solving the systems simultaneously vectorization was achieved. As in the problem here the region was embedded in a rectangle.

We will study the strategy suggested by Buzbee and also study the use of an old convergence test and column iteration decision to speed convergence.

PROBLEM

We first set up the matrix of coupling coefficients. Since several materials may be contained in each cell and each material requires table lookups, the computational cost is extreme. Having constructed the matrix and right hand side we are ready to iterate using SLOR. To be specific consider the following example with i = 60, j = 129.

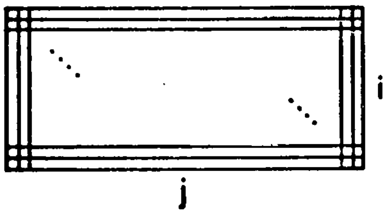


Fig. 1. A 60 x 129 problem.

Using the natural ordering , the resulting matrix can be written as

$$\begin{bmatrix} A_1 & D_2 & & & \\ D_2 & A_2 & & & \\ & & A_3 & & \\ & & & & D_{129} \\ & & & D_{129} & A_{129} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ \\ \\ X_{129} \end{bmatrix} = \begin{bmatrix} B_1 \\ B_2 \\ \\ \\ B_{129} \end{bmatrix} \qquad (2)$$

Where A_i is a symmetric tridiagonal matrix, 60 × 60 and D_i is a diagonal matrix of dimension 60 × 60. Let X_i be the solution vectors of length 60 and let B_i be the vectors representing the right hand side.

ALGORITHM

Let $X_i^{(o)}$ be some initial guess at the solution. Using Buzbee's suggestion write

$$A_j \hat{X}_j^{(n+1)} = D_j X_{j-1}^{(n)} +$$

$$D_{j+1} X_{j+1}^{(n)} + B_j \qquad (3)$$

$$X_j^{(n+1)} = \omega (\hat{X}_j^{(n+1)} - X_j^{(n)}) \qquad (4)$$

$$+ X_j(n)$$

We will solve (3) simultaneously for all the odd values of j and update the guess X_i as in (4) for the odd values of j. We then repeat (3) for the even values of j. Then (4) for the even values of j. The procedure described constitutes one iteration. It will suffice to note that ω is computed every 12 iterations by a prescription described by Carré [3]. See also [4-8] for advice on computing ω.

Several observations can be made. Since the matrixes A_j do not change from one iteration to the next, we can factor each A_j and store the factors. It is necessary then only to backsolve to arrive at $\hat{X}_j$. Initially ω is chosen to be 1.375. The entire iteration is terminated when the sum of the squares of the updates in each column is less than a prescribed tolerance. Finally note that the solution estimate is changed only after each odd (or even) sweep is completed.

Both the initial factorization of A_j and the backsolves are vectorizable.

TIMING

The algorithm described above was implemented in FORTRAN and run on the CRAY-1 using the XFC compiler, OPT = 1 at Los Alamos Scientific Laboratory. The same routine was then vectorized using the MCA vectorizer [9] and executed similarly. The routine consists of setting up the matrix from stored coefficients, the initial factorization of the matrix and the SLOR iteration for the 60 × 129 problem.

TABLE I

TIMING IN SECONDS OF CPU TIME

	FORTRAN VERSION	VECTORIZED VERSION
Initial Factorization	.009721	.001725
Time/Iteration	.0453	.0361
Time/Tridiagonal Solve:		
Even (64)	.06366	.001562
Odd (65)	.06465	.002053
Entire Routine:		
9 Iterations	.4306	.34958
49 Iterations	2.278	1.823

Timing studies under CFT are forthcoming.

OLD CONVERGENCE TEST

In an ancient LASL code which used a variation of SLOR an interesting line convergence test and iteration decision was implemented. Simply stated, a column will not be iterated on if it has converged and both its neighbors have converged. Consider the following example where the three middle columns have converged under the sum of the squares test previously described. See Fig. 2.

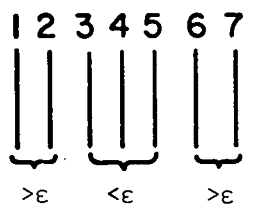


Fig. 2. Columns 3, 4 and 5 have converged.

The routine continues to iterate on columns 1-3 and 5-7. Since a column is converged if and only if its immediate neighbors are converged. As the iteration proceeds more columns will converge and the work load in scalar mode will decrease. Note that a converged column may become "unconverged" on subsequent iterations.

Since the column iteration test involves a FORTRAN IF test within the backsolve loop it is not readily vectorizable and thus there is no need to choose the odd-even ordering. The matrix factorization is still vectorization. Time per iteration will vary since a different number of columns will be iterated on in general. However, total run times were:

9 iterations	.41585
52 iterations	1.086

for the respective cycles in Table I.

It should be noted that if most of the columns are converged then each iteration is quite inexpensive. In problems with local disturbances or front propagation problems either column or line convergence tests could save time.

MULTIGRID APPROACH

The multigrid method [10] users Gauss-Seidel iterations on levels of fine or coarse meshes to accelerate convergence. In the problem discussed here the cost of recomputing the coefficients would prove prohibitive. However, it is quite possible to describe a two level multigrid technique in one of two dimensions. It is best described by Fig. 3.

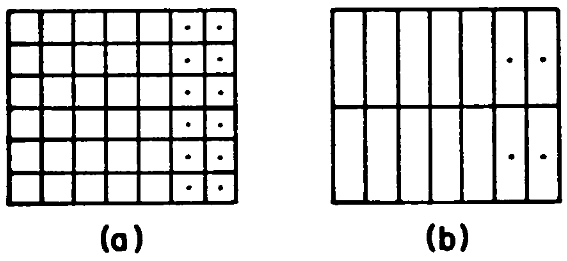


Fig. 3. (a) is the fine mesh, (b) is the coarse mesh. Note that the same number of columns are used.

For the mesh in Fig. 3 (b), the coupling coefficients to neighboring cells are simply the average or sum of the appropriate coupling coefficients originally computed for the fine mesh shown in Fig. 3 (a). The number of rows is reduced in the coarse iteration and the number of columns remains unchanged. The method combines the use of multigrid and the vector solver while minimizing coefficient computation cost.

ACKNOWLEDGMENTS

The author wishes to thank Billy Buzbee of LASL for his support and encouragement. Discussions with Seymour Porter and Richard Varga were also helpful.

REFERENCES

[1] J. G. Sanderson, "An Implicit Scheme for the Solution of the Nonlinear Radiation Diffusion Equation," Los Alamos Scientific Laboratory, 1979.

[2] B. C. Buzbee, L. D. Boley, and S. V. Parter, "Application of Block Relaxation," Society of Petroleum Engineers Fifth Symposium on Numerical Simulation of Reservoir Performance.

[3] B. A. Carré, "The Determination of the Optimum Accelerating Factor for Successive Over-Relaxation," The Computer Journal, Vol. 4, 1961.

[4] G. E. Forsythe and J. Ortega, "Attempts to Determine the Optimum Factor for Successive Over-Relaxation," Info, Proc., UNESCO, Paris, 1959.

[5] H. E. Kulsrud, "A Practical Technique for the Determination of the Optimum Relaxation Factor of the Successive Over-Relaxation Method," Comm. Assoc. Comput. Mach., Vol. 4, 1961.

[6] J. K. Reid, "A Method for Finding the Optimum Successive Over-Relaxation Parameter," The Computer Journal, Vol. 9, 1966.

[7] A. K. Rigler, "Estimation of the Successive Over-Relaxation Factor," Math. Comp., Vol. 19, 1965.

[8] L. A. Hageman and R. B. Kellogg, "Estimating Optimum Over-Relaxation Parameters," Math. Comp. Vol. 22, 1968.

[9] Massachusetts Computer Associates.

[10] R. A. Nicolaides, "On Multiple Grid and Related Techniques for Solving Discrete Elliptic Systems," J. Comp. Phys. 19, 1975.

ADVANCED COMPUTERS AND MONTE CARLO

Thomas L. Jordan
Los Alamos Scientific Laboratory
P.O. Box 1663
MS 265
Los Alamos, New Mexico 87544

ABSTRACT

High-performance parallelism that is currently available is synchronous in nature. It is manifested in such architectures as Burroughs ILLIAC-IV, CDC STAR-100, TI ASC, CRI CRAY-1, ICL DAP, and many special-purpose array processors designed for signal processing. To our knowledge, this form of parallelism has not been of significant value to many important Monte Carlo calculations. Nevertheless, there is much asynchronous parallelism in many of these calculations. A model of a production code that requires up to 20 hours per problem on a CDC 7600 is studied for suitability on some asynchronous architectures that are on the drawing board. The code is described and some of its properties and resource requirements are identified to compare with corresponding properties and resources of some asynchronous multiprocessor architectures. Arguments are made for programmer aids and special syntax to identify and support important asynchronous parallelism.

INTRODUCTION

Monte Carlo calculations predate their computation with electronic computers. An interesting aid for doing Monte Carlo calculations was invented by Enrico Fermi. In 1946 S. Ulam proposed a statistical approach to study neutron behavior in various materials and geometries. Shortly thereafter, John von Neumann developed an algorithm with anticipated use of the ENIAC. Delays in availability of the ENIAC, due to its move to Aberdeen, caused Fermi to think about a substitute. He designed the 13-inch long, hand-operated mechanical computer shown in Fig. 1 to perform these computations. It was built by L. D. P. King. Neutron sample sizes of 100 were used to develop neutron histories and statistics. The operator needed tables of random numbers and mass constants to operate this "computer."

Electronic computers eventually provided the speed and decision-making capability necessary for any realistic modeling of serious simulation of probalistic events. The study of particle physics (neutrons, photons, etc.) has provided a classical setting for the Monte Carlo method. Flexible Monte Carlo codes exist to study detailed particle physics in very elaborate geometries. Because of the computer time required, these codes are used most frequently whenever more accuracy and/or geometric reality is required than can be provided by one- and two-dimensional transport codes. Problems requiring many hours of CDC-7600 time are not uncommon. Monte Carlo would be used more frequently if the method were not so costly.

How has this method faired relative to other types of calculation, say solving partial differential equations, during recent periods of increasing computing power? Once we began getting speed through synchronous parallelism (on computers such as the CDC STAR-100, CRI CRAY-1, TI ASC, ICL DAP, and ILLIAC IV), Monte Carlo calculations have not kept pace. Efforts to vectorize our Monte Carlo codes have not been very successful to date. Although some success has occurred with the simplest of models, a success verdict is not yet in on the big codes. As a consequence, the ability to perform this kind of computation is not keeping pace with a growing need for yet more detailed simulation.

To better understand what is required for Monte Carlo codes, we de-

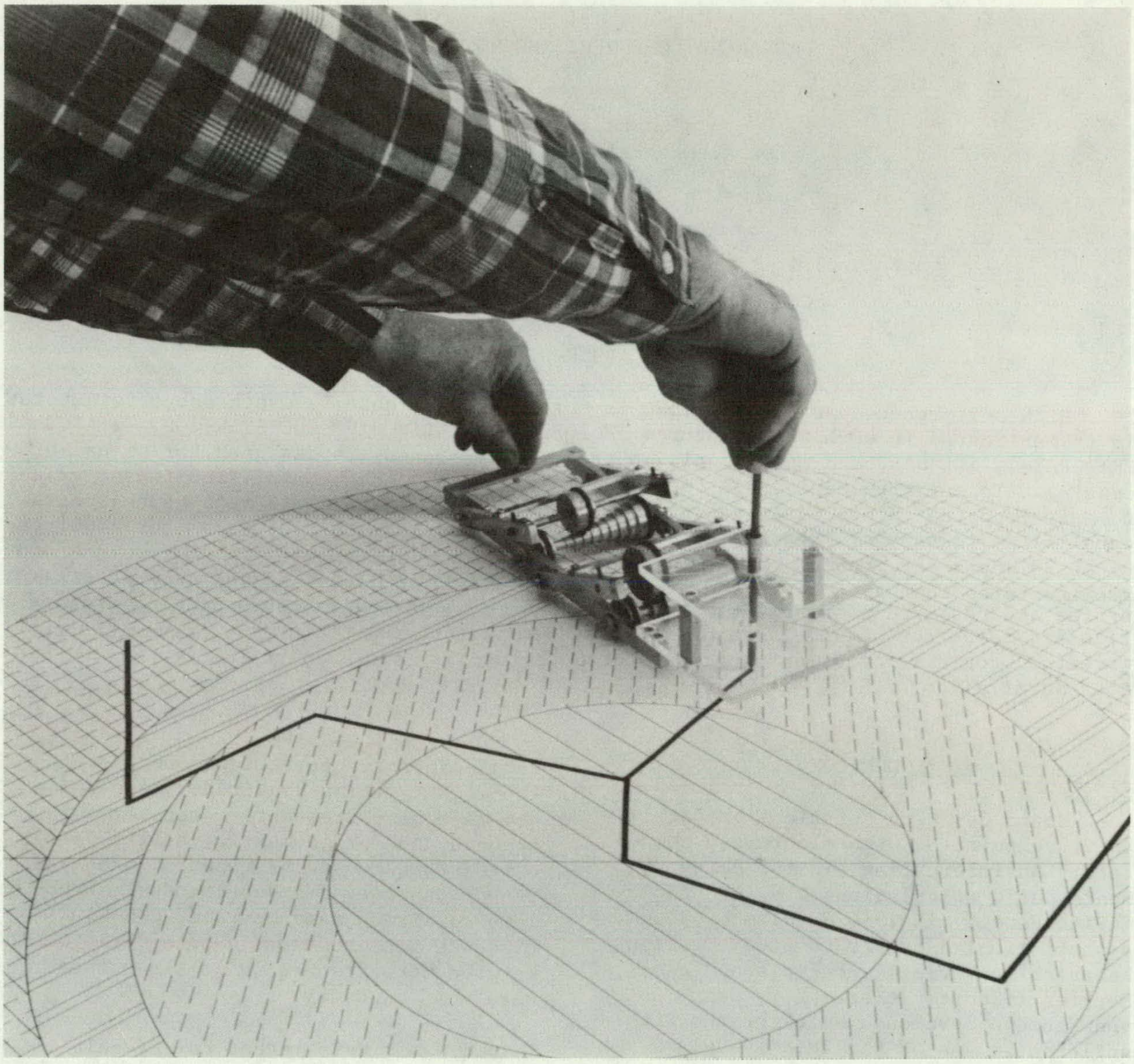


Fig. 1. Fermi's hand-operated mechanical computer.

veloped a model of a production code to compare its properties and resource requirements with those of some asynchronous multiprocessor architectures.

CODE CHARACTERISTICS

The code does not lack for parallelism; in fact, the total computation is almost completely parallel. The tracking of each particle and its progeny is independent of all other histories. Only the accumulation of statistics and the use of statistics in sample biasing couple one particle to another. This is not to say that the computation performed on each history is the same. The randomness and variety of possible events make it very difficult to develop and process queues of similar computations. Today's codes are fraught with conditional and case statements that define the many possible reactions in complex and varied geometries. Given the physical input parameters, the whole calculation is a function of a single random variable.

One must be careful not to extrapolate this computational independence to all codes that might use the Monte Carlo method. As particle dependence or coupling between particles increase, the lengths of the chains of independent computation may decrease. This will in turn decrease the efficiency of multiprocessing. At another extreme some problems in which there is tight coupling (for example, a many-body problem in which one particle's behavior affects every other one similarly) have been highly vectorized.

The code is an ideal example of asynchronous parallelism. In an environment of n asynchronous multiprocessors, the code could be easily modified so that each processor could independently process 1/nth of all the particles before rejoining to accumulate statistics. When the processors have finished their tasks, one or more processors could be used to accumulate total statistics. We believe it is feasible, with little additional work, to perform this calculation on four CDC 7600s coupled only through a common file system. Only concern for total system reliability deters one from doing this in order to speed up the calculation for the more time-consuming problems.

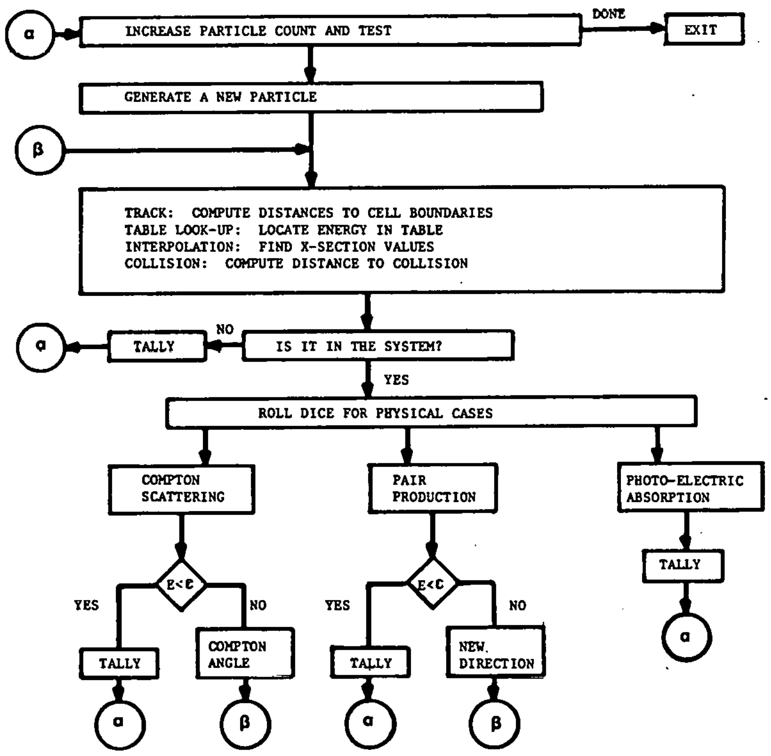
A SIMPLE MODEL

To present the flavor of this Monte Carlo code, an extremely oversimplified model is presented in the form of a flow diagram in Fig. 2. This code accumulates statistics on the behavior of a photon source in a cylindrical drum of carbon. This code is of value only for benchmarking and analysis.

To study the efficiency of this problem on an n-processor system, we took a small number of particles (1024); partitioned them into subsets of 2^k, $k = 0,1,\ldots 9$; and timed the computation of the subsets. If t_{max} is the maximum time for all subsets of size $1024/2^k$ and t_{total} is the sum of these times, then the efficiency is

$$\varepsilon = 1 - \frac{2^k \cdot t_{max} - t_{total}}{t_{total}}$$

The results are shown in Table 1. Hence, we conclude that for particle numbers of interest that the problem efficiency



α
INCREASE PARTICLE COUNT AND TEST
DONE
EXIT
GENERATE A NEW PARTICLE
β
TRACK: COMPUTE DISTANCES TO CELL BOUNDARIES
TABLE LOOK-UP: LOCATE ENERGY IN TABLE
INTERPOLATION: FIND X-SECTION VALUES
COLLISION: COMPUTE DISTANCE TO COLLISION
TALLY
NO
IS IT IN THE SYSTEM?
YES
ROLL DICE FOR PHYSICAL CASES
COMPTON SCATTERING
PAIR PRODUCTION
PHOTO-ELECTRIC ABSORPTION
E<ε
YES
NO
TALLY
COMPTON ANGLE
NEW. DIRECTION

Fig. 2. Flow diagram of a simple Monte Carlo model.

Table 1. Multiprocessor efficiency for 1024 particles distributed over 2^k processors.

k	No. of Processors	No. of Particles / No. of Processors	Efficiency
0	1	1024	1.0
1	2	512	0.97
2	4	256	0.95
3	8	128	0.95
4	16	64	0.90
5	32	32	0.91
6	64	16	0.85
7	128	8	0.74
8	256	4	0.70
9	512	2	0.22

would be very high. In that sense the problem has ideal asynchronous parallelism.

ASYNCHRONOUS SYSTEMS AND THE PRODUCTION CODE

A number of commercially available asynchronous processor systems appear likely to emerge in the not too distant future. Will these systems provide the speed and facilities needed for the Monte Carlo Production code? To get at this question, we will describe some properties of the code that will help determine the adequacy of various architectures to support this computation.

Note that in our crudely coupled system of four CDC 7600s, the total code and data have been replicated entirely. The only benefit to using such a system is to reduce the total elapsed time. Of course this could be sufficient justification for using such a system despite the four-fold amplification of all other costs. In particular our memory requirements have grown from 0.5M words to 2M words. Most of the data, x-section or probability data, is read-only data except for problem initialization. It is read relatively infrequently and can reside in a comparatively slow bulk memory. However, disk storage is too slow. A typical storage requirement might be as given in Table 2.

Obviously only one copy of the read-only data is needed even though all processors must access it. Replication of all data would unduly burden systems with many processors.

One of the important characteristics of this problem is that the data memories are very randomly accessed.

Table 2. A classification of storage requirements.

	Type	Amount (words)	Complexity Level Based on Data Replication
A.	x-sections and constants (read-only)	400,000	0 (all)
B.	code (read-only)	30,000	1 (all but A)
C.	tally (read/write)	20,000	2 (all but A + B)
D.	other (read/write)	4,000	3 (D only)

This helps to avoid memory conflicts. However, it decreases the advantages of cache and virtual memories. Caches for code instructions will be increasingly important as the number of processors grow. However, there are comparatively few DO loops in this kind of code. Consequently, any local piece of code has a relatively low duty cycle. Even in the code concerned with geometry (particles intersecting surfaces), one heavily used DO loop simply controls access to formulas for different types of surfaces. Hence the code selected here is usually different each time through the loop.

The group responsible for the actual production code ran a sampler to identify the calculations that consume the most time. Some are included in Table 3.

Table 3. Times for various subroutine calculations.

Function	Percent
Find minimum distance to cell surfaces	33.9
Compute x-sections and locate isotopes	20.5
Find new cell particle entered	10.6
Tally contributions to detectors	8.9
Table look-up of energy	8.6
Subroutines: RANF, EXP, LOG, SQRT	3.0
Total	85.5

Finally, we note that MIPS (millions of instructions per second) are more important to this problem than MFLOPS (millions of floating point operations per second). Table 4 contains dynamic measurements of the utilization of various classes of instructions on a CDC-7600 computer. Tom Keller of the LASL Computer Science and Services Division used an on-line instruction monitor to obtain the statistics tabulated in Table 4.

Table 4. Operation Mix (dynamic measurement).

Operation class	Percent
Increment	54
Jumps	12
NO OP	9
Boolean	5
Shift	4
Other (including floating point)	16
Total	100

HARDWARE

Asynchronous multiprocessors appear most promising for this particular problem. Carnegie Mellon University's CM* is a working research model of such a computer. Siemens of Germany has a working prototype and has plans for a machine called the SMS-3. At least two of the machines proposed for the Numerical Aerodynamic Simulation Facility (NASF) can operate asynchronously. The proposed Burroughs computer can use its processors either in lockstep or independently. The Texas Instrument (TI) proposal is based on the data-flow principle, which in theory is a complete captor of parallelism. They have a working 4-processor model. A few other manufacturers have discussed plans for asynchronous computers. However a description of their computers is not yet in the public domain.

There is insufficient information on a large scale TI machine to determine whether their computer is appropriate for this Monte Carlo problem. Hence we are able to analyze only the Burroughs NASF and the SMS-3 for feasibility. We do so relative to a system of four CDC 7600s and supply some relevant data in Table 5.

The SMS-3 does not appear to be useful for this problem. The amount of memory directly accessible to each processor is at best marginal for code and read/write data. We see no way to make the read-only data directly accessible to each processor. This computer ships data between processors in synchronized or phased bursts. This method of transferring data is not at all suitable for the random accesses required of the x-section data. Despite the asynchronous operation of the processors, this computer still seems most useful for synchronous parallelism or, at least, a problem processing data in a regular manner.

The Burroughs NASF machine appears to be suitable for this problem. However, more work will have to be done than would be necessary for a 4-CDC 7600 system. Note that with 50M words of memory available in the NASF machine, the problem will not fit a 512-processor system with 0-level of replication (all data and code replicated). Hence, the operating environment would have to be

Table 5. Comparative data of some multiprocessor systems.

	4 CDC 7600s	Burroughs NASF	Siemens SMS-3
No. of Processors	4	512	128
Total speed (MFLOPS)	16	1000	18
Local Memory/Processor	512K	32K	64K (BYTES)
Global Memory	DISK	33M	HOST

modified to reproduce certain variable storage and not others. Less than 100 processors could be used if all data were replicated. This allows at most 20% efficiency in using the system in this manner.

PARALLELISM AND SYNTAX

What sort of parallelism do we expect to capture with asynchronous multiprocessor systems not capturable with synchronous devices? Isn't there competition between what might be called local parallelism and global parallelism? To get at these questions, let us first try to identify some easily recognized forms of parallel activity.

1. Array Computation
```
DO   I = 1, N
   x(I) = f(I)
```
2. Local Independent Task
   a. z = u*v + x*y
   b. x = u*v
      y = w*z
   c. bookeeping overlapping computation
3. Global Independent Tasks
   a. JOB A, JOB B
   b. Monte Carlo type problem

We assume there are no fundamental differences in parallel processes when expressed in term of a dependency graph. All of the independence exists in the total graph. However, we must believe that if we have finite resources to perform various parallel tasks that there will be competition for these resources by the different independent tasks. Will the right subgraphs be selected? We doubt very seriously that much efficiency will be gotten from this problem for many years with an approach other than level-0 replication unless there is some new syntax in which the programmer can tell the operating system how to capture the important parallelism. Consequently, we must think about language features that are needed to direct the operating system.

In contrast to what is done today, we do not want to spend our resources on the inner loops or local portions of this problem. Instead it is the big outer loop with largest payoff. In fact, it is not a DO loop at all. Hence the dependence on the loop index is absent and, therefore, implicit. Certainly global compilation is required if such global parallelism is to be captured automatically.

Somehow we must make it easy to increase the dimensionality of a code. Currently, we introduce DO loops and increase the rank of dimensionality of the appropriate variables as a convenience to the compiler only to have the compiler remove them. This is not practical over a code that requires 30,000 words of instructions. This kind of requirement merely reinforces the long overdue need for an array syntax. Not only must the dimensionality of the data be increased but that of the code itself if different copies of the code are required for each processor. This is just task spawning. What do we need to specify in a task spawning statement? Once many tasks are active they must be told to rejoin and collapse to sequential mode.

Tallying is the major obstacle to achieving full parallelism and is representative of the more general problem associated with vector reduction operations. If the tally data is replicated and only later do we tally the subtallies, then we have vastly simplified the problem. This is the case for replication levels 0, 1, and 2. If for storage reasons we cannot afford this replication, then the tallying process must be done by a single processor if only one copy of the tally data is allowed. In such cases it would appear that re-entrable programs may be needed to allow queueing of tallies without choking the system at this point.

CONCLUSIONS

Given only a glimpse of future asynchronous architectures and a Monte Carlo application that has ideal asynchronous parallelism, we are yet unable to estimate the effort that will be required to fit the problem to the machine. We have observed that memory requirements may be exorbitant in those cases where the processor count is large. In addition, it is unlikely that the more profitable global parallelism will be discovered and selected automatically by the compiler. We believe that new syntax will be needed to assist the programmer: (1) in describing the parallelism available to the compiler and (2) making it easier to increase the dimensionality of the problem without rewriting code.

DETAILED VECTORIZED REACTIVE FLOW SIMULATION ON THE TEXAS INSTRUMENTS ASC

J. P. Boris, D. L. Book, T. R. Young, Jr., E. S. Oran
and M. J. Fritts
Laboratory for Computational Physics
Naval Research Laboratory, Washington, D. C. 20375

ABSTRACT

Detailed modelling, also known as numerical simulation, provides a description of a reactive system by solving numerically the governing time-dependent conservation equations for mass, momentum and energy with source and sink terms. Empirical submodels are only incorporated when the quantities required must be derived from more fundamental models or theories. This is the case for chemical rate constants, for thermal conductivity coefficients, and for other thermophysical and thermochemical data in a detailed reactive flow calculation. There are four kinds of problems in simulating accurately the propagation of a shock or a flame front in a reacting medium. One stems from the widely different time scales characteristic of the interacting fluid and chemical processes. Another arises because conventional numerical methods are unable to resolve accurately the characteristically steep spatial gradients in pressure, density, and temperature. Two others are associated with the twin problems of physical and geometric complexity, which can cause calculation times to increase by orders of magnitude compared with idealized or empirical models. The approach taken in the Reactive Flow Modelling program at the Naval Research Laboratory is to treat the fundamental processes of the problem individually and then to combine them with due concern for the way they interact. This operator-split design philosophy requires "asymptotic techniques" when there are short time scales which we do not wish to resolve.

Numerical techniques developed at NRL following this modular "asymptotic" approach include CHEMEQ,[1] for the solution of stiff ordinary differential equations, the Slow Flow[2] and ADINC[3] algorithms for handling flame propagation problems where it is too cumbersome and costly to treat sound waves explicitly, and DFLUX[2] for the accurate solution of coupled multi-species mass diffusion fluxes. ADNIC and Flux-Corrected Transport (FCT)[4], an explicit transport algorithm technique developed to handle supersonic flow, solve convective equations with near-optimum resolution of steep gradients and fine structure. SPLISH[5], a Lagrangian two-dimensional triangular grid technique for describing flows over complicated surfaces, has been developed to enable solution of problems in complex geometries.

In this paper, these algorithms will be discussed briefly and references to more detailed discussions will be given. Implementation in working codes will be discussed, with particular attention to vectorization and achieving maximum efficiency with NRL's Texas Instruments ASC. Illustrations will be drawn from combustion modelling work currently being pursued at NRL.

REFERENCES

1. T. R. Young, Jr. and J. P. Boris, "A Numerical Technique for Solving Stiff Ordinary Differential Equations Associated with the Chemical Kinetics of Reactive-Flow Problems", J. Phys. Chem., 81, 2424, 1977.

2. W. W. Jones and J. P. Boris, "Flame and Reactive Jet Studies Using a Self-Consistent Two-Dimensional Hydrocode", J. Chem. Phys., 81, 2532, 1977; "Flame - A Alow-Flow Combustion Model", by W. W. Jones and J. P. Boris, Naval Research Laboratory memorandum report No. 3970, July 1979.

3. J. P. Boris, "ADNIC: An Implicit Lagrangian Hydrodynamic Code", Naval Research Laboratory memorandum report No. 4022, June 1979.

4. J. P. Boris and D. L. Book, "Solution of Continuity Equations by the Method of Flux-Corrected Transport", Methods in Computational Physics, 16, p 85, Academic Press, 1979.

5. M. J. Fritts and J. P. Boris, "The Lagrangian Solution of Transient Problems in Hydrodynamics Using a Triangular Mesh", J. Comp. Phys., 31, 173 (1979).

Design Considerations for a Partial Differential Equation Machine[1]

Arvind, and Randal E. Bryant
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

Abstract

Partial differential equation (PDE) simulation provides an attractive area for the application of highly parallel computer systems. The regular and static structures of these problems and the limited data dependencies allow them to be mapped onto a system consisting of many interconnected processors. This paper presents an analysis of a program for simulating the hydrodynamic motion and heat flow in a compressible fluid. Based on this analysis, some of the issues in designing programming languages and computer architectures for PDE simulations are discussed. The data flow model of computation is seen to provide an attractive means for managing the complexity of highly parallel systems. Data flow concepts can be applied to relatively simple architectures specifically designed for PDE simulation.

Introduction

Partial differential equation (PDE) simulation has often been proposed as an ideal area for the application of highly concurrent computer architectures. The high computational requirements of these problems provide an incentive for high speed computation, while the regularity and minimal data dependencies provide hope that this speed can be achieved through parallelism.

Highly parallel computer architectures diverge from traditional, sequential computers to different degrees and in a variety of different ways. This paper examines how a computer architecture and high level programming language can be developed to achieve high performance at a reasonable cost, while maintaining programmability. Some of the architectural considerations include: how the processing resources are allocated, how the activities of the processors are synchronized, and what forms of communication are allowed between processors. Other potentially important decisions such as mechanisms for achieving fault tolerance and for input and output will not be considered.

While the above-mentioned design issues are directed toward the computer architecture, they will also strongly influence the design of the programming languages supported by the architecture. To provide reasonable programmability, the architecture must support some abstract model of computation which can form a basis for a high level programming language. For example, traditional architectures can be viewed as performing a sequence of updates to a set of memory cells, forming the basis for languages such as FORTRAN. Highly parallel architectures, however, must diverge from this model and hence will require new forms of programming languages. Thus we will discuss computer architectures and the languages for these architectures together.

We will assume the system consists of a number of processing elements (or simply "processors"), each capable of storing and executing a program and of storing data. Examples of such systems include the Irvine data flow architecture [3, 8], and the Utah data flow architectures [5, 10]. This model does not encompass the MIT data flow architecture [7] in which the functions of program storage, instruction execution, and data storage are performed by separate units. Nonetheless, much of the analysis should apply to this system as well.

(1) This research was supported by the National Science Foundation under grant MCS-7902782, and by the University of California, Lawrence Livermore Laboratory under contract no. 8545403.

The SIMPLE Code

As a focus for the study we have been studying the SIMPLE code [4], a 1500 line FORTRAN program developed at Lawrence Livermore Laboratories. The SIMPLE code is a simplified version of a program for simulating both the *hydrodynamics*, or mechanical motion, and the *heat flow*, or the conduction of heat between regions of a compressible fluid. Most of the simplifications serve only to decrease the total size of the program without decreasing the complexities of the numerical model. In comparison to other PDE simulation programs, such as for weather simulation or aerodynamic modeling, this program simulates systems undergoing very rapid changes with extremes of temperature and pressure and also with many shocks. As a result this simulation requires a more complex numerical model. The SIMPLE code may present somewhat of a "worst case" example in terms of potential concurrency and regularity of computation.

Although mechanical motion and heat conduction proceed simultaneously in the physical system, SIMPLE separates the two during each time step, simulating first the hydrodynamics and then the heat flow. The fluid is represented in a two-dimensional, Lagrangian formulation. A block diagram for the program is shown in Figure 1. During the hydrodynamics phase of a cycle, the program uses the positions x, and velocities v of the node points and the pressures p, artificial viscosities (1) q, and densities ρ of the zones to compute new positions x' of the nodes by an explicit difference method. Then new values for density ρ' and artificial viscosity q' are calculated along with intermediate values of energy $\hat{\epsilon}$. The heat conduction phase takes these intermediate energy values and transfers energy between zones to represent the flow of heat resulting in new energies ϵ' by an alternating-direction implicit difference method. It also computes a new set of zone pressures p' based on the energy. Finally, a value for the size of the next time step Δt is calculated. The time step must be kept small enough to maintain the stability of the computation [11]. This requires calculating the allowable time step for each zone and finding the minimum of these values over the entire mesh. Following the time step calculation a new cycle can begin.

(1) Artificial viscosity [11] is a computational technique used to smooth out shocks

Inherent Parallelism and Computational Requirements

An analysis of the SIMPLE program reveals the quantity of computation required and the forms of parallelism allowed for a typical PDE simulation. In SIMPLE the amount of concurrency and the data dependencies vary greatly in the different phases of the computation, because of the different numerical methods used. These data dependencies have important implications for exploiting the potential concurrency of the program.

Figure 2 shows the partial ordering on the program variables imposed by the data dependencies. This diagram omits those arcs implied by transitivity. As can be seen, the data dependencies impose a nearly linear ordering on the computations. Most of the variables, however, are two-dimensional arrays. If we consider the array elements as individual values to be computed we can study their data dependencies as well. Figure 2 shows four classes of dependencies:

local:	array element (k,l) depends only on elements (k,l) of the other arrays.
neighbor:	array element (k,l) depends on elements (k,l), $(k+1,l)$, $(k-1,l)$, $(k,l+1)$, $(k,l-1)$.
global:	a scalar value depends on all elements of the arrays.
scalar:	every array element depends on some scalar value.

As can be seen, most zone and node computations depend only on values from neighboring nodes and zones. In fact, many computations are fully localized. In only a few cases must the results of one computation be received from the neighbors before another computation can proceed. This does not take into account any sharing of program or constant data between zone computations to reduce the total storage requirements.

Figure 3 depicts the potential concurrency and computational requirements graphically for a 100 by 100 zone mesh assuming that the two equation of state calculations for each zone take two iterations on average to converge. This figure shows how the computation for one time step would proceed if unlimited processing and communication resources were available. The abscissa shows the elapsed time in units of floating point operation times (all operations are assumed to require the same time.) The ordinate shows the total number of operations proceeding concurrently, typically a small constant times the number of concurrent zone computations. The area of each shaded region then shows the total number of operations for each section of the program.

As Figure 3 demonstrates, with unbounded processing capability the heat conduction section would require 86% of the elapsed time, even though it represents only 5% of the total number of operations due to the restricted concurrency of this section. This analysis is somewhat misleading, however, because even the heat conduction section would allow approximately 220 operations to proceed concurrently. While this is substantially less than the 24,000 to 48,000 concurrent operations allowed by other sections of the program, it still exceeds the capacity of any existing concurrent architecture. The desire for higher concurrency may ultimately call for a different numerical method, but this conclusion should not be reached too hastily. Figure 3 also does not show the possible overlapping of calculations for two time steps. In SIMPLE this possible is limited, because the Δt calculation requires the results from one time step before allowing the next time step to begin.

Irregularities in the Computation

In most sections of SIMPLE, an identical set of operations is performed for every zone. These sections could be carried out by a set of processors executing identical, or at least very similar, instruction streams. Certain aspects of the program, however, perturb this regularity, requiring a different set of operations for some of the zones. Any programming language or computer architecture which cannot deal with these irregularities efficiently may exact a large penalty in programmability or performance.

Boundary calculations always cause irregularities in PDE simulations. SIMPLE only allows a limited class of time-invariant boundary conditions, and the boundaries must correspond to the edges of the rectangular state variable arrays. Nonetheless, these boundary calculations differ in their form and data dependencies from the calculations for internal zones and typically require more computation. In more complex programs, a variety of time-varying boundary conditions may be specified, and the boundaries may cause the logical representations of the state variables to have irregular perimeters and holes. Calculations for boundary conditions will prove the downfall of any language or architecture which requires an identical set of operations over an entire array or vector.

Any part of the program for which the flow of control depends on data-dependent decisions may also cause irregularities in the program. For example, in two sections of SIMPLE the root of an equation is computed iteratively for each zone. The number of iterations required for convergence will differ from zone to zone. Each iteration requires a significant amount of computation, causing large variations in the amount of computation per zone. Similarly, another section of the program approximates a function with a piecewise-polynomial curve. Computing this function first requires searching a table for the appropriate set of coefficients with a data-dependent search time. Finally, whenever an exceptional condition is encountered in the computation, such as a quantity exceeding some upper or lower limit, the program must take steps to correct this condition. Thus, the data-dependent decisions in the program can cause both small and large irregularities in the overall structure of the computation.

Programming Languages for PDE Simulation

Once the difference equations for a PDE simulation have been specified, their coding in a FORTRAN-like language proceeds without difficulty. The array data structures and DO loop control structures provide adequate expressive power for most applications. These programs, however, do not run efficiently on existing high performance computers such as the Star-100, Cray-1, or Illiac IV. The programs must be carefully hand coded (often in assembly language) and optimized before the potential of these machines can be realized. Smart compilers have failed to bridge the gap from traditional languages to high performance machines.

This disappointing performance of FORTRAN programs stems largely from a mismatch of language and high performance architectures. A

FORTRAN program specifies the computation in terms of a sequence of updates to individual memory locations. Array and pipeline computers, however, operate most efficiently when working with entire arrays or vectors. Thus, the compiler (usually augmented by a human) must try to combine and restructure sections of code to make full use of vector instructions. If vectors must be stored contiguously in the memory, further complications arise.

The difficulties in programming existing high performance machines is further compounded by their restrictive architectures. To support high level languages efficiently an architecture must lend itself to a process of abstraction in which the exact size, configuration, and speed of the hardware components are masked. The architecture must then have the flexibility to achieve reasonable performance even with less than optimal programs. Unless the architecture supports some abstract model consistently and efficiently, the programmer will be forced to resort to machine-level coding to take full advantage of the machine's power.

We believe the data flow model of computation [6] provides a suitable basis to be supported by highly concurrent architectures and upon which high level languages can be built. As a basis for high level language, the data flow model allows programs to be written which express the maximal concurrency allowed by an algorithm. Control is based solely on the availability of data rather than on the sequential ordering of program statements. Hence, only data dependencies constrain the program's concurrency.

The data flow model supports *functional* programming languages in which program statements define functions from the input operands to the output values. In a functional language a statement can be executed (i.e. the function evaluated) as soon as the input operands have been computed. Functional languages contrast with *imperative* languages in which each statement defines a command for altering some memory location, and statements must in general be executed sequentially. With imperative languages concurrency can be achieved only by removing the unnecessary sequencing constraints in the program, whereas such constraints never appear in functional programs. Functional languages which have been designed with the data flow model as their basis include Id [3] and Val [1, 2]

Functional programming languages have been stereotyped as amusing diversions for academicians rather than serious tools for expressing production scientific programs. The syntax and data structures of languages such as Lisp seem foreign to most scientific programmers. Such difficulties arise not from their functional nature but rather from the purposes these languages are intended to serve. We believe that functional languages for scientific programming can be developed which will actually simplify the task of coding and maintaining programs. Attempts at reprogramming SIMPLE in Irvine dataflow (Id) have proved quite successful.

Architectures for PDE Simulation

Some high performance computer architectures, such as the Cray-1, have achieved remarkable success while maintaining the basic single sequence control. Others, such as the Star-100 and Illiac IV have failed to live up to their expectations. While the success of the Cray-1 can be ascribed largely to the quality of its engineering, it also results from a greater tolerance of the irregularities in the program structure. The Illiac IV operates efficiently only when performing an identical operation over an entire array, while the Star operates efficiently only on long vectors. Sections of the program requiring scalar or short vector operations move at a much slower pace. As a result, programs must be painstakingly reworked to maximize their regularity, often to a greater extent than is called for by the algorithm. For example, the holes and irregular perimeter of the mesh may be filled with "null" zones to rectangularize the state variable descriptions. The Cray-1, on the other hand, achieves reasonable performance with scalar and short vector computations. As a result, it can tolerate partially vectorized programs. Nonetheless, it too requires careful optimization to achieve maximum performance.

All existing architectures have tried to achieve high performance by maximizing the regularity in the program and then exploiting the parallelism allowed by this regularity. This approach will always force the programmer to carefully think in terms of how the program fits onto the machine. This level of thinking requires machine-level coding to provide the necessary degree of control. Furthermore, many programs simply do not lend themselves to highly regular structuring. Future architectural developments must follow a new path if they are to achieve significantly higher performance and programmability.

As we have seen, PDE simulation programs potentially allow a high degree of concurrency in their execution. To exploit this concurrency effectively, a computer must be capable of concurrently executing different instructions on different data. Within this framework, one can choose from a variety of schemes for processor synchronization, resource allocation, and processor interconnection. These design decisions result in trade-offs between cost, performance, and programmability.

Processor Synchronization

The processors in the system must *synchronize* with one another in order to communicate. With *control-driven* synchronization, the processors transmit and accept values at points in time determined by external control signals. For example, with *lock-step* synchronization the processors are periodically synchronized by a central controller for the purpose of exchanging data. Between synchronization points each processor executes a small code segment based on the newly received data. With lock-step synchronization, a time-consuming computation for one portion of the mesh will cause most of the system to remain idle until this computation is completed.

In a system based on data-driven synchronization the processors independently execute their own instruction streams waiting only when data is needed from some other processor. A processor sends data to another as soon as it has been computed in a "packet" containing the data value and some identification of the data. Data-driven synchronization allows greater autonomy of processors and greater asynchrony in their operation. Small irregularities can be absorbed by nearby processors rather than cause global inefficiencies. Of course, data-driven synchronization does not guarantee that all processors will be fully utilized, but it provides an important step.

Data-driven synchronization also helps provide the flexibility of operation needed to support high level languages. By removing the global synchronization of processors we decrease the severity of the penalty paid by nonoptimal program implementations.

Processor Allocation

A large scale computer system contains a variety of resources for processing, storage, and communication. These resources must be allocated both in time and in space, with the optimal allocation depending on the configuration and speed of the system components as well as on the program itself. Thus, the subject of resource allocation is large and complex. For the purpose of this paper we will consider mainly the allocation of processing resources.

The spatial allocation of processors involves mapping the different activities to be performed onto the processors of the system. With *static* mapping, the spatial allocation is fixed before the program execution begins. PDE simulations, with their regular and well-defined structures suggest a variety of static mapping schemes such as one zone and/or one node per processor, or one row of zones per processor. As long as the size of the problem matches the size of the system, and the amount of computation per processor can be reasonably well equalized, this approach seems quite attractive.

With dynamic mapping the activities of the program are assigned to the processors as the execution proceeds. This approach would in principle maximize the utilization of the processors and allow for highly irregular and dynamically changing program structures. However, the difficulty of effectively mapping tasks onto processors and the overhead needed to perform this allocation may negate the potential benefits.

In addition to mapping the operations onto the processors, the operations of each processor must be ordered in time. This scheduling of tasks within each processor can occur either statically or dynamically. Static scheduling occurs in conventional processors where the order of instruction execution is fixed in advance. While this approach leads to simpler processor design, it is vulnerable to the same problems as lock-step synchronization when applied to multiple processor systems. Unless operations can be scheduled so that data arrives before the operation which needs them is initiated, a processor will sit idle even if it has other tasks to perform. Static scheduling within a processor would require a detailed timing analysis of the program and would fail when computations exceed their expected time. A dynamic scheduling scheme, on the other hand, involves simply maintaining a task list and executing those tasks for which the data is present. The increased flexibility and performance of dynamic task scheduling within each processor will easily offset its overhead.

Processor Interconnection Schemes

A variety of interconnection schemes have been proposed for multiple processor systems [3, 7, 9]. Rather than discussing the details of each of these designs, we shall explore some of the properties of these interconnection schemes in the context of the problem at hand.

Some interconnection schemes such as trees, rings, and Cartesian grids favor local over long-distance communication, whereas others such as the routing networks of the MIT data flow machine [7] require the same communication delay between any pair of nodes. Those favoring local communication typically require fewer components (switches and wires) and allow faster communication in the local case but are slower in the long-distance case.

As was seen in Figure 2, the SIMPLE program shows a great deal of locality between zone computations. Many computations depend only on data local to the zone while many others require data from only neighboring zones. Thus a potential does exist for exploiting the locality of communication. The mapping of operations onto processors, however, must match the locality in the program to the locality in the communication system. The degree to which this can be achieved depends on the activity mapping scheme and the type of interconnection network.

With static spatial mapping, one can easily imagine mapping adjacent zone computations onto adjacent processors. If the program size and structure does not match the system size and structure, however, complete locality cannot be maintained. For example, if the program has a 60 by 160 zone mesh, it cannot be mapped onto a 100 by 100 array of processors while maintaining the locality of the program and utilizing as many processors as possible. If resources are allocated dynamically, on the other hand, the assignment function must map operations which are likely to communicate onto nearby processors. This would greatly complicate the resource allocation problem.

Finally, even SIMPLE requires some global communication for finding a minimum over all zones and for distributing scalar values. One may also want to restrict the number of redundant copies of data or code to save storage, thereby increasing the long-distance communication requirements. Thus, the delay incurred by long-distance communication cannot be too great, although no quantitative requirements have been derived yet.

In summary, the structure SIMPLE code at first glance suggests a simple interconnected array of processors. Such a scheme would minimize the cost and naturally reflect most of the data dependencies of the algorithm. After further study, however, one realizes that global communication would probably take too long with such a scheme, and the configuration would not tolerate program structures and processor assignment schemes which do not match this connectivity. Nonetheless, a fully uniform interconnection scheme does not seem to be required, nor could its inability to take advantage of locality be tolerated. Some elaboration on an interconnected array of processors seems the most cost effective.

Conclusion

Research in data flow has been inspired largely by theoretical models of computation and languages. Hence, programming languages have been studied thoroughly, and greater consensus has been reached on their design. Exercises in programming scientific programs such as SIMPLE in high level data flow languages have proved quite promising in terms of both ease and the amount of concurrency which is shown. It has become clear that programming languages for highly concurrent systems must break away from the sequential memory update model of the Von Neumann computer and instead allow programs to be expressed in a maximally concurrent, functional form.

Research in architecture to support the data flow model, on the other hand, has not coalesced into a well-defined body knowledge. Most efforts have been directed at specific architectures with particular biases in terms of generality, performance, and cost. In studying the range of possible architectures for partial differential equation simulation, it has become apparent that data flow concepts can and indeed *should* be applied at a variety of different levels. At the lowest levels, the architecture would be specialized toward the types of problems to be solved in terms of configuration, processor allocation, and interconnection but would employ data-driven control and a dynamic scheduling of activities within processors. These classes of machines would still require a certain amount of effort in mapping a program onto a machine but would at least allow a much more abstract view than do existing high performance machines. At higher levels of sophistication the architecture would support a very abstract data flow mode and dynamically handle all problems of resource allocation. These machines would allow more general classes of programs and would be less affected by irregular program structures and less than optimal code. Which type of machine should be built depends largely on the nature of the problems to be solved, the sophistication of the user community, and the acceptable cost of a machine. For PDE simulations, with their high computational requirements and statically-defined, regular structures, a specialized machine with static activity mapping and limited processor interconnections may indeed prove the best choice.

Acknowledgements

Our work has been aided greatly by the help given by John Myers as a consultant to the MIT Laboratory for Computer Science. We are also grateful to Chris Hendrickson, Tim Rudy, and John Woodruff of Lawrence Livermore Laboratory for first developing the SIMPLE program and then explaining many of the fine points of PDE simulation.

References

[1] Ackerman, W., "Data Flow Languages," *Proceedings of the 1979 National Computer Conference*, AFIPS (1979).

[2] Ackerman, W., and J. Dennis, "VAL -- A Value-Oriented Algorithmic Language: Preliminary Reference Manual," Computation Structures Group, Laboratory for Computer Science, MIT, Cambridge, Mass. (1979).

[3] Arvind, K. P. Gostelow, and W. Plouffe, *An Asynchronous Programming Language and Computing Machine*, University of California Irvine Technical Report TR-114a (December, 1978).

[4] Crowley, W. P., C. P. Hendrickson, and T. E. Rudy, *The SIMPLE Code*, Internal Report UCID-17715, Lawrence Livermore Laboratories, Livermore, Ca. (Feb., 1978).

[5] Davis, A. L., "A Loosely-Coupled Applicative Multi-Processing System," *Proceedings of the 1979 National Computer Conference*, AFIPS (June, 1979).

[6] Dennis, J. B., "First Version of a Data Flow Procedure Language," *Programming Symposium: Proceedings, Colloque sur la Programmation*, (B. Robinet, Ed.), *Lecture Notes in Computer Science 19* (1974), 362-376.

[7] Dennis, J. B., and D. P. Misunas, "A Preliminary Architecture for a Basic Data-Flow Processor," *The Second Annual Symposium on Computer Architecture: Conference Proceedings*, (January, 1975), 126-132.

[8] Gostelow, K. P., and R. E. Thomas, *Performance of a Data-Flow Computer*, University of California Irvine Technical Report TR-127 (April, 1979).

[9] Gritton, E. C., *et al*, *Feasibility of a Special Purpose Computer to Solve the Navier-Stokes Equations*, Rand Technical Report R-2183-RC, Rand Corporation, Santa Monica, Ca. (1977).

[10] Keller, R., S. Patil, and G. Lindstrom, "An Architecture for a Loosely-Coupled Parallel Processor," *Proceedings of the 1979 National Computer Conference*, AFIPS (1979).

[11] Richtmyer, and Morton, *Difference Methods for Initial Value Problems*, Wiley Interscience, New York (1967).

Figure 1. Block Diagram of SIMPLE

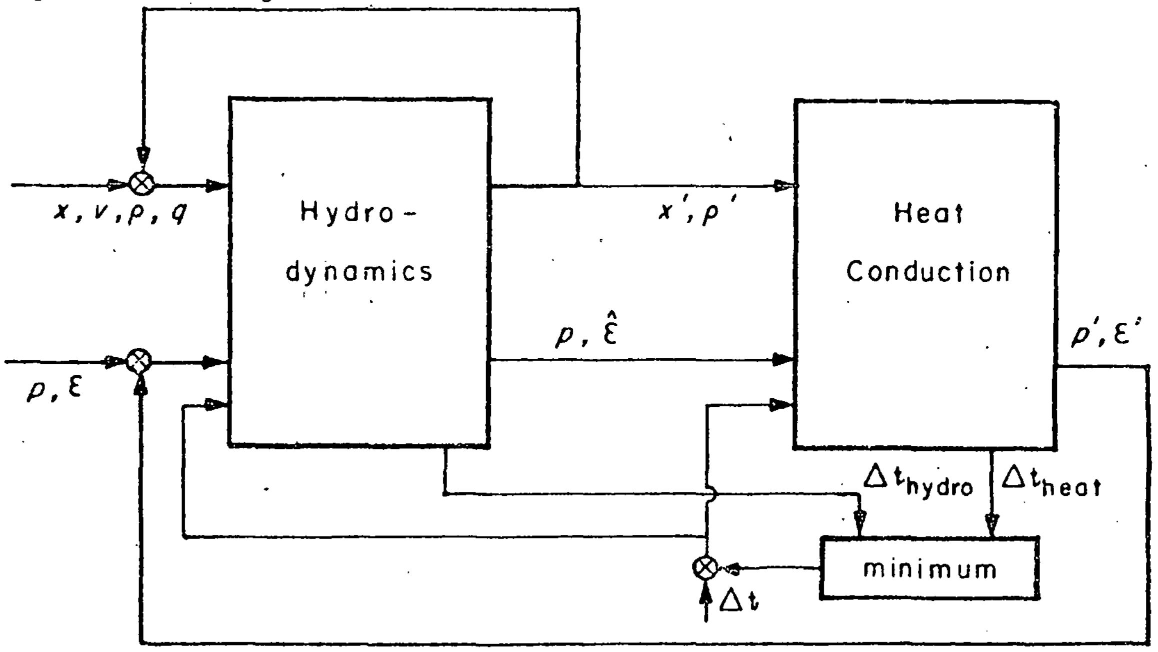
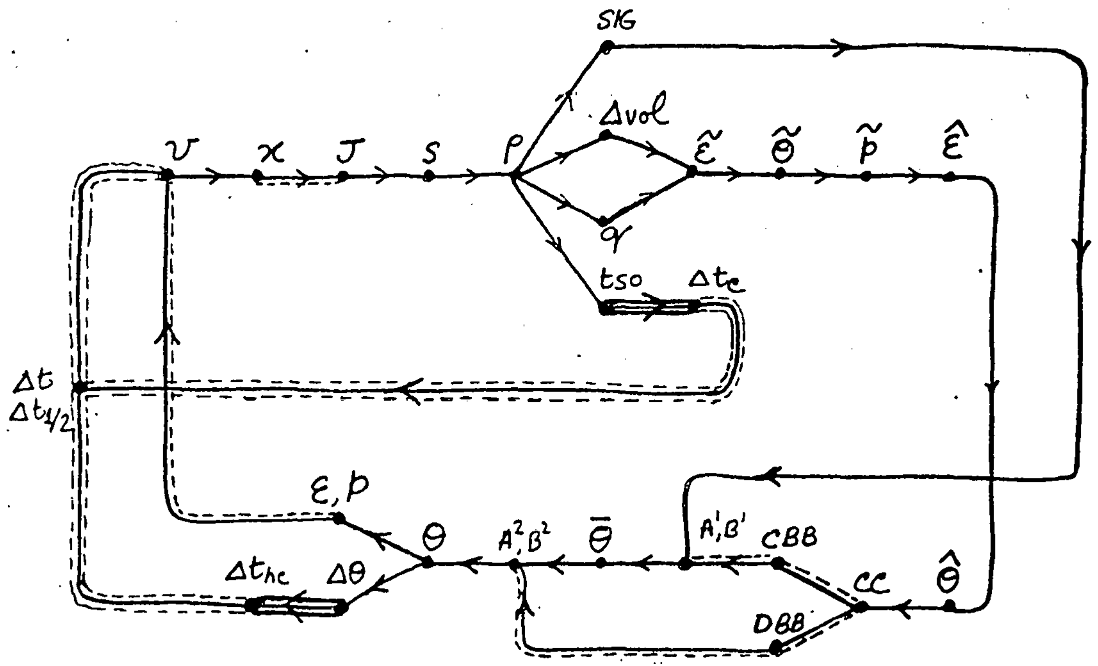
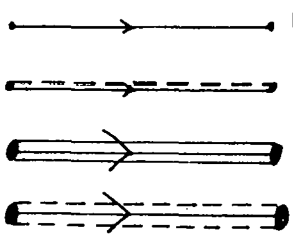


Figure 2. Data Dependencies in SIMPLE



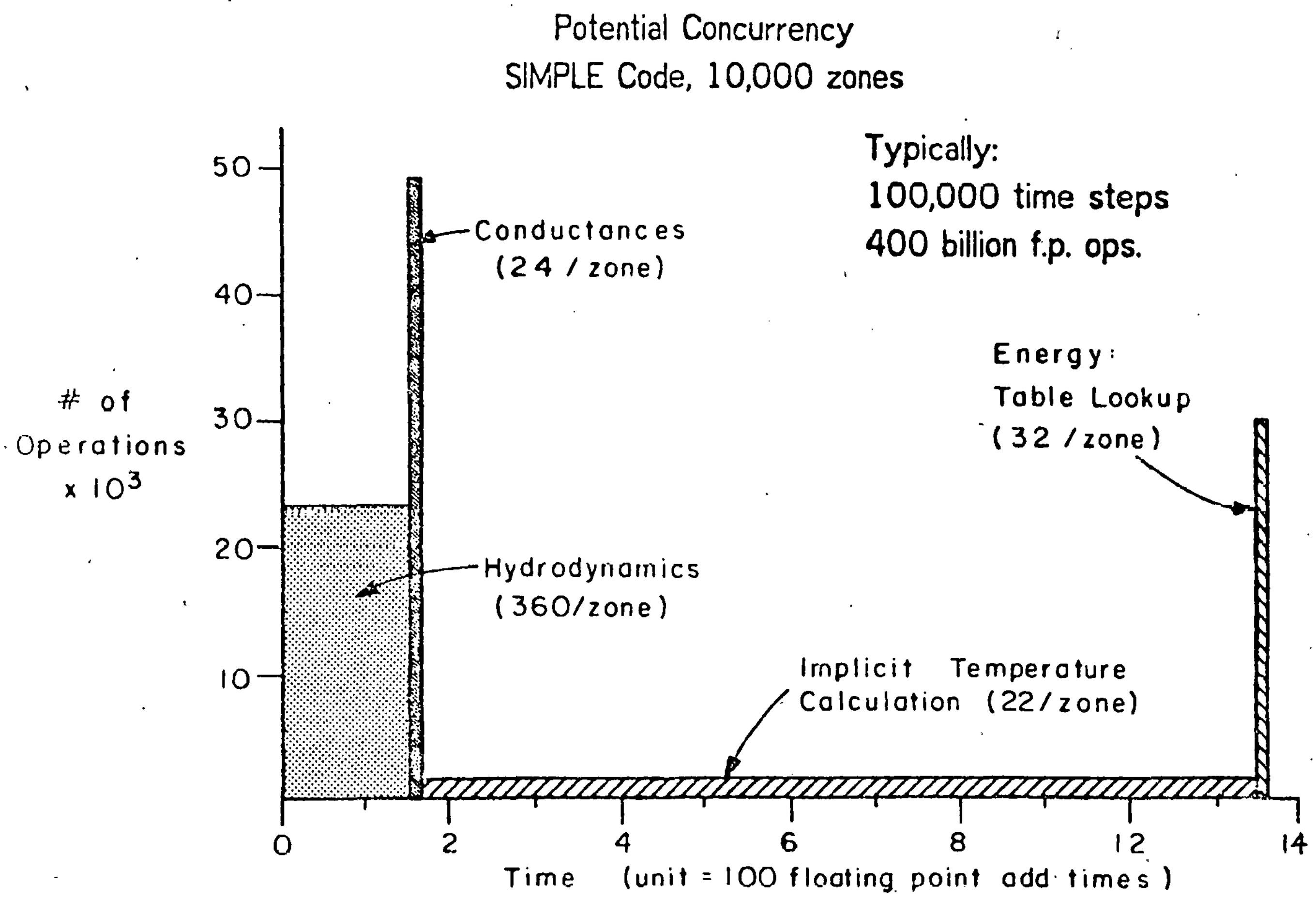

local data dependency

neighbor data dependency

global data dependency

distribution of scalar

Figure 3.



Assumptions:

kmx, *lmx* = 100.

All floating point operations take one time unit.

VECTORIZED SPARSE ELIMINATION

D. A. Calahan
Department of Electrical and Computer Engineering
University of Michigan
Ann Arbor, MI 48109

ABSTRACT

Vectorizable sparse equation solution algorithms are classified by the matrix structure which they favor. The state-of-the-art for solution of relatively dense systems is then reviewed. A hybrid vector construct is defined for the increasingly common structure of both moderate local matrix density and global matrix regularity. Estimates are made of CRAY-1 speedup achievable with this construct. A finite difference matrix is studied as an example.

INTRODUCTION

Direct solution of sparse systems has enjoyed wide application to simulation of lumped physical systems described by ordinary differential equations. Also, the last decade has seen a movement toward implicit solution of partial differential equations away from explicit procedures. An excellent example is Navier Stokes aerodynamic simulation codes, which have changed from the purely explicit, through hybrid explicit-implicit[1] and now purely implicit procedures[2].

The vectorization of direct solution portions of large codes has an immediate aspect related to the recoding of specific equation solvers for a particular architecture. Although most vector architectures have at least a minimal provision for sparse vector operations, an overhead is inevitably incurred in reduced memory bandwidth and/or the loading of associated bit maps and linked lists. It is the goal of research in sparse matrix algorithms to reduce this overhead by re-organization of the computation either (1) to obtain longer vectors, or (2) to reduce data flow, and thus achieve an overall speedup.

This paper (1) classifies sparse matrix characteristics amenable to vector processing, (2) reviews the state-of-the-art in solving certain of these problems, and (3) presents new results in the detection of vectors in patterned sparse systems. All of the experimental results were obtained from the CRAY-1; even the algorithm classifications to be made are useful only for a memory-hierarchical processor of the CRAY-1 class with a range of scalar, short vector, and long vector capabilities.

CLASSIFICATION

Consider the linear system $Ax = b$ solved by triangular factorization of A into L and U. Assume that the factorization has proceeded by outer product column-row operations so that an nxn unreduced system remains. The structure of this unreduced system--which includes fill from the completed portion of the reduction--then becomes the principal issue in determination of the sparsity algorithm to be used during the remainder of the reduction. This is an important generalization beyond examination of only the structure of A, since it suggests the use of different algorithms (polyalgorithms) as the reduction proceeds and fill increases the density of the unreduced portion.

Four sparsity structures will be considered at various parts of this paper; they are listed below to assist in unifying the later discussion. These distinguishing attributes are related to local and global sparsity characteristics:

(a) locally and globally dense, partitioned;
(b) locally dense, globally unpatterned;
(c) locally dense, globally patterned;
(d) locally sparse, globally patterned;
(e) locally sparse, globally unpatterned.

The last is the least vectorizable. Its scalar solution is probably amenable to speedup only by using a MIMD architecture[3] and so will not be discussed further.

BLOCK-ORIENTED SPARSE SOLUTION

INTRODUCTION

Two classes of relatively dense matrices benefit from solution by a general sparse solver which is oriented toward the solution of block structures. Although algorithmically less challenging than the sparser case to be studied later, such structures are becoming more common due to the aforementioned increase in the implicitness of PDE solution codes.

THE DENSE, PARTITIONED CASE

The utility of a general sparse solver in the analysis of full, banded, and other dense systems arises from vector length limitations of the processor, which in turn results from a relatively small cache memory in a hierarchial memory system. Such dense systems must be block-partitioned; in the case of the CRAY-1, these partitions must be limited in one dimension to 64, the maximum vector length of the machine. Using a general solver avoids the writing of specialized assembly language routines for dense systems with globally different density patterns but which are partitioned into locally similar 64-length or smaller dense blocks.

The processing of such large blocks with a sparse solver can be carried out on the CRAY-1 with >99.9% of the solution time in numeric kernels, and with <.1% in processing of lists resulting from the general sparsity assumption. A variety of common compressed storage schemes can also be accomodated[4].

LOCAL DENSITY

Moderate-sized dense blocks occur naturally from the representation of variable and equation coupling, from nodal coupling in a grid, and from coordinate transformations, among other causes. In the absence of other vectorization strategies (to be discussed shortly), it becomes necessary to reduce the system a block at a time with dense matrix kernels of a block-oriented sparse solver. Descriptors of the location and size of the block suffice to guide the solution of such a system[4].

The overhead of list processing of the blocks may be compensated by finely-tuned numeric kernels, with the net result that a general solver can execute at a higher rate than a conventionally-coded specialized solver[4].

The execution rate is of course highly dependent on the matrix sparsity structure. However, a timing model of the numeric kernels and the list processing overhead[4] allows the establishment of MFLOPS bounds for matrices of constant block sizes but arbitrary block sparsity patterns. Such bounds are given in Table 1 for the CRAY-1. The minimum rate is achieved with a single off-diagonal block (e.g., block tridiagonal) and the maximum with r off-diagonal blocks (Figure 1), as $r \to \infty$.

LOCALLY DENSE, GLOBALLY PATTERNED SPARSE SYSTEMS

INTRODUCTION

Table 1 shows that processing block sizes with dimensions below 10 utilizes a small fraction of the CRAY-1 processor speed. To regain a high processing rate, another structural property be-

Block sizes	MFLOPS range
2	1.9 - 7.6
3	5.0 - 17.
4	10. - 26.
6	21. - 43.
8	32. - 60.
12	54. - 84.
16	69. - 98.
32	102. - 124.
64	126. - 141.

Table 1. Performance of general block sparse system solver on the CRAY-1

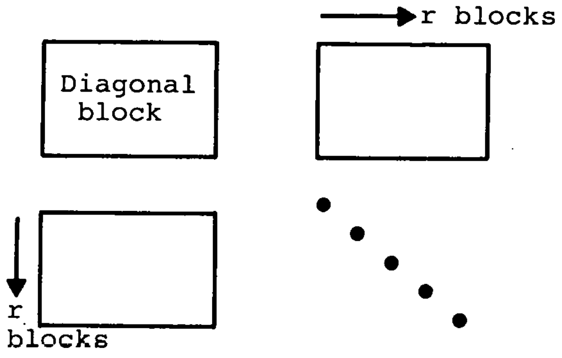


Figure 1. Model of block pivot step

sides density should be exploited. It is proposed to utilize global similarities or patterns to lengthen density-related vectors. These will be termed hybrid vectors and are the subject of the remainder of this paper.

A "bottom-up" approach will be used. After defining and illustrating the model hybrid problem, it will first be demonstrated that the CRAY-1 can achieve considerably higher execution rates on hybrid-related kernels. Then it will be shown how such hybrid vectors can be achieved with common finite difference (or finite element) structures.

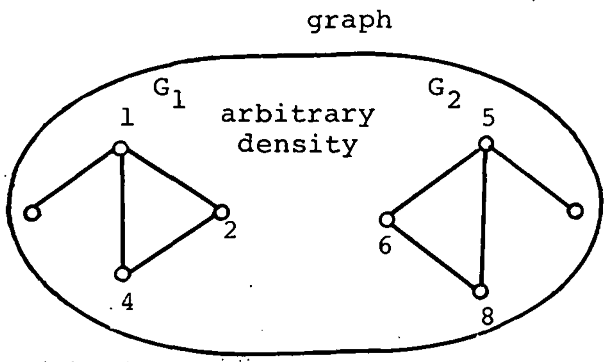
GLOBAL vs. LOCAL PROPERTIES

In establishing the hybrid vector concept, it will be useful to use the notion of the graph of a matrix.

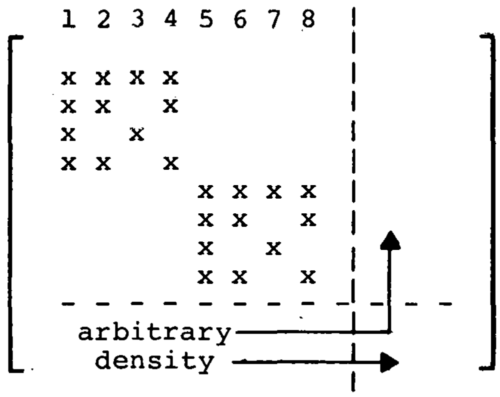
The non-zero structure of a matrix A, where A is structurally symmetric, has a convenient graph theoretic formulation. Assume that $a_{ii} \neq 0$, $i=1,2,\ldots n$. Let $V = \{v_1, v_2, \ldots v_n\}$, with the v_i termed vertices and V the vertex set. Define a set P of ordered pairs of V, called edges, by $(v_i,v_j) \in P$ if and only if $a_{ij}0$ and $k \neq j$. Then $G=G(V,P)$ is called the graph of A. Note that, because the matrix is structurally symmetric, $(v_i,v_j) \in P$ if and only if $(v_j,v_i) \in P$.

To illustrate the relationship between local and global properties, consider the subgraphs G_1 and G_2 of Figure 2(a). These subgraphs are possibly connected by paths through vertices not shown, but are assumed to be not directly connected. If the associated equations are arranged in the numbered order, the partial matrix structure of Figure 2(b) results. This structure is locally dense (contrast full) but globally sparse, since the two dense submatrices are not coupled in the northwest matrix partition. If the equations are reordered so that similarly-connected nodes are consecutively ordered, then each of the resulting 16 partitions is either a diagonal or a null submatrix (Figure 2(c)). Because most sparse blocks are coupled to other sparse blocks by diagonal coupling blocks, the matrix structure is now termed globally dense. (It may be noted that the local density pattern of each dense block of Figure 2(b) is identical to the global density pattern of Figure 2(c).) The factorization of the northwest corner of the system matrix may utilize any algorithm, independently of the algorithms used to reduce the remainder of the matrix.

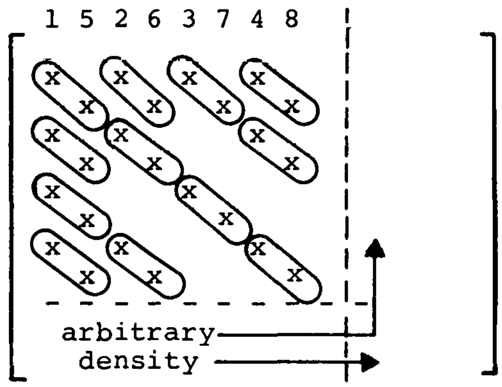
If the connection symmetry between the two sets of nodes undergoing reduction extends to their interconnections to other unreduced nodes as in Figure 3(a), and if these unreduced nodes are properly ordered as shown, then the northeast and southwest parti-



(a) Similar subgraphs (G_1 and G_2)

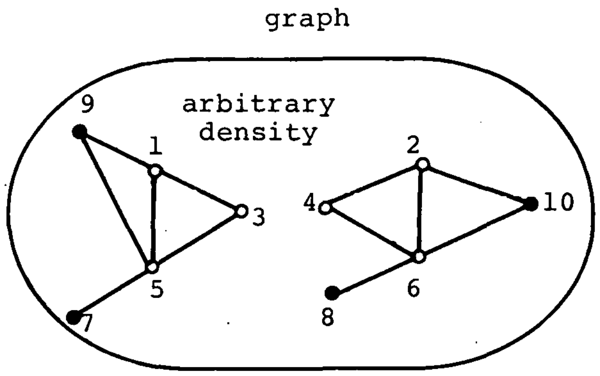


(b) Locally dense, globally sparse corner

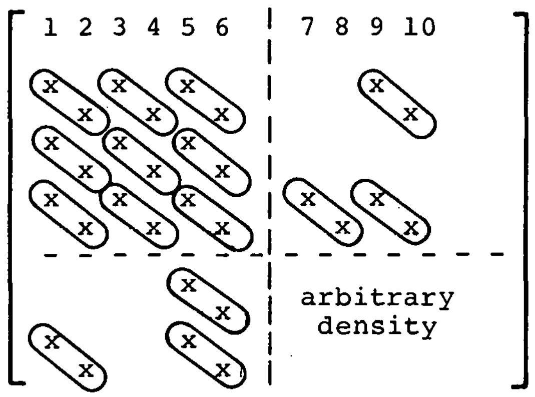


(c) Locally sparse, globally dense corner.

Figure 2. Relationships between local, global matrix properties



(a) Similar subgraphs and connections (o-node undergoing reduction; ●-unreduced node).



(b) Associated matrix

Figure 3. Similar subgraphs with similar connections to rest of graph.

tions can be made to contain similar diagonal coupling matrices (Figure 3(b).

KERNEL STUDY

In solution of large sparse systems, the multiplication/accumulation (M/A) kernel dominates other numeric kernels. Elimination of a strip of row and column blocks symmetrically coupling a diagonal block to r other diagonal blocks (Figure 1) requires (a) factorization of a diagonal block, (b) r block forward and back substitutions, and (c) r^2 multiplications/accumulations. For r=3 (a common number for dissected finite element and finite difference grids), 69% of the operations are of the M/A type. The M/A kernel therefore warrants principle study.

The nature of the M/A model kernel with both local diagonal sparsity and global density is illustrated in Figure 4. It is proposed to study the execution of the kernel

$$C \leftarrow C \pm A*B \qquad (1)$$

where A, B, and C are illustrated in the figure.

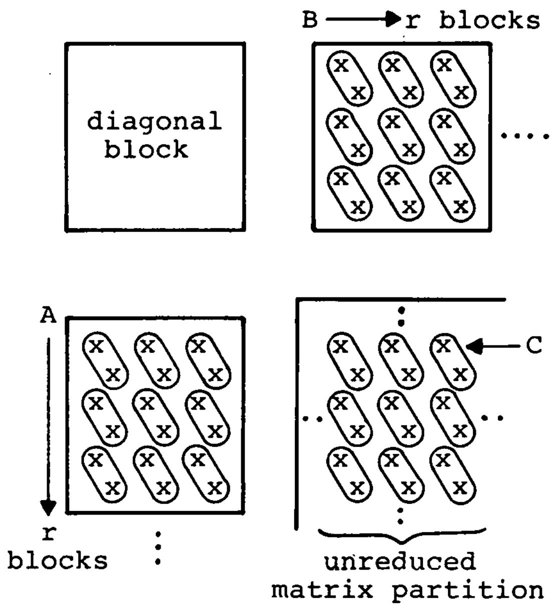
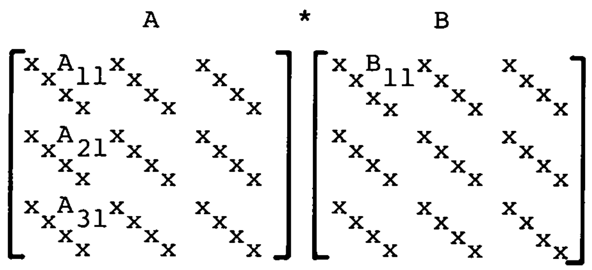


Figure 4. Example of model problem

The preference for processing hybrid kernels can be expected to arise from the interconnection or, more generally, the data flow protocol of the processor. For the CRAY-1, two recursive features of the vector registers permit high performance M/A kernels.

4-Matrix M/A. The pattern illustrated by the matrix multiply



(2)

represents, on equation reordering, the simultaneous multiplication of four 3x3 full matrices. It is proposed to implement the associated accumulation kernel by forming

$$\begin{bmatrix} C_{1j} \\ C_{2j} \\ C_{3j} \end{bmatrix} \leftarrow \begin{bmatrix} C_{1j} \\ C_{2j} \\ C_{3j} \end{bmatrix} \pm \sum_{k=1}^{3} B_{kj} \begin{bmatrix} A_{1k} \\ A_{2k} \\ A_{3k} \end{bmatrix} \qquad (3)$$

i.e., by accumulating a column of diagonal blocks of C. To perform each term of the summation by a single chained multiply-add vector operation with the CRAY-1 requires chain replications of the 4-length B_{kj} to a 12-length vector so that the overhead of the replication does not seriously impact the overall timing.

The basis of this replication is the recursive feature of the vector logical pipeline, whereby, if the same vector register is both operand and result register--usually prohibited in register allocation--data will be delayed four clocks in the pipeline and the desired replication achieved. Figure 5 gives the CAL instruction sequence and the clock level report of a part of the accumulation loop, as reproduced by a CRAY-1 timing simulator[5].

Table 2 gives the execution rates of a complete 4-matrix multiply, in comparison with the rates of two full matrix multiply kernels previously studied. The standard full kernel for short vectors produces large gaps in the floating point pipelines due to the chaining[6,7]. The high-performance matrix multiply kernel avoids chaining and the consequent gaps but suffers from register and pipeline reservations and addressing overhead resulting from four separate invocations of the full matrix multiply. Table 2 shows that nearly three times the execution rate is achieved for multiplication of four 4x4 matrices with the specialized kernel, in comparison with the standard CAL kernel.

8 Matrix M/A. A similar recursive feature of the addition pipeline allows the rapid accumulation of 8-length vectors and consequently the simultaneous multiplication of 8 matrices. This is a well known feature described in [8] and will not be discussed here. It suffices to note in Table 2 the extraordinary speedups achievable with very small matrices. However, the execution rate has a large discontinuity between n=7 and n=8, due to the nature of the algorithm, and is less desirable beyond n=7 than a 4-matrix multiply.

```
T                             FFF     V. REG   BSRRR S S. REG  A A. REG
A INSTRUCTION    P-ADDR   CP  FFFVVV           SCKKK R         R
G                             +*/&>+ 01234567  F1ABC A01234567 A01234567

5 V1  ,A0,A0        54C   628|0Z    | 50  00 |5    |   4     |         |
6 A3  A0+A3         54D   629|0Z    | 50  00 |5    |   4     |    6    |
                          630|0Z    | 50  00 |5    |   4     |6   6    |
7 A3  A3+A1         55A   631|0     | 50  00 |5    |4  4     |    7    |
8 VL  A2            55B   632|0     | 50  00 |5    |         |7   7    |
9 A0  A0+A5         55C   633|0     | 50  00 |5    |         | 9       |
A A5  A5+A6         55D   634|0     | 50  00 |5    |         |99    A  |
                          635|0     | 50  00 |5    |         |A     A  |
B A5  A0+A5         56A   636|0     | 5   0  |     |         |      B  |
C V0  ,A0,A0        56B   637|0     |C*   0  |C    |         |B     B  |
                          638|0     |C5   0  |C    |         |         |
                          639|0     |C5   0  |C    |         |         |
                          640|      |C5   0  |C    |         |         |
                          641|      |C5   0  |C    |         |         |
D V3  V3!V1&VM      56C   642|   D  |CD D 0  |C    |         |         |
                          643|   D  |CD D 0  |C    |         |         |
                          644|   D  |CD D    |C    |         |         |
                          645|   D  |CD D    |C    |         |         |
E V4  V3*RV0        56D   646| E D  |ED EE   |C    |         |         |
F A0  A0+A4         57A   647| E D  |ED EE   |C    |         | F       |
G A4  A4+A1         57B   648| E D  |ED EE   |C    |         |FF   G   |
                          649| E D  |ED EE   |C    |         |G    G   |
                          650| E D  |ED EE   |C    |         |         |
                          651| E D  |ED EE   |C    |         |         |
                          652| E D  |ED EE   |C    |         |         |
                          653| E D  |ED EE   |C    |         |         |
                          654| E D  |ED EE   |C    |         |         |
H V6  V7+FV4        57C   655|HE D  |ED EH HH|C    |         |         |
I VL  A1            57D   656|HE D  |ED EH HH|C    |         |         |
                          657|HE D  |ED EH HH|C    |         |         |
                          658|HE D  |ED EH HH|C    |         |         |
                          659|HE D  |ED EH HH|C    |         |         |
                          660|HE D  |ED EH HH|C    |         |         |
                          661|HE D  |ED EH HH|C    |         |         |
                          662|HE D  |ED EH HH|C    |         |         |
                          663|HE D  |ED EH *H|C    |         |         |
                          664|HE D  |ED EH HH|C    |         |         |
...                             .......
                          704|HE D  |ED EH HH|C    |         |         |
                          705|HE D  |ED EH HH|     |         |         |
J V1  ,A0,A0        60A   706|HE D  |EJ EH HH|J    |         |         |
```

Figure 5. Simulator output for 4-matrix accumulation instruction sequence for VL = 64. V1 is replicated into V3 with VM = $0077..7_8$.

Full matrix size	Stand. full	High-perf. full	4-matrix hybrid	8-matrix hybrid**
2	7.6	8.7	17.2	22.1
4	19.	30.	54.0	55.7
6	NT*	NT	76.3	88.5
8	43.	64.	90.2	72.5
10	NT	NT	97.8	90.8
16	88.	102.	119.	--

* Not tested
** Positive accumulation only

Table 2. Execution rates (MFLOPS) of matrix multiply kernels. Subroutine entry and exit overhead is not included.

AN ALGORITHMIC OVERVIEW

An important algorithmic property of the above hybrid vector construct is that the kernel vector length is proportional to the product of factors related to (1) the local matrix density and (2) the global matrix patterns. In less precise terms, one may claim the length is the product of local coupling and global decoupling.

In solution of large initially sparse patterned systems,as the reduction progresses fill causes the coupling to increase and the decoupling to decrease, leaving the possibility that their product remains a relative constant.

Such a result could produce a very useful generalization of previous work (ref. [9][10]) where vector lengths were assumed a function of the local density only or global patterns only[11]. To lengthen vectors by increasing local density, previous algorithms were inevitably driven to an increase in the arithmetic computational complexity[10].

The following study can be considered an initial investigation into the production of hybrid vectors for finite difference grids. A number of algorithmic questions will be left unanswered, a topic for continuing research.

GENERATION OF HYBRID VECTORS

INTRODUCTION

Given the graph of a matrix, it is proposed to perform operations on this graph which yield hybrid vectors in the matrix reduction with either no increase or a determinable increase in the arithmetic operation count. The example of a 5-point 2-D finite difference grid will be used to illustrate the procedure, because of its connection regularity and because its solution by nested dissection is characterized by exploitation of decoupling to achieve a reduced arithmetic operation count for large grids. The reader is assumed to be familiar with this dissection process[12,13].

FOLDING AND ROTATION

The (diagonal) nested dissection of a 5x5 grid proceeds by recursively dividing the grid into quadrants until each quadrant consists of a single node. This division is performed along diagonal separators, which are lines of nodes whose removal divides the graph into unconnected parts.

It is clear from Figure 6(a) that, since the quadrants have a similar structure, "similarly-positioned" vertices not on a separator may be eliminated simultaneously with vector operations, without increasing arithmetic computation. These vectors will be of length four, as required for the 4-matrix kernel of Figure 5.

"Similarly-positioned" nodes can be generated by overlaying the quadrants so that a single node in the overlay represents 4 nodes. This single-quadrant representation of the 4 quadrants may be achieved by folding or rotating the original graph. This rotation process is illustrated in Figure 6(a)-(b); a recursive folding process - which generates vectors of decreasing length - is discussed in ref. [14].

Because the non-separator nodes are eliminated first in the nested dissection process, these interior nodes are represented by the northwest corner of the system matrix; this corner is consequently guaranteed to consist of 4x4 blocks with either diagonal or null structure. The southeast corner of LU, representing the reduction of the separator nodes, is dense and can be reduced at execution rates exceeding 100 MFLOPS. The northeast and southwest partitions, however, represent coupling between the separators and the interior nodes of the quadrants. Asymptotically in the grid dimensions, operations involving these two partitions consist of approximately 30% of the total, so that the choice of a proper kernel is important (Figure 7). Irregularity in these coupling matrices results in part from the separator nodes shared by Q1 and Q4 (nodes #1, #7 and #13 in Figure 6(a)) in the rotation sequence. The regularity may be restored by cutting the graph along the boundary, adding nodes and associated unknowns, and adding equations that relate the new nodes to the originally shared ones. This cutting process is illustrated in Figure 6(c)-(d).

The structure of L and U resulting from application of such cutting to a 17 x 17 finite difference grid is shown in Figure 8(b). The conventional nested dissection ordering for the same matrix yields the LU map of Figure 8(a). Coupling in the northwest partition appears as 4x4 diagonal blocks, as predicted; coupling in the northeast and southwest partitions appears as 4-length stripes, but not necessarily as 4x4 diagonal blocks. Thus a somewhat modified 4-matrix accumulation kernel would have to be used. The addition of the 8 nodes along the cut also increases each dimension of the dense southeast corner of LU by approximately 25%. The total increase in computation resulting from cutting is as yet undetermined.

SUMMARY

Vectorization and data flow

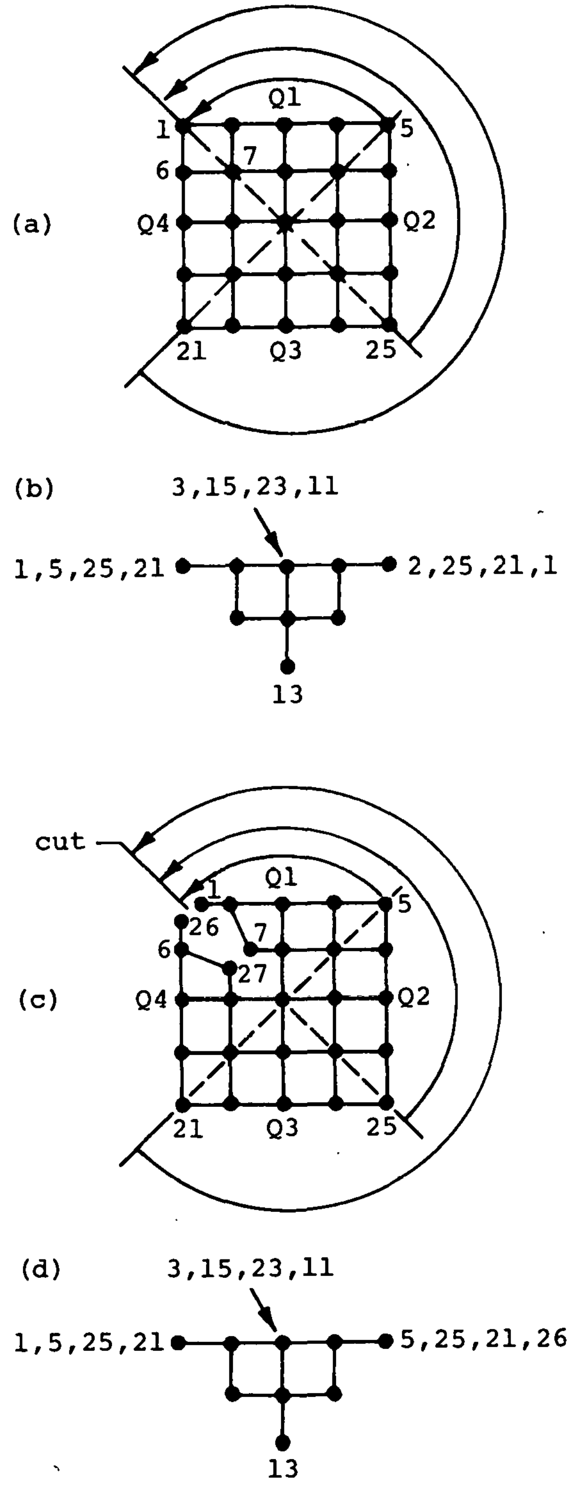
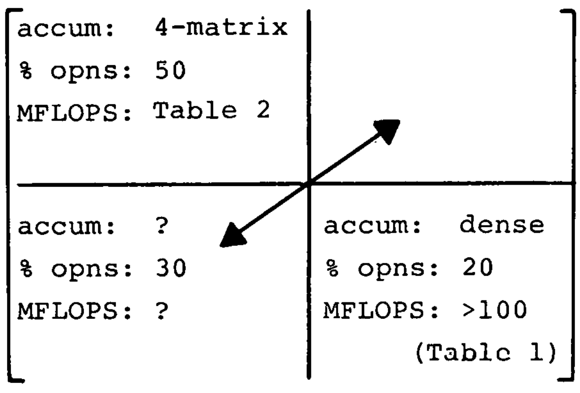


Figure 6. Rotation of quadrants (a) into single-quadrant representation (b); rotation with cut and creation of nodes ((b)-(c)).

for a memory hierarchical processor add two new issues to be considered in the development of codes for the direct solution of 2-D finite difference grids. What is a single algorithm class--nested dissection--for a scalar machine now divides into subclasses of algorithms, of which the above proposal is only one. Checkerboard and related ordering strategies are also attractive; preliminary estimates indicate such codes will execute over 100 MFLOPS on the CRAY-1[15], which can partially compensate for increased arithmetic computation.



*Approx. percent of total arithmetic operations.

Figure 7. Estimated asymptotic perf. of polyalgorithm to perform nested dissection in four matrix partitions.

SPARSE, PATTERNED SYSTEMS

As the coupling in A, B, and C of Figure 4 decreases, each approaches a diagonal matrix. The accumulation then involves at least two vector loads (and usually one vector store) for each floating point M/A operation and sufficient list processing to locate at least two of the matrices in memory. An accumulation kernel written for the CRAY-1, including list processing and a vector store for each accumulation, executes at the rate

$$\text{MFLOPS} = 53.3 \left(\frac{1}{1 + 31.3/\ell}\right)$$

with the maximum value of 35.8 for $\ell = 64$. This is less than 1/4 the asymptotic rate of a dense accumulation. The kernel is memory bound and involves significant start up time for the relatively small floating point computation involved.

CONCLUSION

While the speed of vector processors encourages the formulation of denser systems, their increasingly parallel design favors the construction of longer vectors that can be distributed across many pipelines operating concurrently. In this paper, the vector-lengthening advantages of the hybrid vector construct have been shown at the kernel level and methods have been proposed to produce such vectors directly from the problem structure.

From the algorithm viewpoint, the direct relationship between problem and processor structure offers novel insight possibly useful in developing a family of equation ordering techniques based on folding, rotation,etc. It is also hoped that a high performance software package may be developed for specific 2D grid geometries.

From the viewpoint of processor architecture, this paper has quantified the motion that the less dense the system, the more data flow and other accumulation kernel overhead is required. A patterned system may permit the lengthening of vectors - which reduces the influence of overhead - but does not significantly alter the data flow problem.

ACKNOWLEDGMENT

The author acknowledges the programming assistance of M. Yatchman, and the insight provided by a CRAY-1 simulator developed by D. A. Orbits. This work was sponsored jointly by the Mathematical and Information Sciences Directorate of the Air Force Office of Scientific Research, and by the Air Force Flight Dynamics Laboratory, Wright Patterson AFB under

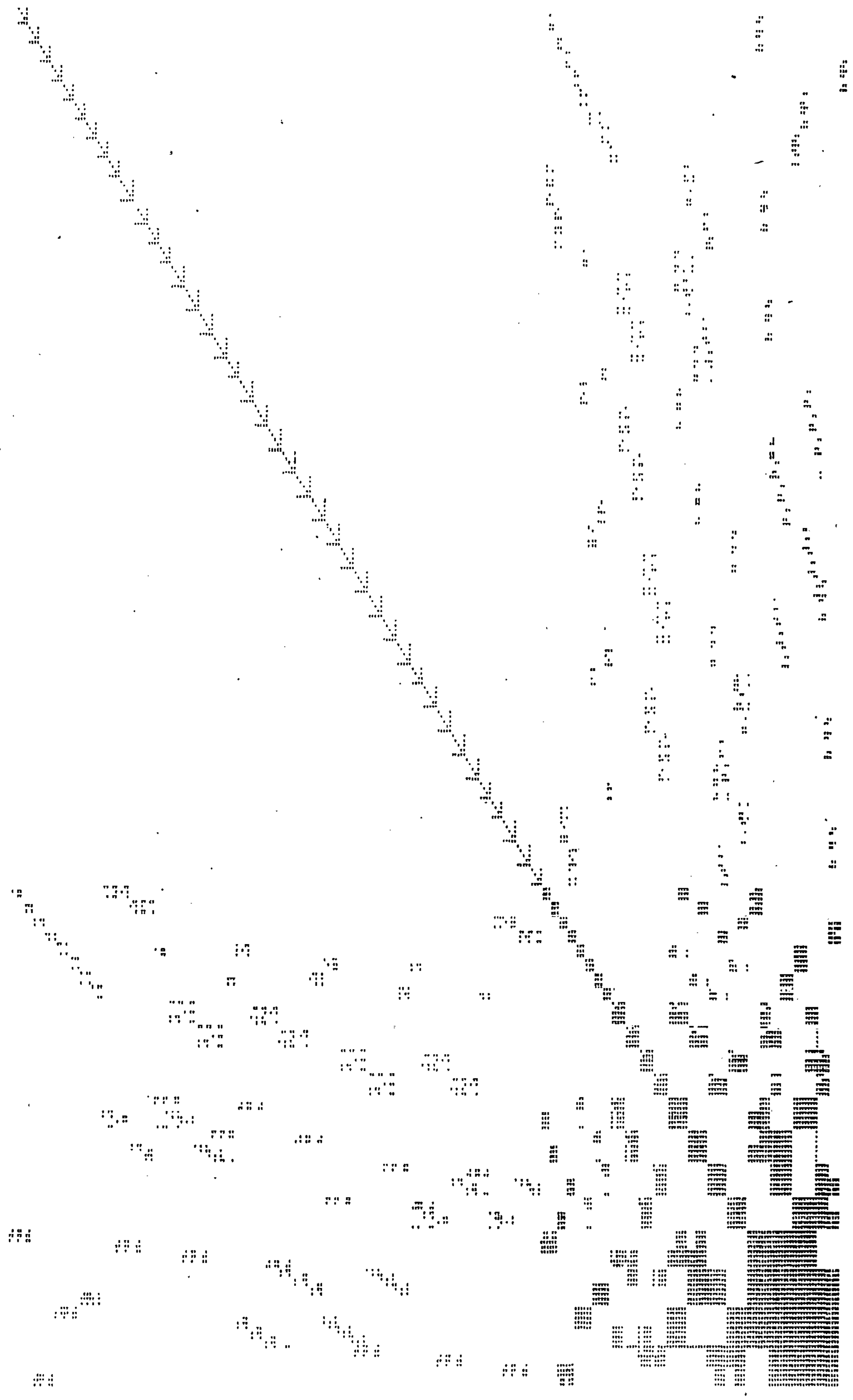


Figure 8(a). Matrix of dissected 17x17 5-point finite difference grid; before rotation.

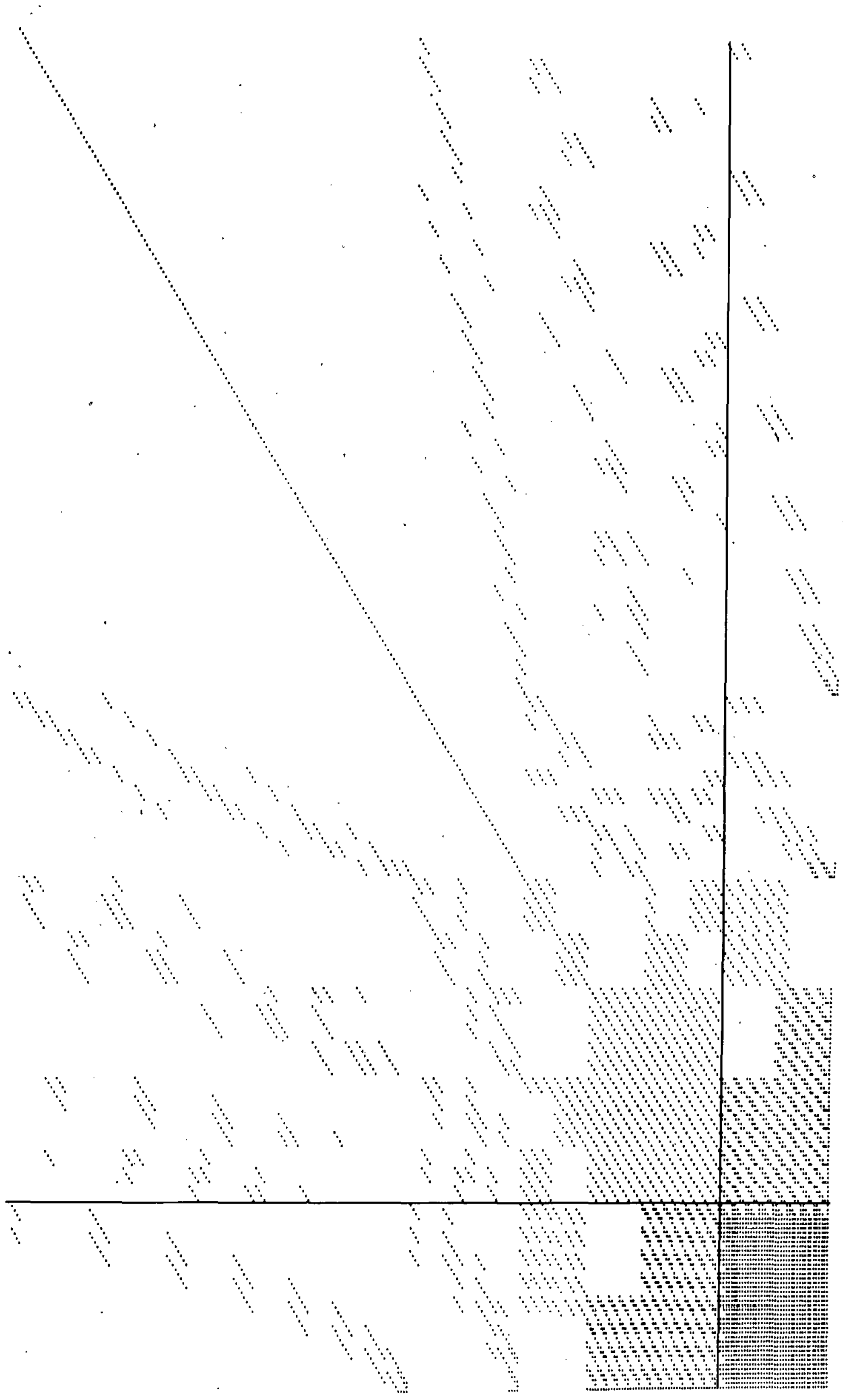


Figure 8(b). Matrix of dissected 17x17 5-point finite difference grid; after rotation.

Grant 75-2812.

References

[1] R. W. MacCormack, "An Efficient Numerical Method for Solving the Time-Dependent Compressible Navier-Stokes Equations at High Reynolds Number," NASA Report TMX-73.129, Ames Research Center, Moffett Field, CA, (July, 1976).

[2] R. M. Beam and R. F. Warming, "An Implicit Factored Scheme for the Compressible Navier-Stokes Equations II: The Numerical ODE Connection," Paper No. 79-1446, AIAA 4th Computational Fluid Dynamics Conf., Williamsburg, VA., (July, 1979).

[3] O. Wing, and J. W. Huang, "An Experiment in Parallel Processing of Gaussian Elimination of a Sparse Matrix," Proc. IEEE 1976 International Symposium on Circuits and Systems, Munich, Germany (April, 1976).

[4] D. A. Calahan, "A Block-Oriented Sparse Equation Solver for the CRAY-1," Proc. 1979 Intl. Conf. on Parallel Processing, Bellaire, MI. (August, 1979).

[5] D. A. Orbits, "A CRAY-1 Simulator," Report #118, Systems Engineering Laboratory, Univ. of Michigan, (Sept., 1978).

[6] D. A. Orbits, and D. A. Calahan, "A CRAY-1 Simulator and Its Use in Development of High Performance Algorithms," Proc. Workshop on Vector and Parallel Processing, Los Alamos Scientific Laboratory, 42-56, (Sept., 1978).

[7] W. G. Ames, et al, "Sparse Matrix and Other High Performance Algorithms for the CRAY-1," Report #124, Systems Engineering Laboratory, Univ. of Michigan (January, 1979).

[8] CRAY-1 Reference Manual, Pub. #2240004, Cray Research, Inc., Chippawa Falls, Wisc.

[9] D. A. Calahan, "Complexity of Vectorized Solution of 2-Dimensional Finite Element Grids," Report #91, Systems Engineering Laboratory, Univ. of Michigan, (November, 1975).

[10] A. George, W. E. Poole, Jr., and R. G. Voigt, "Analysis of Dissection Algorithms for Vector Computers," ICASE Report 76-17, NASA Langley Research Center, Hampton, VA (June, 1976).

[11] D. E. Barry, C. Pottle, and K. A. Wirgan, "A Technology Assessment Study of Near Term Computer Capabilities and Their Impact on Power Flow and Stability Simulation Programs," Final report on Research Project EPRI TPS 77-749, General Electric Co., Schenectady (June, 1978).

[12] A. George, "Numerical Experiments Using Dissection Methods to Solve n by n Grid Problems," SIAM J Numer. Anal., vol. 14, 161-179 (April, 1977).

[13] P. T. Woo, S. J. Roberts, and F. G. Gustavson, "Applications of Sparse Matrix Techniques in Reservoir Simulation," SPE 4544 48th Annual Fall Meeting of Soc. of Pet. Engrs., Las Vegas, Nevada (1973)

[14] D. A. Calahan, and W. G. Ames, "Vector Processors: Models and Applications," (To be published, Trans. IEEE on Circuits and Systems, Fall, 1979).

[15] D. A. Calahan, W. G. Ames, and E. J. Sesek, "A Collection of Equation-Solving Codes for the CRAY-1," Report #133, Systems Engineering Laboratory, Univ. of Michigan (August, 1979).

PARALLEL ALGORITHMS FOR SOLVING BANDED TOEPLITZ LINEAR SYSTEMS

Ahmed Sameh
Joseph Grcar
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

ABSTRACT

Such systems of linear algebraic equations arise in many applications such as the numerical solution of partial differential equations using finite difference descretization. We present several algorithms for the solution of these systems, and compare their efficiencies and numerical stabilities on a model parallel computer with a small number of processors. Assuming unlimited parallelism, however, we show that a positive definite Toeplitz system of order n and bandwidth m<<n can be solved in time $O(m \log_2 n)$ using $O(mn)$ processors.

AN EXPERIENCE WITH THE CONVERSION OF THE LARGE-SCALE PRODUCTION CODE DIF3D TO THE CRAY-1

Keith L. Derstine
Argonne National Laboratory
Applied Physics Division
9700 S. Cass Avenue
Argonne, Illinois 60439

ABSTRACT

Optimized iteration methods for the solution of large-scale fast reactor finite-difference steady state neutron diffusion theory calculations are presented. The methods utilized include the Chebyshev semi-iterative method applied to accelerate the outer fission source iteration and an optimized block successive overrelaxation method for the within-group iterations. The theoretical basis and the computational and data management considerations that enter into the formulation of the overrelaxation method are discussed. A vectorized variant of the overrelaxation method is discussed. The conversion to the CRAY-1 of a computer code employing these methods is discussed and the performance of vector and scalar algorithms on vector and scalar computers is compared for a benchmark problem.

I. INTRODUCTION

Much effort has been devoted to the development of optimized iterative methods and convergence acceleration techniques for application to the finite-differenced form of the multigroup neutron diffusion equation[1-3]. A powerful multidimensional multigroup diffusion code DIF3D[4] employs these techniques for fast breeder reactor (FBR) design at ANL; typical FBR problems require from 5×10^5 to 1.6×10^6 space energy unknowns.

In recent years the advent of advanced computing systems capable of executing upwards of 40 million floating point operations per second (megaflops) has stimulated the development of parallel algorithms exploiting the newly available vector processing capabilities. Recognizing the potential for vastly improved performance, a study has been conducted to determine the impact of the CRAY-1 advanced computing system on the DIF3D computational and data management strategies as optimized for the IBM 370/195 and the CDC 7600 computers; investigations concerned with vectorization achieved by appropriate algorithm modifications requiring minor program modifications were included.

Results of the study indicate that the unmodified scalar algorithms in DIF3D aided by an optimized CAL assembler routine can achieve a 3 fold increase in computation speed over machines such as the CDC 7600 and the IBM 370/195. Minor modifications in which a key algorithm is vectroized yield a nearly 6-fold performance gain. Analysis of the CFT compiler code generation indicates that an 8-fold performance gain is highly probable with suitable CAL optimization to a single vector subroutine.

In Sec. II of this paper, the finite differenced form of the multigroup neutron diffusion equations is presented, along with a review of the properties of these equations which permit the applications of the iterative methods discussed in subsequent sections. The theoretical aspects of the iteration methods used for the inner (or within-group) iterations are described in Sec. III. Section IV consists of a discussion of the computational considerations that strongly influence the manner in which these iteration methods are implemented in this optimized iteration strategy. Because of the massive amount of data that must be dealt with at each iteration cycle in large FBR problems, the data management requirements of a particular iteration strategy have a strong influence on the efficiency of that strategy. Section V of this paper describes the data management considerations that have had a significant impact on the form of the iteration method described in this paper. The relatively rapid conversion of DIF3D for implementation on the CRAY-1 is discussed in Sec. VI. The minor programming changes required to achieve significant vectorization of inner iterations are discussed in Sec. VII. Section VIII concludes with a comparison of the performance of DIF3D variants of both vector and scalar algorithms on vector and scalar computers.

II. THE FINITE DIFFERENCED MULTIGROUP DIFFUSION EQUATION

The time-independent multigroup neutron diffusion equations can be written as

$$-\nabla\cdot D_g(\vec{r})\nabla\phi_g(\vec{r}) + \Sigma_g^r(\vec{r})\phi_g(\vec{r}) - \sum_{g'\neq g} \Sigma_{gg'}^s(\vec{r})\phi_{g'}(\vec{r}) = \frac{1}{\lambda}\chi_g \sum_{g'=1}^{G} \nu\Sigma_{g'}^f(\vec{r})\phi_{g'}(\vec{r}),\ g=1,2,\ldots,G. \tag{1}$$

The symbols used here and elsewhere in this paper that are not defined locally are defined in the Nomenclature. Boundary conditions for Eqs. (1) are of the type

$$D_g(\vec{r})\frac{\partial\phi_g(\vec{r})}{\partial n} + \alpha_g(\vec{r})\phi_g(\vec{r}) = 0, \quad \vec{r}\epsilon\Gamma, \tag{2}$$

where Γ is the boundary of the region of solution R.

Equations (1) are discretized in space by first subdividing the region R into a regular array of subregions or mesh cells. Then, using either the mesh-cell-centered method[5,6,7] or the mesh cell-corner method,[8] the actual finite difference equations for the appropriately defined cell-averaged fluxes are obtained. For energy group g, the resulting equations can be written in matrix form as

$$D_g\vec{\phi}_g + \Sigma_g\vec{\phi}_g - \sum_{g'<g} T_{gg'}\vec{\phi}_{g'} = \frac{1}{\lambda}\chi_g\sum_{g'=1}^{G} F_{g'}\vec{\phi}_{g'} \tag{3}$$

where $\vec{\phi}_g$ is the vector of (approximate) fluxes on the finite difference mesh. The matrices Σ_g, $T_{gg'}$, $F_{g'}$, and χ_g are $N \times N$ diagonal matrices, where N is the number of cells in the finite difference mesh. The vector $\vec{\phi}_g$, of length N, consists of the neutron flux values in each of the mesh cells consistent with the method used to finite difference the equations. For purposes of this paper, it is assumed that the unknowns in $\vec{\phi}_g$ are ordered in a linear fashion, row by row and plane by plane. Given this linear ordering, the $N \times N$ matrix D_g contains three, five, or seven nonzero stripes for one-, two-, or three-dimensional orthogonal geometries, respectively. It operates on $\vec{\phi}_g$ to yield the net leakage across the faces of each mesh cell. Note that in Eq.(3) and throughout the remainder of this paper, it has been assumed that no upscattering is present, i.e., $T_{gg'} = 0$, $g' > g$.

The G Eqs. (3) can be condensed into the single matrix equation,

$$M\vec{\Phi} = \frac{1}{\lambda}B\vec{\Phi}, \tag{4}$$

where M and B are square and of order N^*G and $\vec{\Phi} = \mathrm{col}[\vec{\phi}_1, \vec{\phi}_2,\ldots, \vec{\phi}_G]$. The matrix M is given by

$$M = \begin{bmatrix} A_1 & & & 0 \\ & A_2 & & \\ & & \ddots & \\ 0 & & & A_G \end{bmatrix} - \begin{bmatrix} 0 & & & & 0 \\ T_{21} & 0 & & & \\ \vdots & & \ddots & & \\ \vdots & & & \ddots & \\ T_{G1} & T_{G2} & \cdots & T_{G,G-1} & 0 \end{bmatrix} \tag{5}$$

where $A_g(\equiv D_g + \Sigma_g)$ is the leakage-plus-removal matrix operator and 0 is the null matrix. By defining the $N^*G\times N$ matrices,

$$F = \mathrm{col}[F_1, F_2,\ldots, F_G] \tag{6}$$

and

$$\chi = \mathrm{col}[\chi_1, \chi_2,\ldots, \chi_G], \tag{7}$$

the matrix B can be written as

$$B = \chi F^T, \tag{8}$$

where superscript T denotes the transpose of a matrix.

The matrices used in Eqs. (4) through (8) possess a number of properties that provide a sound theoretical basis for the iteration methods discussed in Sec. III. For any physically realistic set of assumptions, the diagonal matrices $T_{gg'}$, χ_g, and F_g are non-negative matrices. It has been shown[9] that the matrices A_g are irreducible Stieltjes matrices and that the inverse of each A_g has all positive entries, i.e., $A_g^{-1}>0$. Because of these properties, the matrix M is nonsingular[10] and the eigenvalue problem Eq. (4) can be written as

$$\lambda \vec{\Phi} = M^{-1}B\,\vec{\Phi} \qquad (9)$$

Under quite general conditions, Froehlich[11] has shown that Eq. (9) has a unique positive eigenvector $\vec{\Phi}_1$ and a corresponding single positive eigenvalue λ_1 greater than the absolute value of any other eigenvalue of Eq. (9). Furthermore, any positive eigenvector of $M^{-1}B$ is a scalar multiple of $\vec{\Phi}_1$.

The properties of B permit a reduction of the matrix eigenvalue problem that must be solved to obtain λ_1 from one of order N*G[Eq. (9)] to one of only order N (Ref. 12). Advantage is taken of this fact in obtaining the outer iteration method presented in Sec. III, which is used to obtain λ_1 and $\vec{\Phi}_1$. This reduction is accomplished by first noting that $M^{-1}B$ is of order N*G and therefore has N*G eigenvalues. However, the rank of F is only N, thus making the rank of $M^{-1}B$ only N. Hence, (G − 1)*N of its eigenvalues are zero. The nonzero eigenvalues can be determined by considering the reduced but equivalent problem of order N.

Following Ref. 12, but considering a full down-scattering matrix, this reduction is accomplished by first defining the fission source vector, $\vec{\Psi}$, as

$$\vec{\Psi} \equiv F^T\vec{\Phi} = \sum_{g=1}^{G} F_g\vec{\phi}_g , \qquad (10)$$

and the N*G×N matrix L as

$$L = \mathrm{col}[L_1, L_2, \ldots, L_G] = M^{-1}\chi , \qquad (11)$$

where the N×N matrices L_g are defined as

$$L_g \equiv A_g^{-1}(\chi_g + \sum_{g'<g} T_{gg'}L_{g'}) \qquad (12)$$

These definitions plus Eq. (4) permit the group flux vector, ϕ_g, to be written as

$$\vec{\phi}_g = \frac{1}{\lambda} L_g\vec{\Psi} \quad . \qquad (13)$$

Premultiplying Eq. (9) by F^T and using Eqs. (8) and (10) yields the reduced problem

$$\lambda\vec{\Psi} = Q\vec{\Psi} \qquad (14)$$

where

$$Q = F^TL = \sum_{g=1}^{G} F_gL_g \, . \qquad (15)$$

If $\vec{\Phi}$ and λ are an eigenvector and corresponding nonzero eigenvalue of $M^{-1}B$, then Ψ and λ must be an eigenvector and eigenvalue of Q and vice versa. Furthermore, by making use of a similarity transformation, it has been shown[12] that the nonzero eigenvalue spectrum of Q is identical to the nonzero spectrum of $M^{-1}B$ and that any non-negative eigenvector of Q is either a scalar multiple of $\vec{\Psi}_1$ or else corresponds to a zero eigenvalue, where Ψ_1 corresponds to λ_1. Thus the two eigenvalue problems, Eqs. (9) and (14) are equivalent.

III. ITERATION METHODS: THEORY

The solution method presented in this paper utilizes two levels of iteration, the outer or fission source iteration and the inner or within-group iteration. The outer iterations seek to determine the fundamental eigenvector, Ψ_1, and corresponding eigenvalue, λ_1, of Eq. (14) or the fundamental eigenvector Φ_1, and λ_1 of Eq. (9).

Fast reactors tend to be tightly coupled with relatively small nonfissionable regions. In addition, the data management requirements associated with accelerating the flux vector are at least an order of magnitude greater than those associated with the fission source for the 10 to 30 energy groups that are typically for fast reactor calculations. Both of these factors tend to favor the use of an outer iteration procedure based on fission source vector acceleration.

In the method reported here, approximations to λ_1 and $\vec{\Psi}_1$, the fundamental eigenvalue and eigenvector of Q, are obtained by the well-known power iteration method. It is assumed that the eigenvalue spectrum of Q satisfies $\lambda_1 > \lambda_2 \geq \lambda_3 \geq \cdots \geq \lambda_N$ and that $\vec{\Psi}_i$ is the eigenvector associated with λ_i. The power method proceeds as

$$\vec{\Psi}^{(n)} = \frac{1}{\lambda^{(n-1)}} Q\vec{\Psi}^{(n-1)} \qquad (16a)$$

and

$$\lambda^{(n)} = \lambda^{(n-1)} \frac{\|\vec{\Psi}^{(n)}\|_1}{\|\vec{\Psi}^{(n-1)}\|_1} , \qquad (16b)$$

where n is the outer iteration index and $\|\cdot\|_1$ denotes the L_1 norm. The actual computation of the product $Q\vec{\Psi}^{(n-1)}$ in e.g., Eq. (16a) ivolves another level of iteration, and is discussed later in this section.

Because the largest (in modulus) eigenvalue of Q is real and simple, the power method is guaranteed to converge for any arbitrary non-negative initial vector $\vec{\Psi}^{(0)}$ to λ_1, and $c\vec{\Psi}_1$, where c is some positive constant. If it is assumed that the eigenvalue esimates $\lambda^{(n)}$ are sufficiently well converged to λ_1 and that $\vec{\Psi}^{(0)}$ can be expanded in terms of the $\vec{\Psi}_i$, the eigenvectors of Q, then the rate at which $\vec{\Psi}^{(n)}$ converges to $\vec{\Psi}_1$ depends on the separation of λ_1 from the other eigenvalues of Q (Ref. 9). This convergence rate depends on the dominance ration, $\bar{\sigma}$, given by

$$\bar{\sigma} = \max_{i \neq 1} \frac{|\lambda_i|}{\lambda_1} , \qquad (17)$$

with the convergence rate ultimately being controlled by $(\bar{\sigma})^n$.

Dominance ratios for recent large fast reactor designs are typically on the order of .95 or larger, implying relatively slow convergence of the iterative process given by Eq. (16). In addition, typical fast reactor multigroup representations are characterized by nearly full downscattering matrices. The group-by-group calculation of the scattering source required for each outer iteration becomes a costly input/output-bound calculation when such representations are used in large multidimensional calculations. Both of these factors motivate the use of an efficient outer iteration acceleration technique in fast reactor diffusion theory calculations.

The Chebyshev semi-iterative method is utilized to accelerate the outer fission source iteration given by Eq. (16). Its application is based on the assumptions that the eigenvalues of Q are real and non-negative and are ordered as $\lambda_1 > \lambda_2 \geq \lambda_3 \geq \ldots \geq \lambda_N \geq 0$ and that the eigenvectors Ψ_i of Q form a basis for the N-dimensional vector space. The basic power iteration is accelerated by choosing a linear combination of the eigenvector iterates $\Psi^{(n)}$ such that

$$\tilde{\vec{\Psi}}^{(n^*+p)} = \sum_{j=0}^{p} a_{jp} \vec{\Psi}^{(n^*+j)} , \qquad (18)$$

where n^* is the outer index where this acceleration begins and p successive fission source iterates are employed. The objective is to choose the coefficients such that $\tilde{\vec{\Psi}}^{(n^*+p)}$ approximates $\vec{\Psi}_1$ more closely than does $\vec{\Psi}^{(n^*+p)}$.

Ref. 4 outlines the derivation that leads to the accelerated iterative procedure for $p \geq 1$:

$$\vec{\Psi}^{(n^*+p)} = \frac{1}{\lambda^{(n^*+p-1)}} Q\tilde{\vec{\Psi}}^{(n^*+p-1)} ,$$

$$\tilde{\vec{\Psi}}^{(n^*+p)} = \tilde{\vec{\Psi}}^{(n^*+p-1)} + \alpha_p[\vec{\Psi}^{(n^*+p)} - \tilde{\vec{\Psi}}^{(n^*+p-1)}] + \beta_p[\tilde{\vec{\Psi}}^{(n^*+p-1)} - \tilde{\vec{\Psi}}^{(n^*+p-2)}], \qquad (19)$$

where

$$\alpha_1 = \frac{2}{2 - \bar{\sigma}} , \quad \beta_1 = 0 ,$$

$$\alpha_p = \frac{4}{\bar{\sigma}} \frac{\cosh[(p-1)\gamma]}{\cosh[p\gamma]} ,$$

$$\beta_p = 1 - \frac{\bar{\sigma}}{2}\alpha_p - 1 . \qquad (20)$$

To apply the iteration schemes given by Eqs. (16) and (19), the dominance ratio $\bar{\sigma}$ must be obtained and a suitable convergence criterion must be applied to measure convergence; the theoretical aspects of these estimates are discussed in Ref. 4.

The inner iterations are required in carrying out the operation $Q\vec{\Psi}^{(n-1)}$ on the right side of Eqs. (16a) and (19a). From Eqs. (12) and (13), $Q\vec{\Psi}^{(n-1)}$ can be written as

$$Q\vec{\Psi}^{(n-1)} = \sum_{g=1}^{G} F_g L_g \vec{\Psi}^{(n-1)}$$

$$= \lambda^{(n-1)} \sum_{g=1}^{G} F_g \vec{\phi}_g^{(n)} , \qquad (21)$$

where

$$\vec{\phi}_g^{(n)} \equiv \frac{1}{\lambda^{(n-1)}} L_g \vec{\Psi}^{(n-1)} \qquad (22)$$

Given the $\vec{\phi}_g^{(n)}$, $Q\vec{\Psi}^{(n-1)}$ and hence $\vec{\Psi}^{(n)}$ can be easily obtained. The definition of L_g, Eq. (12), defines a series of linear equations:

$$A_g \vec{\phi}_g^{(n)} = \vec{b}_g^{(n)}, \; g = 1, 2, \ldots, G, \qquad (23)$$

which can be solved for the group flux vectors $\vec{\phi}_g^{(n)}$. The source $\vec{b}_g^{(n)}$ is given by

$$\vec{b}_g^{(n)} = \sum_{g'<g} T_{gg'} \vec{\phi}_{g'}^{(n)} + \frac{1}{\lambda^{(n-1)}} \chi_g \vec{\Psi}^{(n-1)} \qquad (24)$$

For multidimensional problems, the direct inversion of A_g matrices in Eq. (23) is not practical. The iterative inversion of A_g for each group comprises the inner iterations.

Because of its sound theoretical basis and computational simplicity (see Sec. IV), the line successive overrelaxation method has been chosen for the solution strategy reported here. The matrix A in Eq. (23) (dropping the group subscript) is split as[14]

$$A = D - E - F , \qquad (25)$$

where D contains the diagonal of A_g plus those off-diagonal coefficients that represent coupling between cell fluxes in each row, E contains those blocks of A that lie below the diagonal blocks placed in D, and F contains those blocks that lie above the blocks in D. The line successive overrelaxation procedure is then given by

$$\vec{\phi}_g^{(m+1)} = L_\omega \vec{\phi}_g^{(m)} + \vec{k}_g , \qquad (26)$$

where

$$L_\omega = (D - \omega E)^{-1}[\omega F + (1 - \omega)D] \qquad (27)$$

and

$$\vec{k}_g = (D - \omega E)^{-1} \omega \vec{b}_g . \qquad (28)$$

The matrix L_ω is the line successive overrelaxation iteration matrix and ω is the overrelaxation factor; both are group dependent. Because A (for each group) is an irreducible consistently ordered two-cyclic Stieltjes matrix for the finite differencing schemes used here, the iteration procedure given by Eq. (26) is convergent for $1 \leq \omega \leq 2$ (Ref. 8). Furthermore there is an optimum value of ω, say ω_b, for which the convergence is the most rapid. This group-dependent value of ω_b is given by[14]

$$\omega_b = \frac{2}{1 + [1 - \rho(L_1)]^{1/2}} , \qquad (29)$$

where $\rho(L_1)$ is the spectral radius of L_1, the associated Gauss-Seidel iteration matrix, which can be obtained from Eq. (27) by setting $\omega = 1$.

Following the procedure outlined in Ref. 15, the value of ω_b can be determined to arbitrary accuracy because the A matrix for each group has the properties listed above. For such matrices, if $x^{(0)} > 0$ and if

$$\vec{x}^{(m)} \equiv L_1 \vec{x}^{(m-1)} \qquad (30)$$

and

$$\delta^{(m)} \equiv \frac{[\vec{x}^{(m)}, \vec{x}^{(m)}]}{[\vec{x}^{(m)}, \vec{x}^{(m-1)}]} , \qquad (30)$$

then

$$\lim_{m \to \infty} \delta^{(m)} = \rho(L_1) . \qquad (31)$$

Furthermore, if $x_i^{(m-1)} \neq 0$ and if

$$\bar{\delta}^{(m)} \equiv \max_i \frac{x_i^{(m)}}{x_i^{(m-1)}} ; \; \underline{\delta}^{(m)} \equiv \min_i \frac{x_i^{(m)}}{x_i^{(m-1)}} , \qquad (32)$$

then

$$\bar{\delta}^{(m)} \geq \rho(L_1) \geq \underline{\delta}^{(m)},$$

$$\bar{\delta}^{(m)} \geq \delta^{(m)} \geq \underline{\delta}^{(m)} ,$$

and

$$\lim_{m\to\infty} \bar{\delta}^{(m)} = \lim_{m\to\infty} \underline{\delta}^{(m)} = \rho(L_1) \ . \qquad (33)$$

The spectral radius (L_1) can be computed by carrying out the iteration given by Eq. (30a), computing $\delta^{(m)}$, $\bar{\delta}^{(m)}$, and $\underline{\delta}^{(m)}$, and observing their convergence to one another. The computational details involved in implementing this procedure for computing ω_b are discussed in Sec. IV.

IV. ITERATION METHODS: COMPUTATIONAL CONSIDERATIONS

In Sec. III, the theory underlying the iteration methods for the solution strategy here has been presented. In this section, the computational considerations that determined the details of their implementation in the DIF3D code are discussed. The inner iteration procedures are presented here, preceded by a summary of the outer iteration procedures.

The obvious ultimate goal of the outer iteration procedure is to be able to apply the Chebyshev acceleration procedure given in Eqs. (19) with accurate estimates of both λ_1 and σ. However, since neither λ_1 nor σ are known when the outer iterations are commenced, a "boot-strap" process is required. As reported in Refs. 12 and 13, it has been found advantageous to perform a limited number of power iterations, Eq. (16), initially to provide a resonable estimate of λ_1 and an initial estimate of σ, which is generally quite low. A series of low-order extrapolation cycles is then utilized, during which the higher overtones are rapidly damped out and more accurate estimates of σ are obtained. Only when all but the first overtone mode are essentially damped out are high-order cycles based on accurate estimates of σ utilized. The precise algorithm is described in terms of four basic parts in Ref. 4.

Computational considerations arise concerning three aspects of the inner iterations. These are the computation of the optimum overrelaxation factor ω_b for each group, the determination of the number of inner iterations that should be carried out for a given group at a particular outer iteration, and the actual procedure used to solve the tridiagonal matrix equations that characterize the line successive overrelaxation method.

It has been shown in Sec. III that the optimum overrelaxation factor for a given group can be computed if the spectral radius of the line Gauss-Seidel matrix, $\rho(L_1)$, is known. The procedure outlined in Eqs. (30), (31), and (32) provides a rigorous method for determining $\rho(L_1)$. With the coding to carry out the inner iterations using the line successive overrelaxation method already in place, the implementation of this procedure is trivial, since L_1 is equal to L_ω, with ω set to unity. The vector k_g in e.g., Eq. (26) also has to be set to the null vector.

To ensure that the actual outer and inner iterations are as efficient as possible, this computation of the overrelaxation factors is done prior to commencing the first outer iteration. Starting with an arbitrary non-negative initial guess $x^{(0)}$, the iteration in Eq. (30a) is carried out for m = 1 to 10. Following each iteration for m > 10, the quantities $\delta^{(m)}$, $\bar{\delta}^{(m)}$, and $\underline{\delta}^{(m)}$ are computed. The related quantities $\omega^{(m)}$, $\bar{\omega}^{(m)}$, and $\underline{\omega}^{(m)}$, defined by[12]

$$\omega^{(m)} \equiv \frac{2}{1 + [1 - \delta^{(m)}]^{1/2}} ,$$

$$\bar{\omega}^{(m)} \equiv \frac{2}{1 + [1 - \bar{\delta}^{(m)}]^{1/2}} ,$$

and

$$\underline{\omega}^{(m)} \equiv \frac{2}{1 + [1 - \underline{\delta}^{(m)}]^{1/2}} , \qquad (34)$$

are also computed. The iterations for a given group are terminated when

$$\bar{\omega}^{(m)} - \underline{\omega}^{(m)} < \frac{2 - \omega^{(m)}}{5} \qquad (35)$$

and ω_b for that group is set equal to $\omega^{(m)}$. The test given by Eq. (35) forces tighter convergence as $\rho(L_\omega)$ increases. The amount of central processor unit (CPU) time required to precompute the ω_b is typically on the order of one to two outer iterations.

The theory presented in Ref. 4 on the Chebyshev acceleration method implicitly assumes that the matrix equation for each group, Eq. (23), is solved exactly during each outer iteration. For multidimensional problems, this is not the case. It has been shown[13] that the effect of solving Eq. (23) iteratively to less than infinite precision for each group is to modify somewhat the system of equations being solved. Although both systems share the same fundamental eigenvalue and eigenvector, the dominance ratio of the modified system is larger than the original system, Eq. (14). Some of the eigenvalues of the modified system may be negative or complex, which would slow convergence of the outer iterations.

The most practical solution to this problem is to do a sufficient number of inner iterations for each group during each outer iteration, so that the effect on the dominance ratio is not appreciable, yet no more than this. It has been determined experimentally for a range of typical fast reactor problems that this can be achieved most economically by doing a fixed number of iterations, m_g, for each group during each of the outer iterations. This eliminates the need for any convergence checking during the inner iterations and thus eliminates the costly divides that would have to be done to determine relative convergence on a component-by-component basis.

This number, m_g, is determined for each group by requiring that the norm of the continued product of the iteration matrices for that group during each outer iteration be less than some desired error reduction factor. This ensures that the norm of any of the components of the error vector is greater than or equal to this error reduction factor during each outer iteration. For a variant of the line-successive overrelaxation method of Eq. (26), where a single Gauss-Seidel iteration precedes (m - 1) successive overrelaxation iterations, the norm of the continued product of the iteration matrices is given by[16]

$$\|L_{\omega_b}^{m-1} L_1\|_2 = (t_{2m-1}^2 + t_{2m}^2)^{1/2}, \quad m \geqslant 1, \qquad (36)$$

where

$$t_m = (\omega_b - 1)^{(m-1)1/2} [\rho(L_1)]^{1/2}$$

$$\times (1 + (m - 1)\{1 - [\rho(L_1)]^{1/2}\}).$$

The single Gauss-Seidel iteration is applied because the norm in Eq. (36) is then strictly decreasing for $m \geqslant 1$. Letting ε_{in} be the desired error reduction factor and given ω_b and $\rho(L_1)$ for a group from the optimum overrelaxation factor calculation just described, Eq. (36) is solved to determine that value of m such that

$$\|L_{\omega_b}^{m-1} L_1\|_2 \leqslant \varepsilon_{in} . \qquad (37)$$

The value of m so obtained is the fixed number of inner iterations, m_g, that are done for group g for every outer iteration.

Experience has shown that choosing $\varepsilon_{in} \leqslant 0.04$ will result in no adverse impact on the outer iteration convergence rate for typical fast reactor problems. For problems with dominance ratios >0.85 (large reactors), a value of ε_{in} as small as 0.01 is sometimes necessary. It is quite obvious when a value of ε_{in} that is too large for the problem at hand has been chosen. The dominance ratio estimates being obtained from the outer iterations grow too large, and oscillatory behavior of the acceleration cycles generally results.

A large percentage of the total CPU time required to solve large problems with this solution method is spent in the inner iterations. In implementing the algorithms used to carry out these iterations, it is essential that the full capabilities of the present-day large-scale scientific computers are utilized. The impact of vector processing capabilities is considered in Sec. VII. A feature shared by some of these computers is the high-speed instruction stack, from which significant gains in execution speed can be obtained when repetitive instruction sequences can be contained in this stack. Multiple functional units and instruction segmentation permit parallel execution of several arithmetic operations, loop indexing, and the storing and fetching of data.

The requirements for utilizing these features efficiently include the following:

1. compact coding for loops
2. no conditional branching performed within the loop
3. avoiding divisions whenever possible

The one-line successive overrelaxation method was chosen in part because it is simple and can be coded compactly. Performing a fixed number of inner iterations for each group eliminates the need for the divides and conditional branching that usually accompanies convergence checking. Finally, by utilizing the procedure outlined below, it is possible to eliminate all divides and conditional branching from the innermost loops of the inner iteration algorithm and reduce those loops to a few lines of machine language coding that easily fit within the instruction stack on an IBM 370/195 or the CDC 7600.

For a particular line of fluxes that are computed simultaneously during each inner iteration, the equations that must be solved are of the form

$$\vec{s}^{(m+1)}_{jk} = \vec{b}_{jk} + C_{j-1k}\vec{\phi}^{(m+1)}_{j-1k} + C_{jk}\vec{\phi}^{(m)}_{j+1k} + B_{jk-1}\vec{\phi}^{(m+1)}_{jk-1} + B_{jk}\vec{\phi}^{(m)}_{jk+1} , \quad (38)$$

$$A_{jk}\hat{\vec{\phi}}^{(m+1)}_{jk} = \vec{s}^{(m+1)}_{jk} , \quad (39)$$

$$\vec{\phi}^{(m+1)}_{jk} = \omega_b[\hat{\vec{\phi}}^{(m+1)}_{jk} - \vec{\phi}^{(m)}_{jk}] + \vec{\phi}^{(m)}_{jk}, \quad (40)$$

where j and k are the row and plane indices of this line and A_{jk} is a tridiagonal Stieltjes matrix given by

$$A_{jk} = \begin{bmatrix} e_1 & -d_2 & & & \\ -d_2 & e_2 & -d_3 & & \\ \cdot & \cdot & \cdot & & \\ & \cdot & \cdot & \cdot & \\ & & \cdot & \cdot & \cdot \\ & & -d_{I-1} & e_{I-1} & -d_I \\ & & & -d_I & e_I \end{bmatrix}, \quad (41)$$

where I is the number of mesh cells in the line. The diagonal matrices C_{jk} and B_{jk} are the off-diagonal coefficients that represent coupling between cell fluxes in neighboring rows.

The solution of Eq. (39) utilizes a forward-elimination backward-substituion algorithm similar to Gaussian elimination. The forward elimination on the matrices A_{jk} is performed only once, prior to the beginning of the outer iterations, in such a fashion as to in computing the $\vec{\phi}_{jk}$. The backward sweep and overrelaxation are then combined into a single loop to save memory fetches and stores.

The forward elimination on A_{jk} is given by

$$\gamma_1 = \frac{1}{e_1} , \quad (42a)$$

$$\theta_i = d_{i+1}\gamma_i , \qquad i = 1, 2, \ldots, I - 1, \quad (42b)$$

$$\gamma_i = \frac{1}{e_i - d_i\theta_{i-1}} , \qquad i = 2, 3, \ldots, I. \quad (42c)$$

The γ_i values are saved for subsequent use in the inner iterations by storing over the e_i values, which are no longer needed. Given $\vec{s}^{(m+1)}_{jk}$ for one inner iteration, the forward sweep on it is given by

$$\eta_1 = \gamma_1 s_1, \quad (43a)$$

$$\eta_i = \gamma_i(s_i + d_i\eta_{i-1}), \qquad i = 2, 3, \ldots, I, \quad (43b)$$

where s_i is the i'th component of s_{jk}. A second loop then performs the remainder of the work on line j, k according to

$$\mu_I = \eta_I,$$

$$\phi^{(m+1)}_I = \phi^{(m)}_I + \omega_b [\mu_I - \phi^{(m)}_I], \quad (44a)$$

$$\mu_i = \eta_i + d_{i+1}\gamma_i\mu_{i+1}, \qquad i = I - 1, \ldots, 2,1 \quad (44b)$$

$$\phi^{(m+1)}_i = \phi^{(m)}_i + \omega_b [\mu_i - \phi^{(m)}_i], \qquad i = I - 1, \ldots, 2, 1 \quad (44c)$$

This procedure permits extremely efficient use of the scalar arithmetic capabilities of high-speed computers.

V. DATA MANAGEMENT CONSIDERATIONS

Strong consideration must be given to the data management implications of any solution method that is contemplated for use in a code capable of treating problems where the number of space-energy unknowns can exceed 10^6. From the previous sections, it is obvious that such considerations have influenced the form of the solution method presented in this paper. These considerations are summarized in this section.

The primary goal of the solution strategy described here is to reduce the number of outer iterations to a minimum, even at the expense of investing relatively greater effort in the inner iterations performed during each outer iteration. By minimizing the number of outer iterations, the number of scattering source calculations (one per group per outer iteration) is kept at a minimum. These scattering source calculations necessitate the transfer of large amounts of data from peripheral storage to core memory for problems utilizing ten or more energy groups, yet there is little arithmetic to be done while these data transfers are taking place. As a result, CPU utilization can be quite low during the scattering source calculations, even if efficient asynchronous data transfer methods are utilized.

Data management considerations also led to the decision to apply the Chebyshev polynomial acceleration technique to the fission source vector $\vec{\Psi}$ rather than the flux vector $\vec{\Phi}$. Three complete fission source or flux vectors, depending on which are to be accelerated, have to be stored on peripheral storage devices and transferred to core to carry out the acceleration procedure for each outer iteration. Again, there is little arithmetic associated with this acceleration method, so that CPU utilization can again be low if large amounts of data have to be transferred. Since the fission source vectors are only (1/G) as long as the flux vectors, a significant reduction in data transfer requirements is achieved by accelerating the $\vec{\Psi}$ vector.

VI. CONVERSION OF DIF3D TO THE CRAY-1

The implementation of the entire DIF3D code (43000 cards) on the CRAY-1 at the National Center for Atmospheric Research (NCAR) was accomplished with relative ease. DIF3D is designed with portability in mind, and to this end a simple preprocessor activates or deactivates coding appropriate for the intended host computer.

Among the available options coding for a longword single level memory hierarchy machine was selected for implementaion on the CRAY-1. Several changes to this source code included ENTRY point syntax and removal of overlay calls. The CDC FTN4 ENTRY point syntax invoked by the longword coding was modified to the IBM compatible syntax supported by the CFT compiler. One million words is ample storage for the problems likely to be considered presently, so that the overlay option has not yet been invoked.

Miscellaneous compiler and machine dependent items requiring change included several utility function (END-OF-FILE status and word address) names and a machine dependent routine that invokes the MEMORY macro to dynamically increase user memory. Dynamic allocation of arrays is performed in a storage container adjacent to the last word of user code.

DIF3D is organized to take advantage of asynchronous random access I/O on machines which support these features. All unformatted binary I/O is isolated in a few standard I/O subroutines[17] to facilitate local adaptions where standard FORTRAN performance is unacceptable. Only synchronous I/O has been attempted with DIF3D on the CRAY-1.

The data management strategy in DIF3D is dynamically selected based on the available memory for the problem at hand. A variety of regimes are permitted; two are of interest here. First, large two- or three-dimensional problems require that only one of the typically 20 to 30 energy groups of data be core contained during the inner iterations on a group flux. Second, depending on the user supplied container size, three-dimensional problems may be forced into a concurrent inner iteration strategy that requires data for only a fraction of the total number of mesh planes in a group be core contained. Consequently an unlimited number of mesh planes with a plane size of nearly 33000 mesh cells is permitted on the CRAY-1 with one million words of memory. The corresponding limit for two-dimensional problems is nearly 90000 mesh cells in a plane.

VII. VECTORIZATION of SLOR

The recursive aspect of the solution algorithm for solving the tridiagonal system of equations, Eq. (39) can be avoided by adopting an odd/even line ordering[18,19]. When solved simultaneously, the mutually independent blocks of lines yield a vectorization with vector length equal to half the number of lines under consideration. The recursions now become recursions on vectors of length equal to the number of systems being solved.

For computational convenience an odd/even ordering on a plane was implemented in DIF3D. The theory of Sec. III is readily shown[14] to apply to this reordered system of equations so that the computational equations of Sec. IV remain unchanged except for their order of application to the mesh lines. The reordered algorithm for mesh plane k successively solves Eqs. (39), (43) and (44) simultaneously for the odd numbered lines on plane k. The process is then repeated for the even numbered lines on plane k. The results in Sec. VIII show that computation speeds between 20 and 30 megaflops are achieved.

VIII. PERFORMANCE RESULTS

The fact that 75% of DIF3D execution time on the IBM 370/195 computer is accounted for by the scalar SLOR algorithm Eqs. (38), (39) and (40), led to the creation of a DIF3D kernel that largely consists of two small subroutines (SORINV and ROWSRC) at the heart of the optimized algorithm.

To assess relative computing speeds of selected large scale scientific computers for this algorithm, a benchmark model with a 50×50 mesh using 25 inner iterations was repeated ten times for a total of 625000 mesh cell iterations with 13 floating point operations per mesh cell iteration. Several optimized SLOR algorithm options were compared: (1) Vectorized FORTRAN coding (odd/even ordering); (2) Scalar FORTRAN coding; (3) Scalar FORTRAN coding with an assembler optimized SORINV. The relative computing speeds are illustrated in Table I.

Analysis of the code generated by the CFT compiler indicates that CAL assembler optimization of the vectorized SORINV subroutine could obtain execution rates of 33 megaflops compared to the 22 megaflops presently achieved.

To assess the performance of the entire DIF3D code, a sample two-dimensional two group problem with a space mesh of 170×170 (57800 unknowns) was chosen. Problem results are displayed in Table II. A three-dimensional two group problem with a space mesh of 34×34×75 (173400 unknowns) was also solved. Problem results are displayed in Table III. The relatively short vector length in this realistic three dimensional problem significantly reduces the megaflop rate. The vector length could be doubled in this problem by reorienting the spatial dimensions; thereby attaining a corresponding increase in speed.

Although the spectral radii of the inner iteration matrices were comparable for both scalar and vector SLOR algorithms, the effect on the outer iterations was noticably different. In vector mode, the two-dimensional problem required about 19% fewer outers and the three-dimensional problem required about 10% more outers than their respective scalar counterparts. It is hypothesized that an alternate strategy utilizing an odd/even ordering of lines in three-dimensions, not just on a plane, would improve the Chebyshev acceleration of the outers and at the same time yield significant performance advantage due to the increased vector lengths.

In conclusion, these preliminary experiences with DIF3D on the CRAY-1 indicate that a code designed primarily for optimal scalar performance can be efficiently implemented on the CRAY-1 in both scalar and vector mode, achieving 10 megaflops in the former and 20-30 megaflops in the latter mode for realistic size problems. More extensive algorithm modifications and careful optimization has the potential for reaping improved performance at the possible expense of portability.

Table I. Kernel problem relative execution rates in units of 4.2 megaflops.

METHOD/MACHINE	CRAY-1	IBM 370/195	CDC 7600
Vector Fortran	5.4	1.05	1.09
Scalar Fortran	1.7	0.98	1.13
Scalar Fortran with Assembler SORINV	3.3	1.0	1.4

Table II. 2-D sample problem execution rates in units of 3.9 megaflops.

METHOD/MACHINE	CRAY-1	IBM 370/195	CDC 7600
Vector Fortran	4.4 (5.3)[a]	0.83	-
Scalar Fortran	1.6	-	0.97
Scalar Fortran with Assembler SORINV	2.7	1	-

[a](5.3) denotes the actual speed advantage due to 19% fewer outer iteration.

Table III. 3-D sample problem execution rates in units of megaflops.

METHOD/MACHINE	CRAY-1	IBM 370/195
Vector Fortran	11.	-
Scalar Fortran		2.98

REFERENCES

[1]R. S. VARGA, *Matrix Iterative Analysis*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey (1962).

[2]E. L. WACHPRESS, *Iterative Solution of Elliptic Systems and Applications to the Neutron Diffusion Equations of Reactor Physics*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey (1966).

[3]R. S. VARGA, *IRE Trans•Nucl• Soc•*, NS-4, 52 (1957).

[4]D. R. FERGUSON and K. L. DERSTINE, "Optimized Iteration Strategies and Data Management Considerations for Fast Reactor Finite Difference Diffusion Theory Codes," *Nucl• Sci• Eng•* 64, p. 593 (1977).

[5]R. W. HARDIE and W. W. LITTLE, Jr., "3DB, A Three-Dimensional Diffusion Theory Burnup Code," BNWL-1264, Battelle-Pacific Northwest Laboratories (1970).

[6]D. R. VONDY, T. B. FOWLER, and G. W. CUNNINGHAM, "VENTURE: A Code Block for Solving Multigroup Neutronics Problems Applying the Finite-Difference Diffusion-Theory Approximation to Neutron Transport," ORNL-5062, Oak Ridge National Laboratory (1975).

[7]T. A. DALY et al., "The ARC System Two-Dimensional Diffusion Theory Capability, DARC2D," ANL-7716, Argonne National Laboratory (1972).

[8]R. S. VARGA, *Matrix Iterative Analysis*, Chap. 6, Prentice-Hall, Inc., Englewood Cliffs, New Jersey (1962).

[9]R. S. VARGA, *Proc. Symp. Appl. Math.*, 11, 164, American Mathematical Society, Providence, Rhode Island (1961).

[10]G. BIRKHOFF and R. S. VARGA, *J. Soc. Ind. Appl. Math.*, 6, 354 (1958).

[11]R. FROEHLICH, "A Theoretical Foundation for Coarse Mesh Variational Techniques," *Proc. Int. Conf. Research Reactor Vitalization and Reactor Mathematics*, Mexico, D.F., 1, 219 (May 1967).

[12]L. A. HAGEMAN, "Numerical Methods and Techniques Used in the Two-Dimensional Neutron Diffusion Program PDQ-5," WAPD-TM-364, Bettis Atomic Power Laboratory (1963).

[13]L. A. HAGEMAN and C. J. PFEIFER, "The Utilization of the Neutron Diffusion Program PDQ-5," WAPD-TM-395, Bettis Atomic Power Laboratory (1965).

[14]R. S. VARGA, *Matrix Iterative Analysis*, Chap. 4, Prentice-Hall, Inc., Englewood Cliffs, New Jersey (1962).

[15]R. S. VARGA, *Matrix Iterative Analysis*, Chap. 9, Prentice-Hall, Inc., Englewood Cliffs, New Jersey (1962).

[16]R. S. VARGA, *Matrix Iterative Analysis*, Chap. 5, Prentice-Hall, Inc., Englewood Cliffs, New Jersey (1962),

[17]R. DOUGLAS O'DELL, "Standard Interface Files and Procedures for Reactor Physics Codes, Version IV," LA-6941-MS, Los Alamos Scientific Laboratory (1977).

[18]B. BUZBEE, Los Alamos Scientific Laboratory, personal communication.

[19]D. L. BOLEY, "Vectorization of Block Relaxation Techniques Some Numerical Experiments," Proceedings of the 1978 LASL Workshop on Vector and Parallel Processors, LA-7491-C, p. 166 (1978).

NOMENCLATURE

Any symbol not defined in the Nomenclature is defined locally in the text.

Scalars

g	=	index number of the energy group
G	=	total number of energy groups
ϕ_g	=	scalar neutron flux [n/cm^{2}s)] in enercy group g
D_g	=	diffusion coefficient for neutrons in group g (cm)
Σ_g^r	=	macroscopic removal cross section for group g
Σ_g^r	=	$\Sigma^a + \sum_{g'=g} \Sigma_{g'g}^s$
$\Sigma_{gg'}^s$	=	macroscopic scattering cross section from group g' to group g
Σ_g^a	=	macroscopic absorption cross section in group g
χ_g	=	fission spectrum in group g
$\nu\Sigma_g^f$	=	average number of neutrons per fission times the macroscopic fission cross section in group g
λ	=	k_{eff} of reactor
N	=	total number of finite difference mesh cells
$\lambda^{(n)}$	=	estimate for λ_1 at outer iteration n
n	=	outer iteration index

Matrices and Vectors

$\vec{\phi}_g$	=	scalar neutron flux vectors, group g
D_g	=	three-, five-, or seven-stripe diffusion matrix for group g
Σ_g	=	diagonal removal matrix for group g
$T_{gg'}$	=	diagonal scattering matrix, group g' to group g
χ_g	=	diagonal fission spectrum matrix, group g
F_g	=	diagonal production matrix, group g
A_g	=	leakage plus removal matrix operator, group g
$\vec{\psi}$	=	fission source vector
Q	=	outer (fission source) iteration matrix
$\vec{\psi}_i$	=	eigenvector of Q corresponding to η_i
$\vec{\psi}^{(n)}$	=	estimate of $\vec{\psi}_1$ at outer iteration n
$\vec{\phi}^{(n)}$	=	estimate of $\vec{\phi}_g$ at iteration n

ACKNOWLEDGMENTS

The author is indebted to Larry Rudsinski for initiating the project which provided access to the CRAY-1 at NCAR and to Floyd Dunn for providing optimized CAL and COMPASS versions of SORINV.

This work was performed under the auspices of the U.S. Department of Energy.

TURBULENCE/ HYDRODYNAMICS

PAGES 129 to 130

WERE INTENTIONALLY

LEFT BLANK

CALCULATIONS OF WATER WAVES AND VORTEX ARRAYS BY NUMERICAL SOLUTION OF INTEGRO-DIFFERENTIAL EQUATIONS

P.G. Saffman, B. Chen, R. Szeto
Department of Applied Mathematics 101-50
California Institute of Technology
Pasadena, California 91125

ABSTRACT

Steady gravity-capillary waves of permanent form on deep water and the shapes of vortices in a linear array are calculated numerically. For the waves, it is shown that finite amplitude waves can bifurcate and new types of steady waves exist in which crests and troughs may be of unequal height. Gravity-capillary waves of maximum height are calculated. For the vortices, the shapes are found for various values of the size/separation. It is found that there exists a maximum size for given separation and properties of the array are determined. The non-linear equations were solved by Newton's method using the CDC STAR-100 computer at the CDC Service Center in Minneapolis.

INTRODUCTION

Steady solutions of the incompressible Euler equations (inviscid Navier-Stokes equations) are of considerable interest for many fluid mechanical problems because of the insight they can provide into some of the physical processes that govern the behavior of fluids of small viscosity and constant density. If the flow is everywhere irrotational, i.e. free of vorticity and circulation, then the calculation of the flow field reduces to finding solutions of Laplaces equation with suitably given boundary conditions. This class of problems is of limited interest. But if the flow contains vorticity, either continuously distributed or concentrated into sheets or both, then the mathematical problems become more challenging and the physical relevance may be significant. A large part of the mathematical difficulty, which is directly connected with the physics, arises from the fact that the position and strength of the vorticity are in general unknown and are to be determined. Thus even in those parts of the flow where the motion is irrotational and governed by Laplaces equation, the shapes of the boundaries are unknown and one faces a free boundary value problem. The problems tend to be strongly nonlinear, and relatively little has been found so far using the classical methods of perturbation theory or the qualitative concepts of functional analysis (including catastrophe theory) apart from indications of possible trends of suggested classifications. This is not enough, because existence and uniqueness problems are now not academic niceties, but real physical questions, and there some exact analytical solutions available to show that existence and uniqueness cannot be taken for granted. The study of the bifurcation and limit point behavior associated with the lack of uniqueness, and the investigation of the stability of the flows to small disturbances, require actual numbers and quantitative details for a proper <u>qualitative</u> understanding to be obtained, especially if application to real flows is to be made. Numerical solution in an appropriate way of the equations seems to be the most powerful tool currently available for the discovery of new qualitative behavior, and the fact that it gives the numbers at the same time is an invaluable bonus.

We shall discuss here two examples to support this statement. The problems to be described are capillary-gravity waves of permanent form on deep water and the structure of an infinite array of uniform vortices. The first has been of mathematical interest since the work of Stokes in 1840 and applications to ocean

This research was supported by Control Data Corporation, the Department of Energy (EY-76-S-03-0767) and the U.S. Army Research office, Durham (DAAG 29-78-C-0011).

engineering and extraction of energy from the sea has led to a present high level of interest in water waves. The second is related to the recent discovery of organized structures in turbulent mixing layers and the idea that turbulent transport and mixing might be understandable in terms of the interaction of two-dimensional vortices of finite size. (A review of vortex interactions is provided by Saffman and Baker[1].) For both problems, we wished to use a method which would uncover qualitative properties, as well as provide the quantitative details, and moreover would work if the steady flow is unstable. Newton's method proved ideal. However, the calculations would not have been possible without the availability of time on a large, fast computer because of the relatively large number of variables. Our computations were performed on the Control Data Corporation STAR-100 Computer located at the CDC Service Center in Minneapolis, Minnesota. We are grateful to Control Data Corporation for making the maching available to us, and for giving us the opportunity to demonstrate how new insights can be obtained by the use of a sufficiently powerful computer.

WATER WAVES

We consider periodic, steady or permanent, one dimensional, inviscid irrotational, progressive water waves of finite amplitude on deep water. The crests are supposed parallel and straight, and the problem is to determine the wave profile, i.e. the shape of the free surface, and the speed of propagation as functions of the wave height or average slope. The mathematical problem is to find a solution of the Euler equations, which in this case reduce to Laplaces equation, such that the unknown free surface is a streamline relative to an observer moving with the wave on which the pressure is constant. Generalizations to water of finite depth, or interfacial waves between fluids of different density, or waves on a uniform shearing flow, are in principle straightforward but remain to be studied in detail. There are several mathematical formulations of the problem. We used one based on the concept that the free surface can be regarded as a vortex sheet between the water and the air. Chen and Saffman[2] showed that this leads to a complex singular integro-differential equation for the parametric equations $x = x(\sigma)$, $y = y(\sigma)$ of the free surface, which can be written

$$\left(1 - \frac{gL}{\pi c^2}\,\mathrm{Im}\, z + \frac{2\kappa}{c^2 R}\right)\frac{dz^*}{d\sigma} - 1 = -\frac{i}{2\pi}\,P\!\int_0^{2\pi}\cot\frac{z(\sigma)-z(\sigma_1)}{2}\,d\sigma_1 \qquad (1)$$

for $0 \le \sigma \le 2\pi$. Here, $z = x + iy$, g is the acceleration due to gravity, κ is the surface tension, L is the period of the motion, c is the unknown wave speed, P denotes Cauchy principal value, and R is the radius of curvature of the free surface

$$\frac{1}{R} = \frac{2\pi}{L}\,\mathrm{Im}\,\frac{d^2z}{d\sigma^2}\frac{dz^*}{d\sigma} \Big/ \left|\frac{dz}{d\sigma}\right|^3 . \qquad (2)$$

The actual physical coordinates are obtained by multiplying x and y by $L/2\pi$, and the origin is chosen to be in the mean level of the surface. A periodic solution (modulo 2π) is required so that

$$z(\sigma+2\pi) \equiv z(\sigma) + 2\pi . \qquad (3)$$

The trivial solution is $z = \sigma$, in which the surface is flat. This solution bifurcates with infinite degeneracy (the so-called primary bifurcation) into infinitesimal or linear progressive waves of complex amplitude a

$$z = \sigma + aie^{iN\sigma} + O(a^2) . \qquad (4)$$

$$\frac{2\pi c^2}{gL} = \frac{1}{N} + \frac{4\pi^2\kappa}{gL^2}N + O(a^2) , \qquad (5)$$

where N is an arbitrary positive integer which specifies the wavelength $\lambda = L/N$ (i.e. the distance between crests) or the number of waves in the window of length L.

Finite amplitude numerical solutions were calculated by solving a discretized form of Eq. (1). A uniform mesh in σ was introduced and equations for the values of z at the mesh points were obtained as follows. The derivatives in Eqs. (1) and (2) were replaced by a sixth order finite difference formula. To evaluate the integral, an integration mesh was introduced midway between the first mesh, the values of z were evaluated on the integration mesh by a sixth order interpolation formula, and the integral was then evaluated on the mesh by the trapezoidal rule. The set of non-linear transendental equations that results from satisfying Eq. (1) were solved by Newton's method, combined with Euler continuation in the wave height or an equivalent parameter to construct a branch of solutions. The linear solution given by Eqs. (4) and (5) was used to give the initial guess for small amplitude waves. Bifurcation and limit point behavior was treated for by monitoring the sign of the Jacobian in the Newton iteration. Keller's[3] method of pseudo arc length continuation was used to follow branches in the neighborhood of critical points.

The infinitesimal waves are all symmetrical about crests and troughs, and without loss of generality the argument of a can be fixed so that $\sigma = 0$ is a crest. It has been proved that waves of given height and wavelength are unique and symmetrical provided $4\pi^2\kappa/gL^2 \neq 1/M$, where M is an integer greater than one and the height is sufficiently small. In the first instance therefore, we supposed that the waves of finite amplitude were symmetrical about $\sigma = 0$, i.e.

$$z(\sigma) = -z^*(-\sigma) \quad . \qquad (6)$$

Each branch corresponding to a particular value of N is then well defined for small amplitude and continuation to finite amplitude was straightforward. One little trick was, however, useful. When the waves become steep, they tend to be cusped at the crests and resolution there is impaired. We then replaced σ for calculation on a particular branch by a new independent variable σ', defined by

$$\sigma = \sigma' - \frac{\alpha}{N}\sin N\sigma' \qquad (7)$$

and used a uniform mesh in σ'. By choosing α just less than 1, points are concentrated in the neighborhood of the crests.

In the calculations we used 40 and 80 mesh points. For the smaller value, it took about 1 sec. to compute and invert the Jacobian. Four iterations were usually sufficient to reduce the residuals to $O(10^{-10})$.

It may be asked why it is necessary to calculate on branches with different N, since surely solutions on the different branches are similar because a wave with surface tension κ, wavelength L and speed c is the same as one for which these quantities are κ/N^2, L/N and $c/N^{\frac{1}{2}}$. Thus the first branch $N = 1$ should give all the other branches. It turns out that this is only true when the waves are small. The branches with $N > 1$ bifurcate, and new branches exist at finite amplitude which cannot be found just by considering the continuation of the $N = 1$ branch to steep waves. We wish to emphasize that these secondary bifurcations were found by numerical solution and were completely unsuspected by analytical theoreticians, although in retrospect it becomes clear that they are associated with the special behavior when $4\pi^2\kappa/gL^2 = 1/M$ and can be analyzed for capillary-gravity waves by perturbation theory when this condition is approximately satisfied[2].

GRAVITY WAVES

Pure gravity waves (i.e. $\kappa = 0$) have been calculated in recent years by several authors and their properties along, in effect, the $N = 1$ branch have been studied from infinitesimal waves to the wave of greatest height $h/L = 0.141$ (h = vertical distance from trough to crest) for which the crests are cusped with a slope of 30° (see, e.g. Cokelet[4]). The results display some curious features. For instance, the wave speed and wave energy are not monotonic functions of the wave height and the maximum slope can exceed 30°. We calculated the $N = 1$ branch with our method, to check the approach and search for bifurcation. Our results agreed to at least 4 significiant figures with the most reliable of the others, and more significiantly no critical points were discovered.

Results for computation along the branches $N = 2$ and $N = 3$ are shown in Figures 1-4.

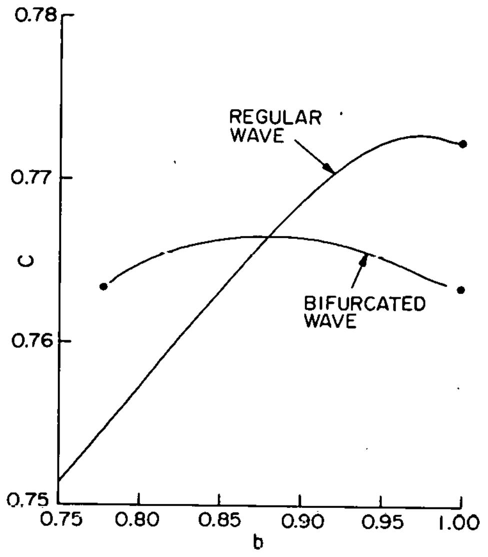


Figure 1. Wavespeed vs. height, N = 2.

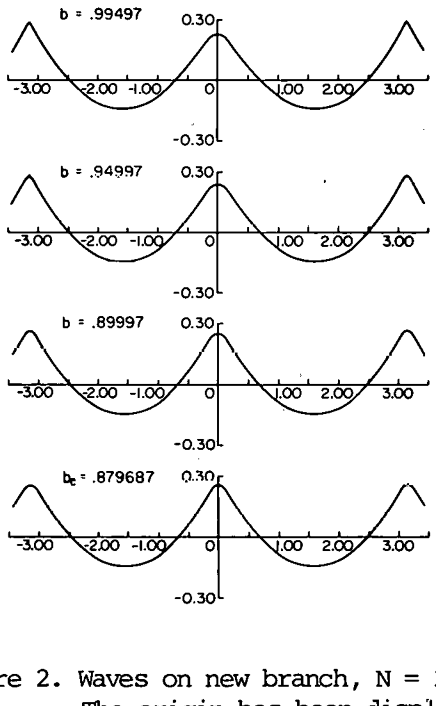


Figure 2. Waves on new branch, N = 2. The origin has been displaced to $\sigma = \pi$ and the mean water level.

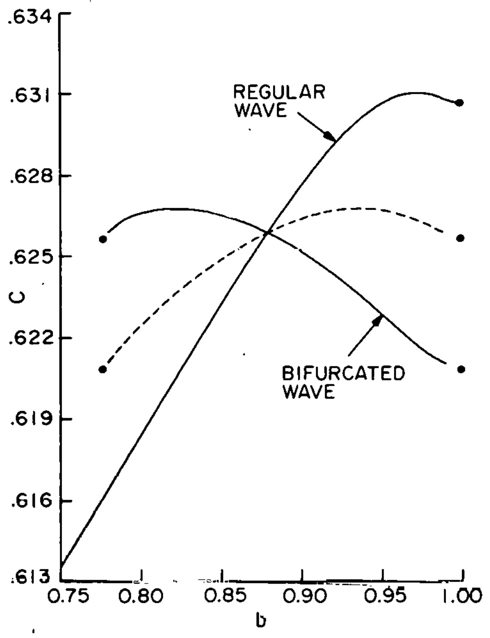


Figure 3. Wavespeed vs. height, N = 3.

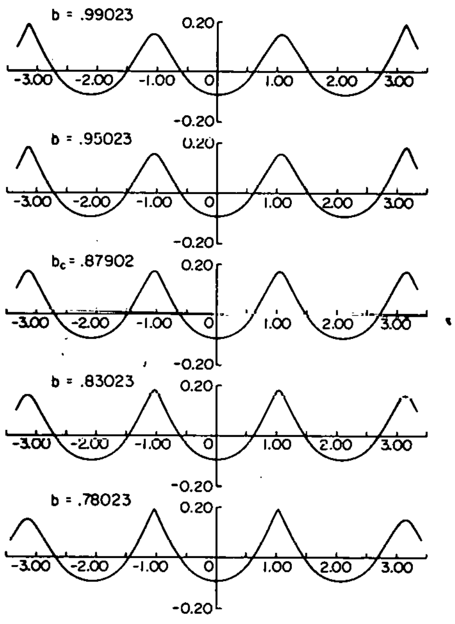


Figure 4. Waves on new branch, N = 3.

The wave speed has been made dimensionless by taking $g = 1$ and $L = 2\pi$. The parameter b is a dimensionless measure of the height at $\sigma = 0$; $b = 0$ is the flat surface and $b \to 1$ when the crest at $\sigma = 0$ peaks. Keller's[3] method enabled us to follow the new branches without trouble. The term regular wave refers to the branch that is the scaled

N = 1 branch. The crests of waves on the bifurcated branches are of unequal height. The decrease in b along half the new branch does not mean that the wave is getting flatter, but that the heighest crest is not at $\sigma = 0$. Actually for $N = 2$, the waves on each side of the new branch are identical, the difference is a horizontal displacement of $\frac{1}{2}L$.

The dotted line of Figure 3 shows the result of a calculation in which the possibility of bifurcation into unsymmetrical waves was studied. As is clear from Figure 4, the new waves for $N = 3$ are not symmetrical about all crests. The calculation picked up the waves of Figure 4 referred to an unsymmetrical crest. No completely unsymmetrical waves have been discovered, but it remains an open question whether they can exist. The stability of these waves to small disturbances and the bifurcation properties of branches with $N > 3$ remains to be investigated.

GRAVITY CAPILLARY WAVES

The richness of bifurcation behavior in this case is too great to be summarized succinctly and in fact has not yet been properly classified. In Figures 5-7, we show some examples of capillary-gravity waves found by following branches. The waves of greatest height for $\kappa \neq 0$ are limited by the surface touching itself and enclosing a bubble. Continuation in κ shows that the limit $\kappa = 0$ is singular and gravity waves cannot be obtained as the continuous limit of a capillary-gravity waves as $\kappa \to 0$. For further details, see Chen and Saffman[5].

VORTEX ARRAYS

As a second example, we consider the problem of calculating the equilibrium shapes of a linear array of equal uniform two-dimensional vortices of finite size. This problem has become of practical importance in recent years because of the discovery of 'big eddies' or coherent structures in the turbulent mixing layer (Roshko[6]) we suppose that each vortex has strength Γ and area A. The centers lie on a straight line, distance L apart. The vorticity inside each vortex is constant of value Γ/A.

Let $Z(s) = X(s) + iY(s)$ denote the surface of the vortex whose center is at the origin. Then it can be shown[7]

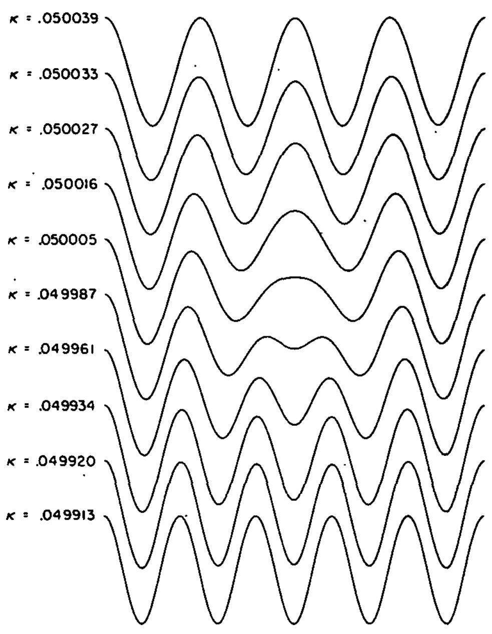


Figure 5. Continuous transition from $N = 4$ to $N = 5$.

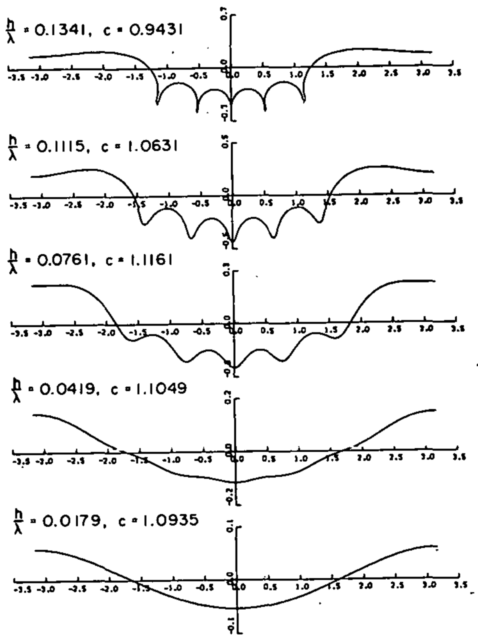


Figure 6. The branch $N = 1$ for $\kappa = .19$.

that $Z(s)$ is a solution of the equation

$$\operatorname{Im}\left\{\frac{dZ^*}{ds} \oint \log\left|\sin\pi(Z-Z')/L\right| dZ'\right\} = 0. \qquad (8)$$

For $A/L^2 \ll 1$, it can be shown that

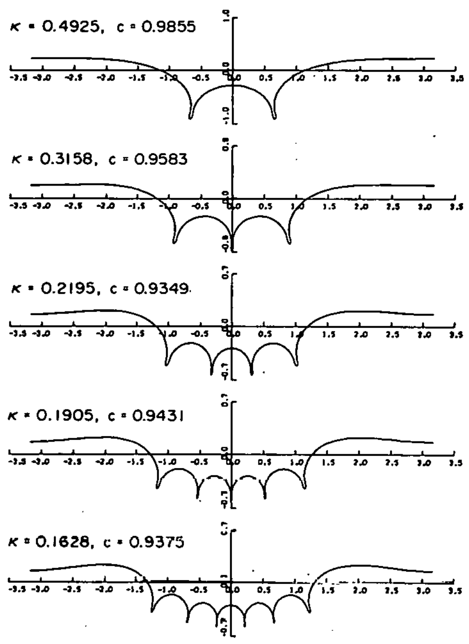


Figure 7. Waves of greatest height, N = 1, for various κ.

the approximate solution is

$$z = \left(\frac{A}{\pi}\right)^{\frac{1}{2}} e^{i\theta}\left(1 + \frac{1}{3}\pi\frac{A}{L^2}\cos 2\theta\right) . \qquad (9)$$

The problem is to determine the shape for finite values of A/L^2.

We restrict attention to shapes with the same symmetry as those of Eq. (9), i.e. elliptical with one axis (in fact the major axis) along the line of centers. A polar coordinate representation was used for the shape; $z = Re^{i\theta}$ where

$$R = a_0 + \sum_1^N a_{2n}\cos 2n\theta . \qquad (10)$$

Substitution into Eq. (8), and evaluating the integral by the trapezoidal rule at the mesh points

$$\theta_j = \pi j/2(N+1), \quad j = 1,2,\ldots,N \qquad (11)$$

gives N non-linear, transcendental equations for the (N+1) unknown Fourier coefficients. Putting the area equal to A closes the system.

Again Newton's method and Euler continuation in $A^{\frac{1}{2}}/L$ was used to construct solutions, with the solution of Eq. (9) giving a first guess for small $A^{\frac{1}{2}}/L$. Critical points were again detected by monitoring the Jacobian of the Newton iteration and handled without difficulty by pseudo-arc length continuation. For N = 80, each iteration took about 3 sec. Typical shapes (normalized on L = 1) are shown in Figure 8.

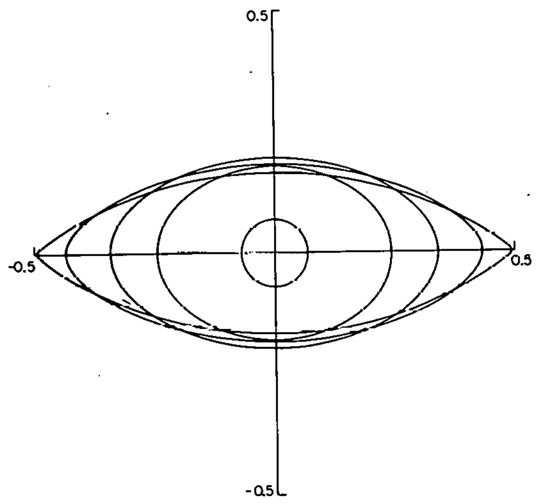


Figure 8. Shapes of vortices in an array.

A continuous branch of solutions exist for $0 < a/L \leq 0.5$, where 2a is the length of the major axis. At $a/L = 0.5$, the vortices touch and the continuation of the branch is a family of connected vortices which end up in a uniform vortex sheet of finite thickness, as shown in Figure 9.

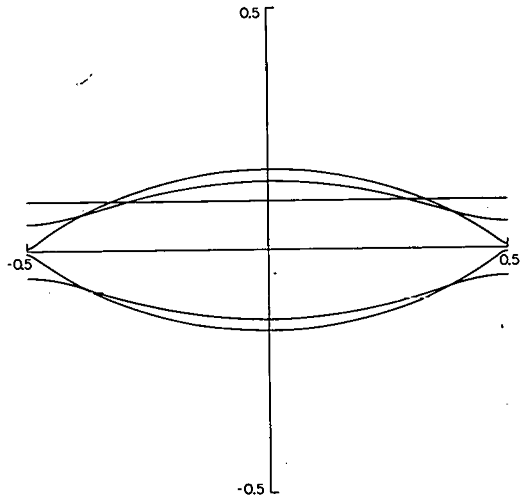


Figure 9. Shapes of connected vortices.

In Figure 10, we show a/L and b/L (b = semi minor axis) plotted against $A^{\frac{1}{2}}/L$. The full line shows the properties of isolated vortices. The dotted line described the connected continuation. (The dashed line is the prediction of an approximate analytical model by Saffman and Szeto[7]).

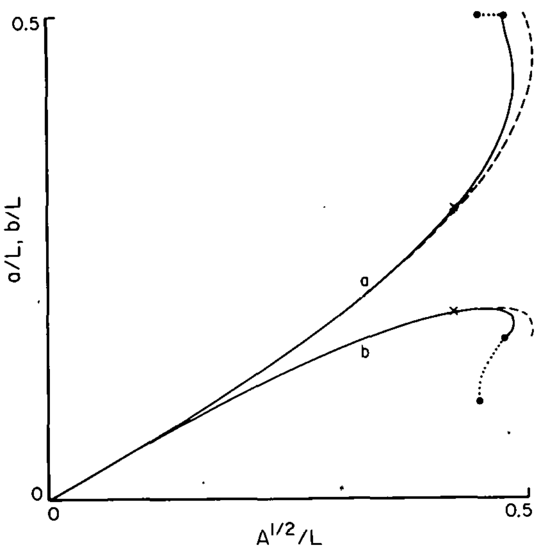


Figure 10. Dimensions of the vortices.

In this problem, no bifurcations were detected, but there is limit point behavior as the figure clearly shows that there is an upper limit on the value of $A^{\frac{1}{2}}/L$ for steady, symmetrical solutions to exist. The existence of this upper limit was guessed at by Moore and Saffman[8], and used by them to propose an explanation for the coalescence of coherent structures in the turbulent mixing layer.

The stability of the vortices to small disturbances is of interest. We repeat that the convergence of Newton's method is independent of the stability of the flow. Fortunately, a qualitative argument of Lord Kelvin enables us to make predictions about stability by calculating the energy of the configuration and avoids the necessity of calculating the eigenvalues of small disturbances. The excess energy per unit length (i.e. the difference in energy of the configuration and a vortex sheet of the same strength and zero thickness) is shown in Figure 11. The excess energy is a minimum for the vortex of maximum area. According to Kelvin's argument, the more circular or less deformed shapes with $A < A_{max}$ are therefore stable to two-dimensional disturbances, whereas the more deformed and connected vortices are unstable.

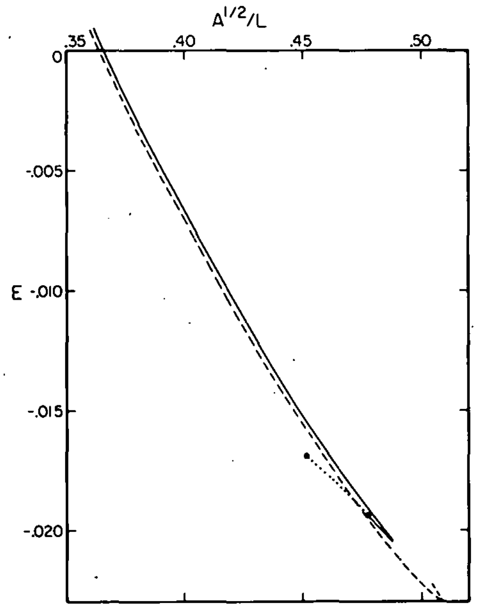


Figure 11. Energy excess of the array.

REFERENCES

[1]P.G. Saffman and G.R. Baker, Ann. Rev. Fluid Mech. 11, 95 (1979).
[2]B. Chen and P.G. Saffman, Stud. App. Math. 60, 183 (1979).
[3]H.B. Keller, Applications of Bifurcation Theory p. 359. Academic Press (1977).
[4]E.D. Cokelet, Phil. Trans. Roy. Soc. A286, 183 (1977).
[5]B. Chen and P.G. Saffman, New types of gravity waves and finite amplitude steady capillary gravity waves. Stud. App. Math. (to appear).
[6]A. Roshko, A.I.A.A.J. 14, 1349 (1976).
[7]P.G. Saffman and R. Szeto, Structure of a linear array of uniform vortices. Submitted to J. Fluid Mech.
[8]D.W. Moore and P.G. Saffman, J. Fluid Mech. 69, 465 (1975).

STEADY HIGH REYNOLDS NUMBER FLOW PAST A CYLINDER

Bengt Fornberg
Department of Applied Mathematics 101-50
California Institute of Technology
Pasadena, California 91125

ABSTRACT

Viscous flow past a circular cylinder becomes unstable around Reynolds number Re = 40. With a new numerical technique, based on Newton's method, steady (but unstable) solutions of high accuracy have been obtained up to Re = 300. A new trend in the solution was found when the Reynolds number was increased above 260. The wake bubble begins to decrease in length as vorticity in the wake is convected back towards the body. The numerical calculations were performed on the CDC STAR-100 computer at the CDC Service Center in Minneapolis.

INTRODUCTION

Viscous steady flow past a circular cylinder at high Reynolds numbers has become one of the classical problems in numerical fluid mechanics. There are several reasons for the continuing interest in this problem. One is that it forms a good model problem for flows around other bodies of more practical interest. Steady solutions for Reynolds numbers (based on the diameter) higher than 40 may in the future be achieved by flow control methods. This may give rise to flows with practical applications. Complete, steady flow fields have so far only been obtained numerically up to around Re = 100. The first reference[1] gives a brief survey of previous work and describes also this present work in some detail.

Contradictory suggestions have been made for the limit of $Re \to \infty$. Brodetsky[2] suggests a solution with vortex sheets bounding an infinite wake region containing stagnant flow. Batchelor[3] suggests a limit which has a finite wake with piecewise constant vorticity and no drag on the body. Up to $Re \approx 100$, all evidence has been in support of a wake growing approximately proportional to Re. We will see a quite sudden reversal of trends around Re = 260, which casts definite doubt on Brodetsky's solution ('the free streamline' model).

Our numerical calculations were performed on the Control Data Corporation STAR-100 Computer located at the CDC Service Center in Minneapolis, Minnesota. We wish to express our gratitude to Control Data Corporation for making this system available to us. The solution of large banded linear systems was the most time-consuming part of the present calculations. These solutions ran about 200 times faster on the CDC STAR-100 than on the Caltech IBM 370/158 (which was used for some preliminary tests and the graphical output).

MATHEMATICAL FORMULATION

With a unit cylinder and Re based on the diameter, the Navier-Stokes equations take the form

$$\Delta\Psi + \omega = 0 \tag{1}$$

$$\Delta\omega + \frac{Re}{2}\left\{\frac{\partial\Psi}{\partial x}\cdot\frac{\partial\omega}{\partial y} - \frac{\partial\Psi}{\partial y}\cdot\frac{\partial\omega}{\partial x}\right\} = 0 \tag{2}$$

In most of the work we use

$$\psi(x,y) = \Psi(x,y) - y \tag{3}$$

This research was supported by Control Data Corporation and ERDA (Grant No. 04-3-767).

instead of Ψ. This variable ψ gives the streamlines of the perturbation from free stream.

The main problems that earlier investigators have encountered are

1. Boundary conditions for ψ at large distances.
2. Boundary condition for ω at the body surface.
3. Convergence rate of numerical iterations.
4. Convergence to a smooth solution without a loss of accuracy that goes with upwind differencing.
5. Economical choice of computational grids.

Our numerical method, described in the next section, was designed specifically to overcome these difficulties, if necessary at the price of a high computational cost per iteration.

NUMERICAL METHOD

All vorticity is concentrated on the body surface and in a quite thin streak downstream of the body. Outside this region, we can use the much simpler equations

$$\Delta\psi = 0 \tag{4}$$

$$\omega = 0 \tag{5}$$

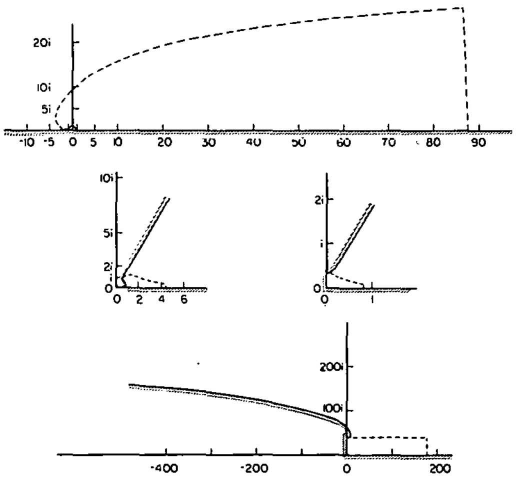


Fig. 1. Conformal mapping of the inner region.

The top part of Fig. 1 shows a region which includes all the vorticity. This will form the inner of the two computational regions we will be using. The rest of Figure 1 illustrates the steps in a conformal transformation to a rectangle. The steps are

1. $\rho = x^{1/3}$
2. $z = c(\rho - \frac{1}{\rho})$ (constant $c = .2$)
3. $\zeta = \frac{z}{c^3}(1 + z^2)$

The inverse transformation can also be expressed explicitly.

The Navier-Stokes equations were transformed to this new coordinate system (also stretched to increase boundary layer resolution at the surface) and approximated numerically in a straightforward way (centered, second order accurate approximations). Newton's method was then used to solve this system together with the boundary conditions. There are two conditions for ψ on the surface (no fluid passing the surface and no slip), and

$$\frac{\partial\psi}{\partial y} = \frac{\partial\omega}{\partial y} = 0$$

on $y = 0$. On the curved upper side of the region, we have $\omega = 0$ and we require $\frac{\partial\omega}{\partial x} = 0$ on the far right edge. The physics require no information on ω from the outflow side. This freedom can be used to eliminate the possibility of staggered mesh oscillations for ω, a frequent complication with centered approximations for the vorticity transport equation at high Reynolds numbers. To find ψ on the upper and right sides, an outer computational region has to be introduced and an iteration between the grids will be performed.

A polar coordinate system can be introduced by

$$\xi + i\eta = \frac{1}{\pi}\ \ell n(x + iy) \tag{6}$$

and refined in the wake by

$$\eta' = \eta^{1/2} \tag{7}$$

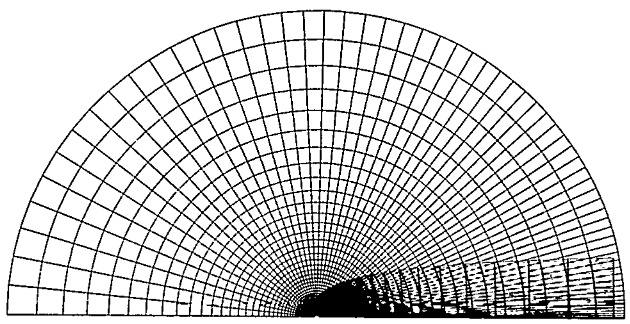


Fig. 2. Outer and inner computational grids.

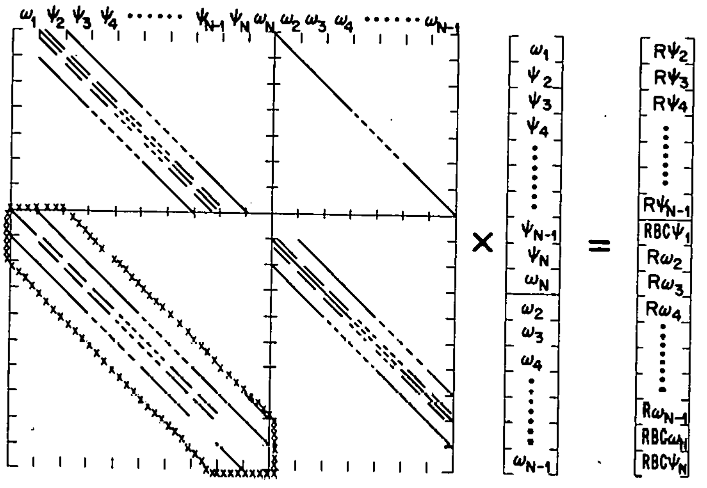
We express now equation (1) in ξ, η'-coordinates and solve for ψ by 'black-red'-ordered SOR. Figure 2 shows the two computational grids superimposed. In the actual calculations, the grids had twice the illustrated densities. (The inner grid had 65x114 points, the outer grid 129x132 points. Both extended to 600 radii from the cylinder). The outer boundary condition for ψ on this last grid has been discussed in detail[1]. It was found that the usual free stream $(\psi = 0)$ was very unsatisfactory but that $\frac{\partial\psi}{\partial\xi} = 0$ worked successfully at low Reynolds numbers. At high Reynolds numbers, a satisfactory condition of the form

$\frac{\partial\psi}{\partial\xi} = f(\psi)$ was found.

Our final complete method was a repetition of the following four steps:

1. Perform one Newton iteration on the inner grid.
2. Interpolate ω to the outer grid.
3. Solve for ψ on the outer grid.
4. Interpolate ψ-values back to the edge of the inner grid.

Although step 1 in itself is quadratically convergent, this inner-outer iteration scheme converged only linearly. The convergence was nevertheless very rapid, about a factor of 10 per each cycle of the four steps. Six to eight cycles were sufficient in all cases, each costing about one minute of computer time.



ω_1 ψ_2 ψ_3 ψ_4 ...... ψ_{N-1} ψ_N ω_N ω_2 ω_3 ω_4 ...... ω_{N-1}
×
ω_1 ψ_2 ψ_3 ψ_4 ψ_{N-1} ψ_N ω_N ω_2 ω_3 ω_4 ω_{N-1}
=
$R\psi_2$ $R\psi_3$ $R\psi_4$ $R\psi_{N-1}$ $RBC\psi_1$ $R\omega_2$ $R\omega_3$ $R\omega_4$ $R\omega_{N-1}$ $RBC\omega_N$ $RBC\psi_N$

Fig. 3. Structure of the Jacobian matrix.

Figure 3 shows the structures of the Jacobian matrix that entered as coefficient matrix in Newton's method. Here, ω_1 denotes a vector of ω-values on the first grid line (body surface), ψ_2 the ψ-values on the next grid line etc. The right-hand-side contains the residuals in the different equations and boundary conditions. The particular ordering shown gives a structure that allows immediate simplification. With use of suitable multiples of the equations in the top half, all entries in the bottom right corner can be eliminated. We are left with a banded system confined to the dotted region in the bottom left corner. It will contain only 13 non-trivial diagonals. Table 1 shows its size and the cost to solve it by Gaussian elimination.

size of inner grid	M*N = 65*114
system band width	4*M-7 = 253
number of equations	(M-2)*(N+1) = 7245
CPU-time on CDC STAR-100	
LU-decomp.	29.3 s.
Back Subst.	1.3 s.
estimated time on IBM 370/158	
LU-decomp	1 hr. 40 min.

RESULTS

The flow results are illustrated in Figures 4-14 below. The most prominent feature is the recirculation of vorticity starting around Re = 260. It affects quite dramatically some flow quantities (like the length and width of the wake bubble) but leaves others (like the drag coefficient, pressure distribution etc) quite unaffected. The calculations were not carried beyond Re = 300 since higher Reynolds numbers would have required a still finer grid. A grid twice as dense would have been too costly at the present time.

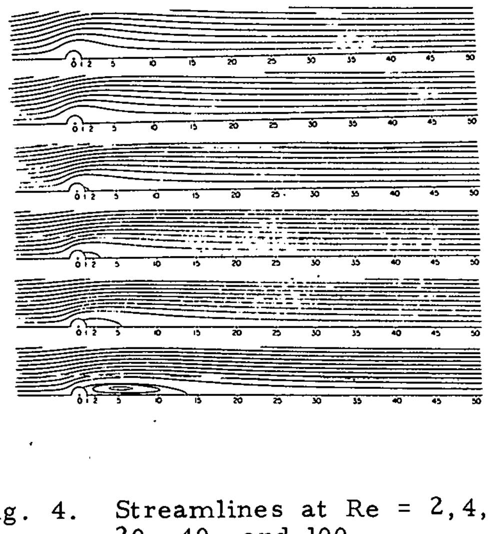


Fig. 4. Streamlines at Re = 2, 4, 10, 20, 40, and 100.

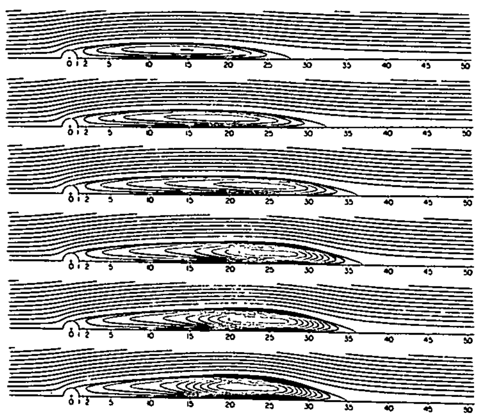


Fig. 5. Streamlines at Re = 200, 230, 260, 290, 295 and 300.

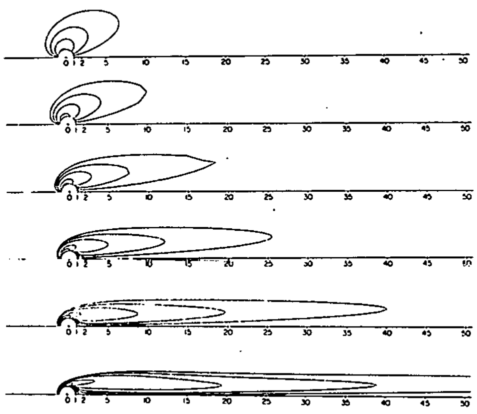


Fig. 6. Lines of equal vorticity at Re = 2, 4, 10, 20, 40 and 100. (The first three cases were obtained by a different method).

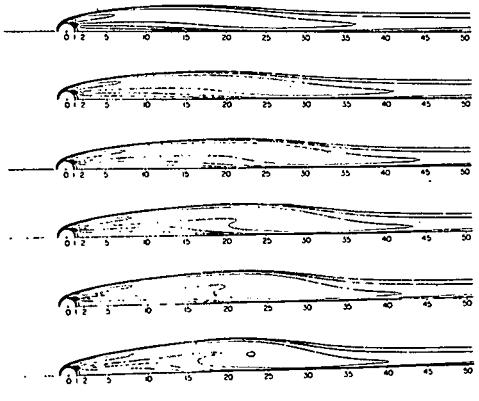


Fig. 7. Lines of equal vorticity at Re = 200, 230, 260, 290, 295 and 300.

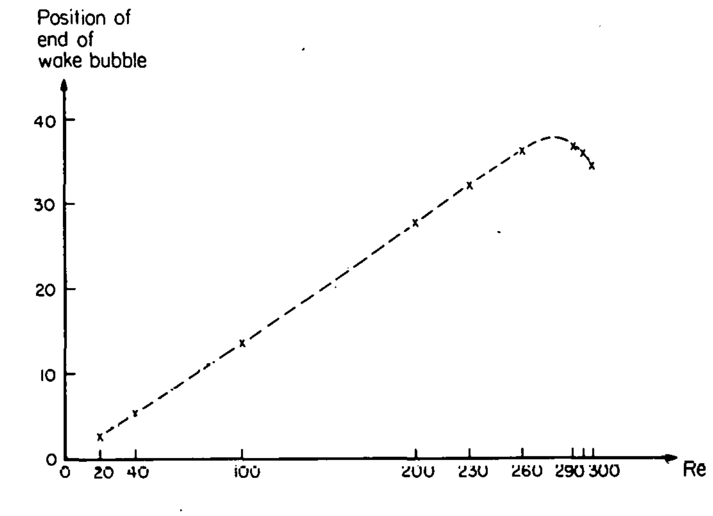


Fig. 8. Position of end of wake bubble.

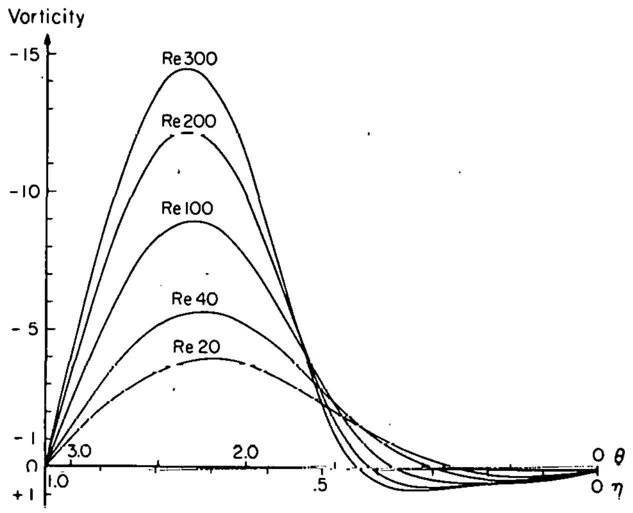


Fig. 10. Vorticity distribution on the body surface.

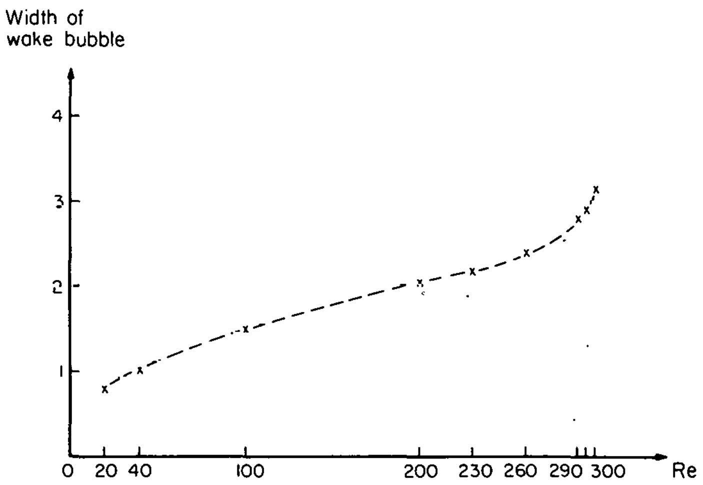


Fig. 9. Width of wake bubble.

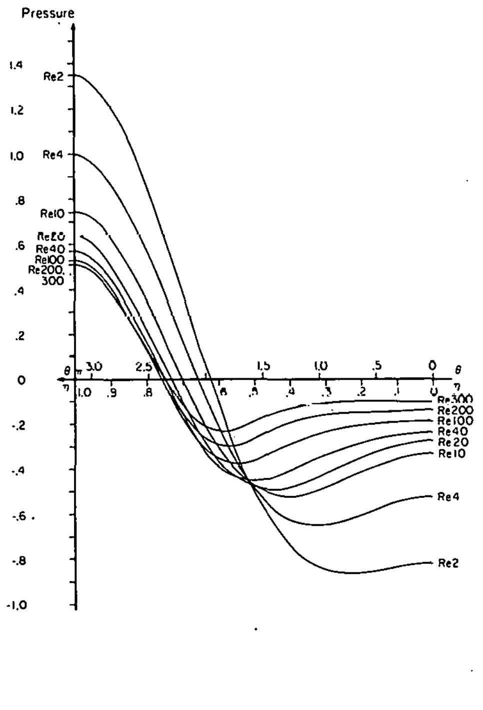


Fig. 11. Pressure distribution on the body surface.

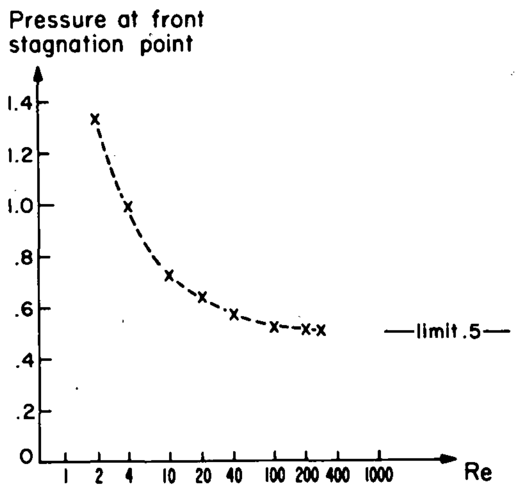


Fig. 12. Pressure at the front stagnation point.

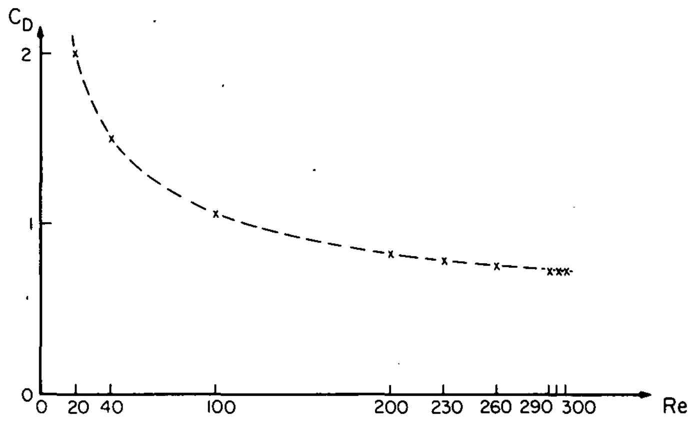


Fig. 14. The drag coefficient.

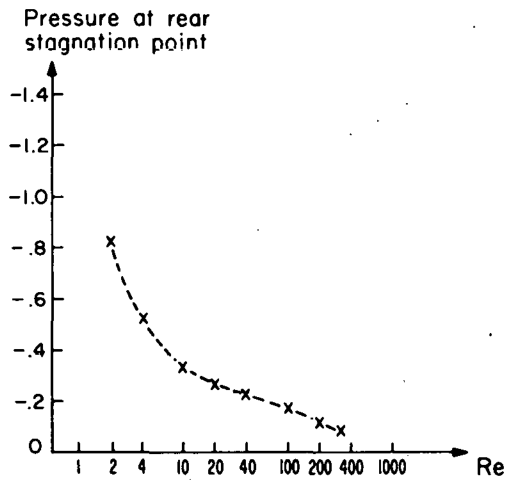


Fig. 13. Pressure at the rear stagnation point (on the body).

REFERENCES

1. Fornberg, B., A numerical study of steady viscous flow past a circular cylinder. Submitted to JFM.

2. Brodetsky, S. Proc. Roy Soc. London A102, 542 (1923).

3. Batchelor, G.K., JFM 1, 338 (1956).

VECTORIZATION TECHNIQUES FOR AN ITERATIVE ALGORITHM

Dennis V. Brockway
Fred Gama-Lobo
Los Alamos Scientific Laboratory
P. O. Box 1663
Los Alamos, New Mexico 87545

ABSTRACT

A major vectorization effort on a large complex hydrodynamics code on the CRAY-1 was recently completed, resulting in a factor of 10-speed increase over the original code on the CDC-7600. To accomplish this required vectorizing an iterative algorithm. Some new techniques were developed to do this, which will be described in detail.

INTRODUCTION

A major effort was completed recently to vectorize a large 2D Lagrangian hydrodynamics code for the CRAY-1. This particular code was chosen because it was the largest consumer of computer time at LASL. When this code was first converted from the CDC-7600 to the CRAY-1, which was just a straight FORTRAN conversion with minimal changes, it gained a factor of 2.5 in speed. The initial target of the vectorization effort was to get up above a factor or 4 over the CDC-7600. The result of the vectorization was a factor of 10 speedup over the CDC-7600 (or a factor of 4 over CRAY-1 scalar code).

The reason our initial target was so low was that approximately 40% of the execution time of the code was spent solving an iterative algorithm that did not appear at first analysis to be vectorizable. Assuming we got a factor of 10 for the section of the code taking 60% of the time and remained at a factor of 2.5 for the sections of code taking 40% of the time, then the overall factor F can be computed from

$$1/F=0.6 \times (1/10) + 0.4 \times (1/2.5)$$

resulting in F=4.5.

We did, however, develop a successful technique for vectorizing the iteration, which will be described in what follows. As a result, we were able to vectorize about 95% of the code so that we actually got a factor of 10

OVERALL VECTORIZATION EFFORT

CODE REWRITE

To vectorize the hydro code we rewrote nearly all of the computational section of the code. This was done by restructuring the code into simple DO loops based on the basic guidelines for vectorization describ- in Chapter 4 of the *CRAY-1 FORTRAN (CFT) REFERENCE MANUAL*[1] and in the paper *How To Get More from Your Vector Processor*.[2] We went beyond these basic techniques in various places in the code, and this paper describes some of those.

CFT COMPILER

The vectorized code was written mainly in standard FORTRAN using the CFT Compiler, which automatically generates vector instructions for simple DO loops. The main exception was the use of the conditional vector merge statements available in CFT. These are equivalent to simple IF statements, and whenever they were used in the code, they were preceeded by a comment describing the equivalent IF.

SAMPLE

We made an extensive use of a routine called SAMPLE, which determines the distribution of time during code execution and produces a histogram of time used per subroutine and for locations within each subroutine.[3] Initially, the distribution of time in the code was fairly flat with no single subroutine taking a large percentage of the time, which prompted our decision to rewrite the entire computational section of the code. After we vectorized a major portion of the code, we used SAMPLE to locate those places that still took significant amounts of time and were candidates for fine tuning.

OVERALL EFFORT

The overall effort was considerable

due to the size and complexity of the code. It took 3 people 6 months to complete the effort, which was actually quite a bit less manpower than early projections predicted. This was due to some extent to the people being quite familiar with the code to begin with and the use of mostly standard FORTRAN. The people who did the work were Dennis Brockway, Fred Gama-Lobo, and Karl Wallick of Group TD-9 at LASL. We would also like to acknowledge the work of Alex Marusak and Don Willerton of C-3, who wrote SAMPLE.

VECTORIZATION OF THE ITERATIVE ALGORITHM

CODE DESCRIPTION

The hydro code calculates motion in an object represented by a 2D mesh of quadrilaterial zones. All mesh variables are held in doubly dimensioned arrays and in each computational cycle a time step is taken; all the hydrodynamics calculations are performed on these variables in double DO loops. To vectorize the code it has to be rewritten so that all inner loops are simple calculational loops that satisfy the conditions for vectorization in the CFT compiler.[1]

TTS ITERATION

The iteration that took up 40% of the execution time in the scalar code is called Temporary Triangular Subzoning (TTS).[4] For TTS each quadrilateral zone is divided into four triangles. For each triangle the internal energy is computed from the equation

$$E(\text{triangle}) = EO - 0.5 \times \left(P(\text{triangle}) + P(\text{zone})\right) \times DTAU(\text{triangle})$$

All quantities in this equation can be computed directly except for the Energy and Pressure of the triangle, which are both functions of Temperature and Density. These functions are defined in Equation of State (EOS) Tables. The Density is known, but the Temperature is not, so an iteration must be performed to find a Temperature such that E(triangle) and P(triangle) satisfy the above equation.

EOS TABLE LOOKUP

The EOS table lookup does not require a table search since values are equally spaced based on logarithms of base 2. Given a Temperature and Density, indices to retrieve values for Energy and Pressure from the tables can be calculated directly using the LOG function. For each table lookup the code needs to do a LOG to compute indices to the table, fetch 8 quantities from memory by indirect addressing (gather), and do 2 exponential interpolations requiring 2 calls to EXP.

The reason the TTS is so time consuming is because of the great number of times that the EOS table lookup must be done and that each table lookup requires a LOG, 2 EXP's and 8 gathers. The table lookup must be performed for each iteration for each of the 4 triangles for each of the zones. In a typical problem there are 7000 zones, and it takes 3 iterations on the average for the TTS equation to converge. That means that for each computational cycle the table lookup is done 3x4x7000 (or 84,000 times).

TTS FLOW

The following is the flow of the TTS iteration in the original scalar code:

```
     DO 100 K=1,70
     DO 100 L=1,100
     DO 100 ITRI=1,4

     Step 1.  Compute EO , PZONE , DTAU

     Step 2.  Compute  initial guess of
              Temperature T

     Step 3.  Call EOS to look up ETRI and
              PTRI

     Step 4.  Test for convergence -

     ENG = EO - 0.5 * (PTRI + PZONE) * DTAU
     TEST = ABS ( (ENG - ETRI) / ENG )
     IF (TEST .LT. 1.E-5) GO TO 100

     Step 5.  Compute new guess of tempera-
              ture T

     GO TO Step 3

100 CONTINUE
```

VECTORIZATION OF TTS

The first step in speeding up the TTS was vectorizing the table lookup, which consisted of writing a routine producing arrays of results rather than a single result. The new routine is a factor of 4 faster than CRAY scalar per table lookup. The main reason for the speedup is that the vector LOG and EXP functions can now be used. These functions are provided by CRAY Research in their library supporting the CFT compiler. They are used by referencing ALOG and EXP just as is done in standard FORTRAN. If the references to the functions are in a DO loop which otherwise satisfies the CFT conditions for vectorization, the compiler will automatically generate calls to the vector functions. These functions are approximately a factor of 15 times faster per element than their CRAY scalar counterparts. We were disappointed that we were unable to speed up the gathers in the routine, but the speed increase in LOG and EXP still allowed us to make a very good gain overall for the EOS table lookup.

The next step was to vectorize the iteration itself. This consisted of computing all the terms in the TTS equation in vector loops, calling the table lookup routine to get arrays of values for E(triangle) and P(triangle) based on an array of initial guesses for Temperature, and then computing the difference between the left and right hand sides of the equation in a vector loop. This difference is used for testing convergence. An array IDOESIT is used to keep track of convergence. For a set of iterations being done in vector loops, we set IDOESIT(L)=1 if this iteration for zone L did not converge and set IDOESIT(L)=0 if this iteration converged. Then we check this array to see if any iterations have not converged; and if any have not, new guesses for Temperature are computed from appropriate derivatives and the process is repeated until all iterations converge.

VECTORIZED TTS FLOW

The following is the flow of the vectorized TTS iteration:

```
     DO 100 ITRI=1,4
     DO 100 K=1,70

     DO 10 L=1,100

     Step 1.  Set IDOESIT(L)=1 for zones
              flagged for TTS

10 CONTINUE

     DO 20 L=1,100

     Step 2.  Compute EO(L) , PZONE(L)
              DTAU(L) in vector loop

     Step 3.  Compute guess of Temperature
              T(L) in vector loop

20 CONTINUE

     Step 4.  Call up vectorized EOS to look
              up arrays ETRI and PTRI

     DO 30 L=1,100

     Step 5.  Compute convergence criteria
              in vector loop

     ENG(L) = EO(L) - 0.5 * (PTRI(L) + PZONE(L) * DTAU(L)
     TEST(L) = ABS ((ENG(L) - ETRI(L)) / ENG(L)

     Step 6.  Set IDOESIT(L) = 0 if
              convergence -

     IDOESIT(L) = CVMGP (IDOESIT(L) , 0 , TEST(L) - 1.E-5)

Note:  The above CFT conditional vector
       merge statement is equivalent to
       the following IF statement -

     IF (TEST(L) .LT. 1.E-5) IDOESIT(L) = 0

30 CONTINUE

     DO 40 L=1,100

     Step 7.  Check if any iterations have
              not converged -

     IF (IDOESIT(L) .NE. 0) GO TO 50

40 CONTINUE

     GO TO 100

50   DO 60 L=1,100

     Step 8.  Compute new guess of  empera-
              ture T(L) in vector loop

60 CONTINUE

     GO TO Step 4

100 CONTINUE
```

LOOP LIMITS

After vectorizing the TTS by the above method, we merely broke even with the scalar code. The reason was that on the average it takes 3 iterations for the TTS equation to converge, but there are usually some triangles in the inner loop that take twice as many or more iterations to converge. Some anomalous triangles may take 20 to 30 iterations to converge. In the above method all the calculations in the inner loops are done for all the triangles as many times as it takes for the worst triangle to converge. This meant that the vector code was doing at least twice the calculations of the scalar code, which wiped out any gains due to using vector operations.

To get around this problem we varied the limits of the inner loops each cycle of the iteration to avoid doing most of the unnecessary calculations. The way this was done is to use the array IDOESIT and set limits based on the ranges of non-zero values in the array. Suppose IDOESIT has the following set of values:

```
0 0 0 1 1 1 1 1 0 0 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 0 0
      4       8     11 13                   23  25
```

For this case the code would set up 3 sets of limits, 4-8 and 11-13 and 23-25, and the inner loops would be repeated for these 3 sets of limits. The code will actually go up to 4 sets of limits. If there are more than four, it will bridge all but the 4 largest gaps of zeros. It will also bridge a gap of zeros smaller than 10% of the size of the array. Therefore, in the above example the zeros in positions 9 and 10 would be bridged and the code would actually only set up 2 limits, 4-13 and 23-25.

FLOW FOR VECTORIZED TTS WITH VARYING LIMITS

```
     DO 100 ITRI=1,4
     DO 100 K=1,70

     DO 10 L=1,100

     Step 1.  Set IDOESIT(L) = 1 for zones
              flagged for TTS

10 CONTINUE

     DO 20 L=1,100

     Step 2.  Compute EO(L) , PZONE(L) ,
              DTAU(L) in vector loop
     Step 3.  Compute guess of Tempera-
              ture  T(L) in vector loop

20 CONTINUE

     Step 4.  Set up limits based on
              IDOESIT.  N is the number of
              limits, LIM1 is the array of
              lower limits, and LIM2 the
              array of upper limits.

25   DO 35 I=1,N

     L1 = LIM1(I)
     L2 = LIM2(I)

     Step 5.  Call vectorized EOS to look
              up arrays ETRI and PTRI for
              L values L1 to L2.

     DO 30 L=L1,L2

     Step 6.  Compute convergence criteria
              in vector loop -
   ENG(L) = EO(L) - 0.5 * (PTRI(L) * PZONE(L)) * DTAU(L)
   TEST(L) = ABS ((ENG(L) - ETRI(L) / ENG(L))

     Step 7.  Set IDOESIT(L) = 0 if conver-
              gence -
   IDOESIT(L) = CVMGP (IDOESIT(L) , 0 , TEST(L) - 1.E-5)

30 CONTINUE

35 CONTINUE

     DO 45 I=1.N

     L1 = LIM1(I)
     L2 = LIM2(I)

     DO 40 L=L1,L2

     Step 8.  Check is any iterations have
              not converged -

     IF (IDOESIT(L) .NE. 0) GO TO 50

40 CONTINUE

45 CONTINUE

     GO TO 100

50 CONTINUE

     Step 9.  Reset limits based on IDOESIT

     DO 65 I=1,N

     L1 = LIM1(I)
     L2 = LIM2(I)

     DO 60 L=L1,L2

     Step 10. Compute new guess of tempera-
              ture T(L) in vector loop.

60 CONTINUE

65 CONTINUE

     GO TO 25

100 CONTINUE
```

USE OF THE VECTOR MASK

After putting in the varying limits, we got a speed increase over a factor of 2 over scalar. Use of SAMPLE[3] indicated that a significant amount of time was still being spent in some remaining scalar loops that had previously been ignored and in the scalar calculation of the limits described above. The calculation of the limits required a scalar search of the array IDOESIT for non-zero elements. The way we vectorized this process was to write an assembly language routine to create a vector mask from the array IDOESIT. If IDOESIT has the values in the example used before, the vector mask would consist of a single word with the bit pattern:

```
     000111110011100000000001110 ... 0
```

The CFT functions LEADZ (tally of leading zeros) , SHIFTL (left shift) , and COMPL (bit-by-bit) logical complement)[1] are then used to successively count zero and non-zero bits in the mask and thus set up the limits.

SUMMARY

After eliminating most of the remaining scalar code in inner loops, the TTS ran a factor of 3.8 faster than scalar. The gains in the rest of the code averaged above a factor of 4 so that the overall gain was a factor of 4 over CRAY-1 scalar (or a factor of 10 over the CDC-7600).

REFERENCES

1. CRAY-1 FORTRAN (CFT) Reference Manual, Manual #2240009, Cray Research, Inc., 1/79.

2. Brian Q. Brode, How To Get More Out Of Your Vector Processor, Massachusetts Computer Associates, 9/78.

3. Alex Marusak and Don Willerton, Determining Distribution of CPU Time Used on The CRAY-SAMPLE, LASL Program Library writeup, 7/79.

4. Philip L. Browne and Karl B. Wallick, The Reduction of Mesh Tangling in Two-Dimensional Lagrangian Hydrodynamics Codes By the Use of Viscosity, Artifical Viscosity, and TTS (Temporary Triangular Subzoning for Long Thin Zones), LASL Document LA-4740-MSm 11/71.

EVOLUTION OF THE MHD "SHEET PINCH"

W. H. Matthaeus and D. Montgomery
Physics Department, William and Mary
Williamsburg, VA 23185

ABSTRACT

A magnetohydrodynamic (MHD) problem of recurrent interest for both astrophysical and laboratory plasmas is the evolution of the unstable "sheet pinch", a current sheet across which a dc magnetic field reverses sign. We follow the evolution of such a sheet pinch with a spectral-method, incompressible, two-dimensional, MHD turbulence code. Spectral diagnostics are employed, as are contour plots of vector potential (magnetic field lines), electric current density, and velocity stream function (velocity streamlines). The non-linear effect which seems most important is seen to be current filamentation: The concentration of the current density onto sets of small measure near a magnetic "X point." A great deal of turbulence is apparent in the current distribution, which, for high Reynolds numbers, requires large spatial grids ($\geq (64)^2$).

INTRODUCTION

We report a numerical solution of the problem of an evolving MHD "sheet pinch": a topic which has generated a voluminous literature, but about which unanswered questions remain. The problem is inherently a turbulence problem, involving spatial excitations over a wide range of spatial scales, thus requiring high spatial resolution. Our grid size (64 x 64) is at the lower limit of what is required to compute the phenomenon accurately. We utilize a two-dimensional, incompressible MHD code of the Orszag-Patterson[1] spectral type (Galerkin approximation), employing periodic boundary conditions. The periodic boundary conditions demand certain compromises with the physics, but the gain in computational simplicity is great.

TEXT

The initial magnetic field line geometry for all runs is shown in Fig. 1. The periodic boundary conditions require two current sheets, into and out of the xy plane, parallel to the xz plane. All variables are assumed z-independent. If the two current sheets are far enough apart, their interaction should be minimal, and we do not believe the evolution to be significantly different than it would be for a single current sheet. The basic square is 64 x 64 cells, with a maximum-to-minimum wave number ratio of 32. The current sheets are about four cells wide, and the magnetic field reverses sign twice, once across either current sheet. The box size, in our units, is 2π units of length.

The velocity field $\underset{\sim}{v}$ and the magnetic field $\underset{\sim}{B}$ are in the xy plane and the vector potential $\underset{\sim}{a} = a\hat{e}_z$ is normal to it. $\underset{\sim}{B} = \nabla a \times \hat{e}_z$, and $\underset{\sim}{v} = \nabla\psi \times \hat{e}_z$, where ψ is the stream function. The vorticity is in the z direction, and has magnitude $\omega = -\nabla^2\psi$, while the vector potential and current density j are also related by Poisson's equation: $\nabla^2 a = -j$. The direction of the current density is along the z axis. The time evolution comes from advancing the (Fourier-transformed) pair of equations:

$$\frac{\partial a}{\partial t} + \underset{\sim}{v}\cdot\nabla a = \mu\nabla^2 a \qquad (1)$$

$$\frac{\partial \omega}{\partial t} + \underset{\sim}{v}\cdot\nabla\omega - \underset{\sim}{B}\cdot\nabla j = \nu\nabla^2\omega \qquad (2)$$

We have used the dimensionless units of Fyfe et al,[2] whose papers should be consulted for a detailed description of the method. In these units, the dimensionless magnetic diffusivity μ and dimensionless viscosity ν are the reciprocal magnetic and mechanical Reynolds numbers. The case of most physical interest is the case where μ and ν are small but non-zero.

The initial dc magnetic field can be well represented by the Fourier modes with $k_x = 0$ and $k_y = \pm1, \pm3, \pm5, \ldots, \pm15$. We call these the "sheet pinch modes", and after $t = 0$, allow their Fourier coefficients to advance on the same footing as all the others. The unstable growth is initiated by adding small random values to the Fourier coefficients of the non-sheet pinch modes. This random initial noise is small enough that the non-sheet pinch Fourier amplitudes are typically down from the sheet pinch Fourier amplitudes by factors of $<10^{-3}$.

After several tens of time steps, some of the non-sheet pinch modes have temporally growing components which emerge from the initially rather unsystematic MHD activity observed at the outset. The most rapidly growing $\underset{\sim}{k}$ modes move slowly out to values of the order of $k \cong 10$, but do not get near the maximum k of 32. Systematic growth of both the kinetic energy and non-sheet pinch magnetic energy are observed for a few thousand time steps. Most of the magnetic excitation is initially, and remains in the modes with $k^2 = 1$. The modes $\underset{\sim}{k} = (0,\pm1)$ are sheet pinch modes and those with $\underset{\sim}{k} = (\pm1,0)$ are non-sheet pinch modes. Except for a very slow drain of the $(0,\pm1)$ modes by the $(\pm1,0)$ modes, the growth of the kinetic energy and non-sheet pinch magnetic energy appears to have ceased by about 5000 time steps even for $\mu=\nu=0$ (an unphysical case, but nonetheless an instructive one, to be discussed presently). For finite μ, ν, the saturation occurs even sooner, and the total energies decay throughout the run, as predicted by Eqs. (1) and (2). This decay can be kept small by keeping μ and ν small enough.

It is instructive to consider first the (unphysical) case $\mu=\nu=0$. Fig. 2 shows the time evolution of the non-sheet pinch magnetic energy, the total mean square current, the total mean square vector potential, and the total kinetic energy. During the growth phase, the most active Fourier modes are in the range $k \cong 10$. The growth qualitatively speeds up at approximately time step 1000, and saturates near time step 5000. Late in the run, it is necessary to halve the time step in order to preserve the conserved quantities. After saturation, the spectrum appears to be heading for an absolute equilibrium spectrum[2], but does not reach it over the times we compute: strong anisotropy in $\underset{\sim}{k}$ space remains. Rather surprisingly, the state in the region of limiting non-sheet pinch magnetic energy bears considerable similarity to that for the finite dissipation cases. Fig. 3a shows the contours of constant current density j in xy space at $t = 0$. Fig. 3b shows the j contours after 2000 time steps. The current distribution, close to a uniform sheet in Fig. 3a, has filamented, and has concentrated itself in sets of small measure near a magnetic field zero of the "X point" type. This effect appears to be fundamental.

Since we are working with a finite, discrete representation of the fields, the various pointwise invariants (or "topological" invariants) of ideal MHD are not conserved. Contrary to what is sometimes asserted, this non-conservation is not connected with dissipation. The present system is non-dissipative.

At a still later time (4500 time steps), Fig. 3c shows that the j contours have scattered randomly and essentially homogeneously over the square; this feature is not observed at finite μ and ν. Much less activity is visible in the contour plots of constant vector potential a (magnetic field lines, in two dimensions) for the same three times in Figs. 1, 4a, and 4b. The vector potential spectrum is dominated by the longest wavelength terms, and in that part of the spectrum, strong anisotropy persists.

Figs. 5a, b, c are contour plots of constant ψ (velocity streamlines) at times $t = 0$, after 2000 time steps, and after 4500 time steps. Fig. 5b shows a characteristic "jetting" of the magnetofluid: the fluid is rather violently expelled laterally from the weak pair of corners at the X point in the magnetic field. This has been seen also in a very different kind of computation by Sato and Hayashi.[3] Finally, Fig. 6 shows the directionally averaged $\underset{\sim}{B}$ and $\underset{\sim}{v}$ field spectra to which the configuration has evolved, by the end of the run. This is not an absolute equilibrium spectrum[2], and considerable anisotropy has been obscured by the directional averaging. Solid lines are equilibrium spectra.[2]

The preceding results cannot claim accurately to represent the physics, but they do anticipate some of the conclusions for finite μ and ν. Table 1 lists some important parameters for both the run $\mu=\nu=0$ just described and for $\mu=\nu=.0025$, the other case for which we shall present results here (a more extended presentation will be given elsewhere[4]). Fig. 7 shows the time history of the bulk quantities (non-sheet pinch magnetic energy, mean square current, etc., as in Fig. 2) for $\mu=\nu=.0025$. Figs. 8a, b show the contours of constant j at time steps 2000 and 5500. The finite μ, most effective at high k, has wiped out much of the short wavelength activity apparent in Fig. 3c; the sheet pinch geometry remains visible to the end of the run. Figs. 9a, b show the magnetic field lines (which give little indication of the disordered activity shown in Figs. 8a, b) at time steps 2000 and 5500. In all runs carried out, single magnetic "islands" were always the end product, as far as magnetic structure was concerned. The filamentation of the current and the jetting of the velocity field were also always observed. Figs. 10a, b show stream function contours at time steps 2000 and 5500, and show a strong persistence of the jetting, or horizontal magneto-fluid expulsion, to the end of the run.

A qualitative physical picture of the unstable development and filamentation might go as follows, keeping in mind the fact that in two dimensions, the contours of constant vector potential a are magnetic field lines, and both j and $\underset{\sim}{B}$ are expressible as spatial derivatives of a. The magnetic volume force on an element of fluid can be shown to be $(\nabla\times\underset{\sim}{B}) \times \underset{\sim}{B} = j\nabla a$. Moreover, j generates a through Poisson's equation, $\nabla^2 a = -j$. Thus, even though electric current distributions are not "frozen in" to the fluid, two fluid elements carrying currents in the same direction attract each other. Current filaments distributed around a magnetic "O" point (a maximum in a, if j is out of the paper) will feel a force toward the O point, but the fluid elements cannot move toward the O point because the velocity field is divergenceless; effectively, collapse toward an O point is prohibited by the mechanical pressure which builds up. No such prohibitions occur concerning collapse toward an X point, since ∇a points toward the X point on the strong magnetic field sides of the X point, and away from it on the weak magnetic field sides. An X point is a saddle point in a. Current elements feel a force on the strong magnetic field sides of the X point which accelerate fluid elements toward it. Eq. (1) shows that for small μ, the field lines of $\underset{\sim}{B}$ will be dragged with the fluid element. They will be stretched in the process, raising the local value of j. This obviously is a self-enhancing effect, and is not compensated by the fact that fluid elements are simultaneously being accelerated away from the X point at the weak magnetic field corners: there are fewer field lines there and they are not in general stretched by the expulsion. Fig. 11 shows the essential orientation of the relevant vectors.

Something similar to this filamentation appears to be visible in Fig. 7 of Orszag and Tang.[1]

The collapse of the current distribution would appear to be limited by the finite μ, which becomes effective at the smaller spatial scales. The local dissipation rate varies as μj^2, which contributes a larger total integral, for a given total current, as that current becomes concentrated into a smaller and smaller area. The collapse ceases when the dissipation becomes great enough to balance the magnetic energy which can be dragged into the region. This is difficult to estimate analytically.

It is significant that the kinetic energy of the dissipative magnetofluid is never more than 0.05% of the magnetic energy. The high ratio of mean square vector potential to magnetic energy (the ratio at $t = 0$ is 0.8836, while the maximum value the ratio can have is 1.0) effectively locks most of the energy into the magnetic field for a long time.

What is not entirely clear is whether the growth we are seeing is a linear instability of the kind proposed by Furth, Killeen, and Rosenbluth[5,6,7] or a nonlinear coalescence involving the interaction between perturbed current distributions. It seems clear that the saturation mechanism is highly nonlinear.

ACKNOWLEDGEMENTS

We would like to thank Drs. J. Drake and C. S. Liu each for a valuable discussion, and to Dr. Dennis Bushnell for making the NASA-Langley computational facilities available to us. This work was supported in part by NASA under Research Grants NsG-7416 and NGL-16-001-043, and in part by the U. S. Department of Energy.

Table 1 and Key

	$\mu=\nu=0$			$\mu=\nu=.0025$	
time step	0	2000	4500	2000	5500
ε_B (s.p.)	2.7489	2.552	2.199	2.499	2.216
ε_B	2.7503	2.702	2.6259	2.5018	2.231
ε_v	$8.74\text{x}10^{-3}$	$4.64\text{x}10^{-2}$	$1.29\text{x}10^{-1}$	$2.75\text{x}10^{-4}$	$2.26\text{x}10^{-4}$
J	8.936	16.62	44.78	4.471	3.13
A	2.4307	2.4285	2.4251	2.324	2.153
Ω	.100	12.58	40.52	$2.73\text{x}10^{-2}$	$1.74\text{x}10^{-2}$
k_B	1.06	1.055	1.041	1.037	1.018
k_v	10.7	16.5	17.7	10.5	8.77
k_μ	*	*	*	34.59	31.68
k_ν	*	*	*	9.67	8.64

ε_B = total magnetic energy

ε_B (s.p.) = magnetic energy in "sheet pinch modes"

ε_v = total kinetic energy

J = total mean square current

A = total mean square vector potential

Ω = total mean square vorticity (enstrophy)

$k_B^2 = \varepsilon_B/A$ = (mean magnetic wave no.)^2

$k_v^2 = \Omega/\varepsilon_v$ = (mean kinetic wave no.)^2

$k_\mu = |(d\varepsilon_B/dt)\mu^{-3}|^{1/4}$ = magnetic dissipation wave no.

$k_\nu = |(d\varepsilon_v/dt)\nu^{-3}|^{1/4}$ = kinetic dissipation wave no.

Time step size $\Delta t = (256)^{-1}$, both runs; (becomes $(512)^{-1}$, late in dissipative run).

Figure Captions

Fig. 1. Magnetic field lines (contours of constant vector potential a) for the initial sheet pinch configuration. $\underset{\sim}{B}$ passes through zero at each sheet.

Fig. 2. Time evolution, for the non-dissipative run, of non-sheet pinch magnetic energy ε_B (nsp), mean square current J, mean square vector potential A, and kinetic energy ε_v.

Fig. 3a. Contours of constant current density j for non-dissipative (and dissipative) run at t = 0. Compare with Fig. 1.

Fig. 3b. Contours of constant j at 2000 time steps, for $\mu=\nu=0$.

Fig. 3c. Contours of constant j at 4500 time steps, for $\mu=\nu=0$.

Fig. 4a. Vector potential contours ($\mu=\nu=0$) after 2000 time steps.

Fig. 4b. Vector potential contours ($\mu=\nu=0$) after 4500 time steps.

Fig. 5a. Contours of constant stream function (velocity streamlines) for $\mu=\nu=0$ at t = 0. The velocity field shown is essentially infinitesimal random noise.

Fig. 5b. Stream function contours for $\mu=\nu=0$ at 2000 time steps.

Fig. 5c. Stream function contours for $\mu=\nu=0$ at 4500 time steps.

Fig. 6a,b. Modal $\underset{\sim}{B}$ and $\underset{\sim}{v}$ energy spectra (directionally averaged over all $\underset{\sim}{k}$ vectors corresponding to a given k^2), averaged over time steps 4500 to 5000, for $\mu=\nu=0$.

Fig. 7. Time history of ε_B(nsp), J, A, ε_v for $\mu=\nu=0.0025$ run. The same quantities for zero dissipation, starting from the same initial conditions, are plotted in Fig. 2.

Fig. 8a. Constant j contours at 2000 time steps for $\mu=\nu=0.0025$.

Fig. 8b. Constant j contours at 5500 time steps for $\mu=\nu=0.0025$.

Fig. 9a. Constant a contours at 2000 time steps for $\mu=\nu=0.0025$.

Fig. 9b. Constant a contours at 5500 time steps for $\mu=\nu=0.0025$.

Fig. 10a. Stream function contours at 2000 time steps for $\mu=\nu=0.0025$. [The t = 0 contours are shown in Fig. 5a.]

Fig. 10b. Stream function contours at 5500 time steps for $\mu=\nu=.0025$.

Fig. 11. Schematic diagram showing the direction of the accelerations, for j > 0, in the neighborhood of a magnetic X point. Field lines dragged toward the X point from the high field sides are stretched.

Footnotes

[1] S. A. Orszag, Stud. Appl. Math. 50, 293 (1971).
G. S. Patterson and S. A. Orszag, Phys. Fluids 14, 2358 (1971).
S. A. Orszag and C. -M. Tang, J. Fluid Mech. 90, 129 (1979).

[2] D. Fyfe and D. Montgomery, J. Plasma Phys. 16, 181 (1976).
D. Fyfe, G. Joyce, and D. Montgomery, ibid. 17, 317 (1977).
D. Fyfe, D. Montgomery, and G. Joyce, ibid. 17, 369 (1977).

[3] T. Sato and T. Hayashi, Phys. Fluids 22, 1189 (1979). (This paper also contains a rather more complete bibliography of the astrophysical background of this problem than we present here.)

[4] W. H. Matthaeus and D. Montgomery, submitted to Phys. Fluids, 1979.

[5] H. P. Furth, J. Killeen, and M. N. Rosenbluth, Phys. Fluids 6, 459 (1963).

[6] E. M. Barston, Phys. Fluids 12, 2162 (1969).

[7] J. F. Drake, N. T. Gladd, C. S. Liu, and C. L. Chang, "Microtearing Modes and Anomalous Transport in Tokamaks", University of Maryland Plasma Preprint PL #79-026 (April, 1979).

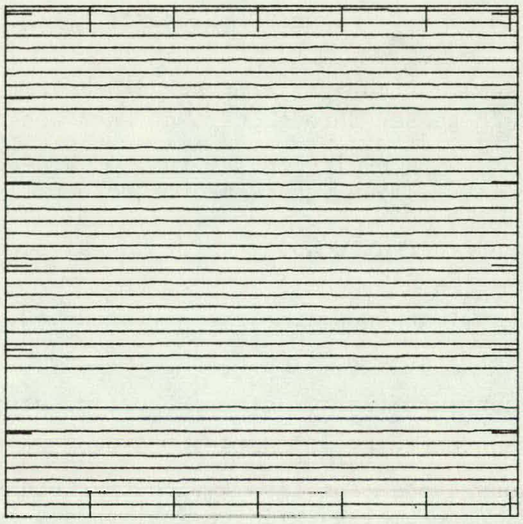


Fig. 1

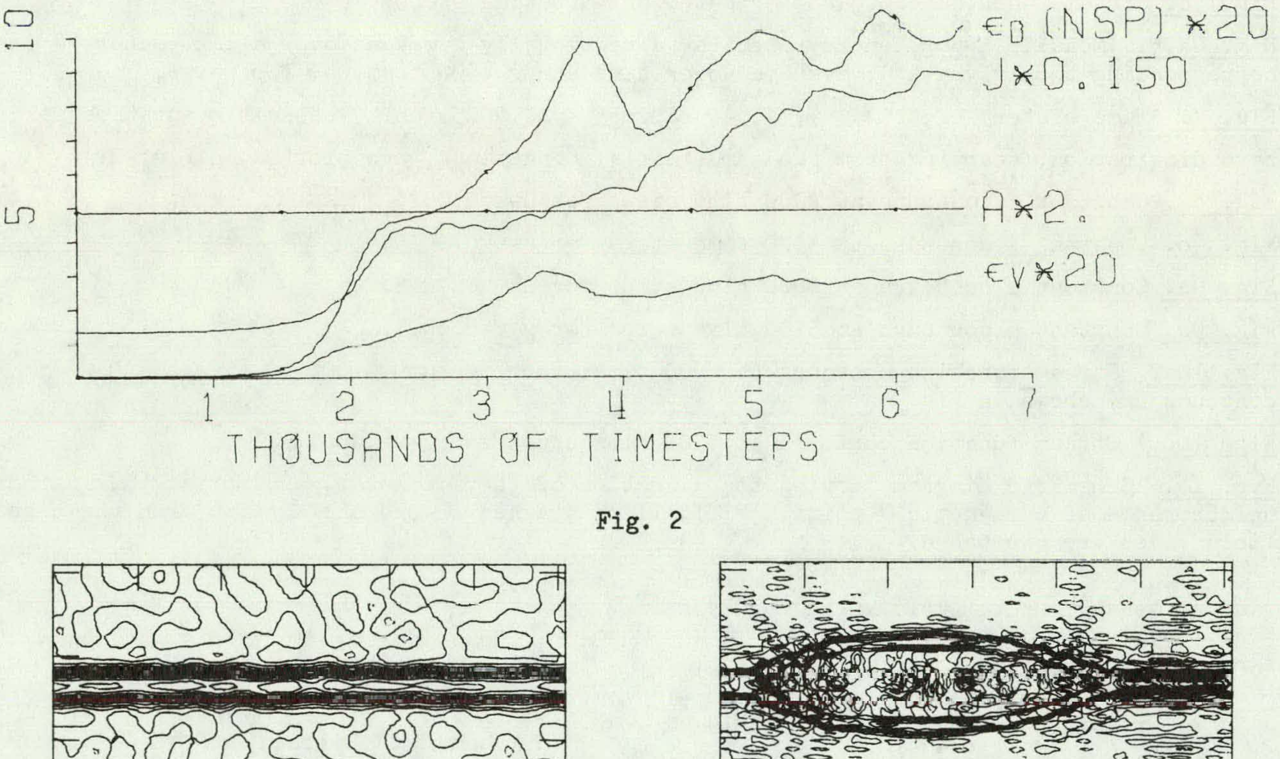


Fig. 2

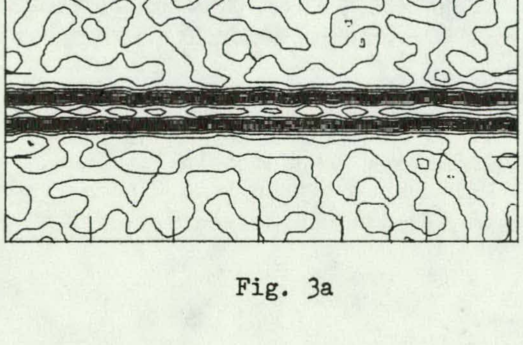


Fig. 3a

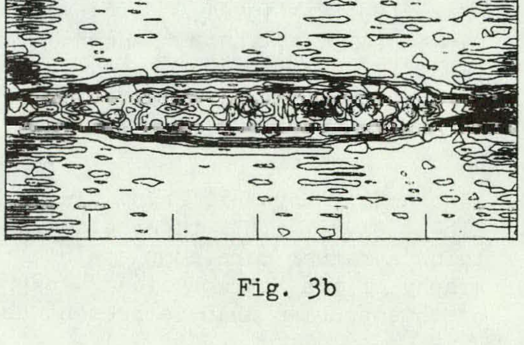


Fig. 3b

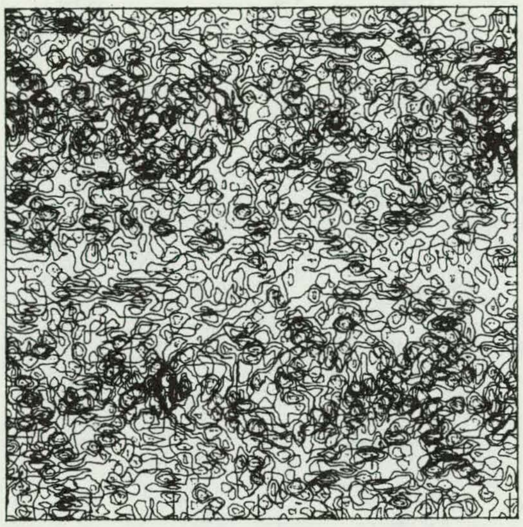


Fig. 3c

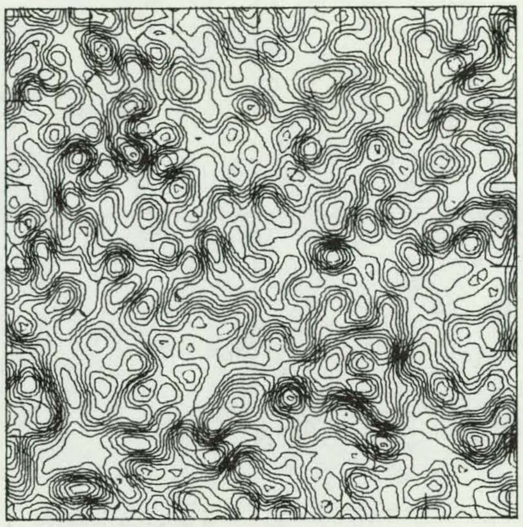


Fig. 5a

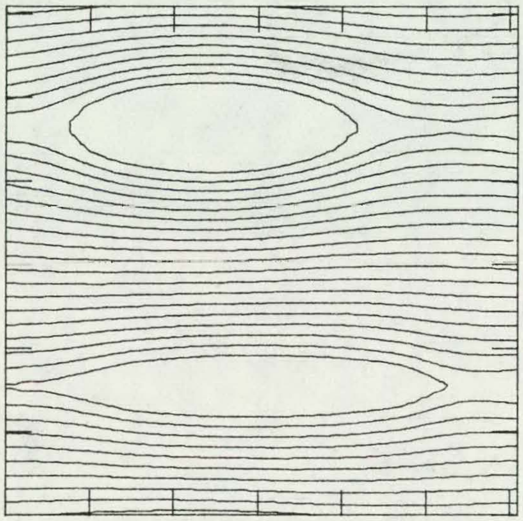


Fig. 4a

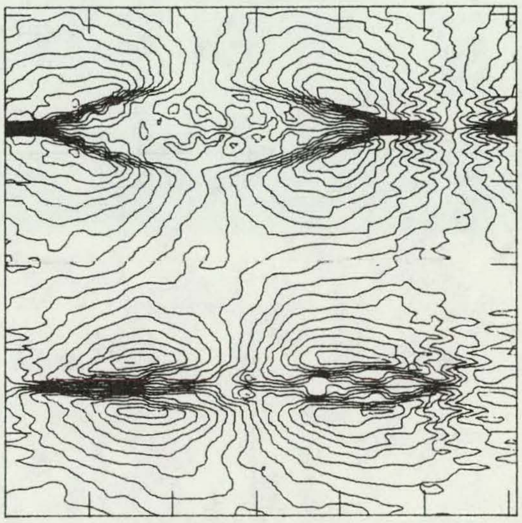


Fig. 5b

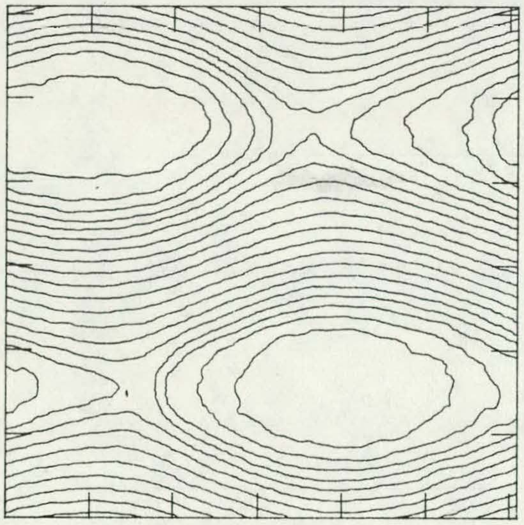


Fig. 4b

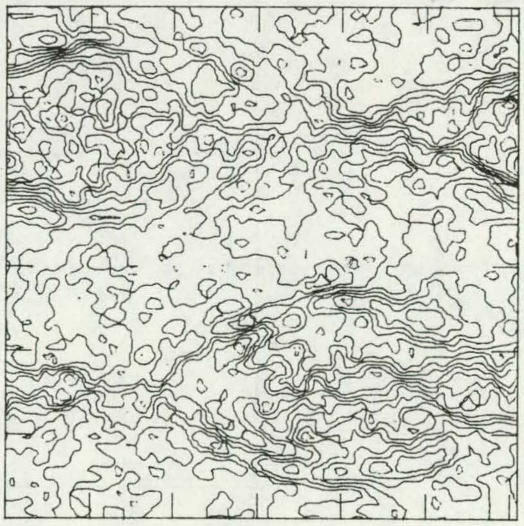


Fig. 5c

Fig. 6a

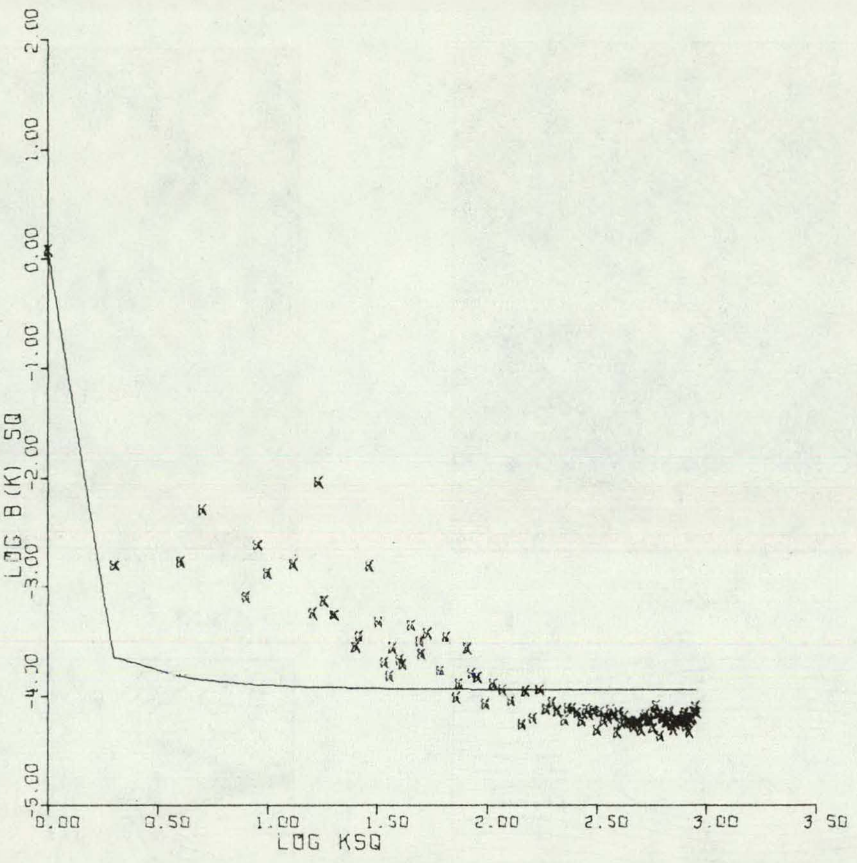

LOG B(K) SQ
2.00
1.00
0.00
-1.00
-2.00
-3.00
-4.00
-5.00
0.00
0.50
1.00
1.50
2.00
2.50
3.00
3.50
LOG KSQ

Fig. 6b

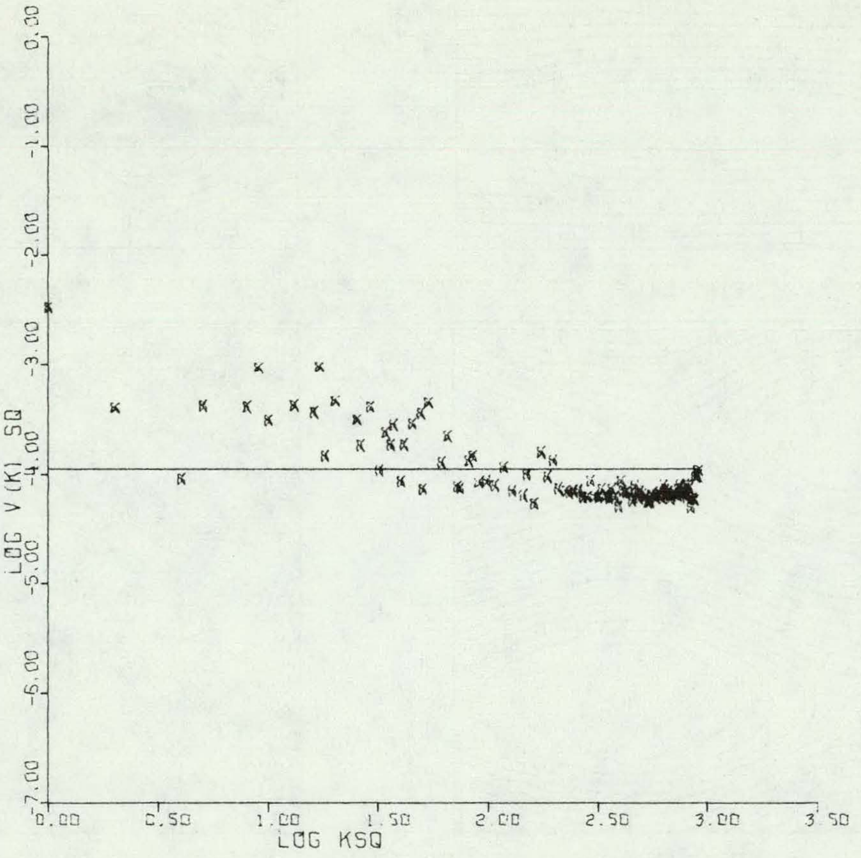
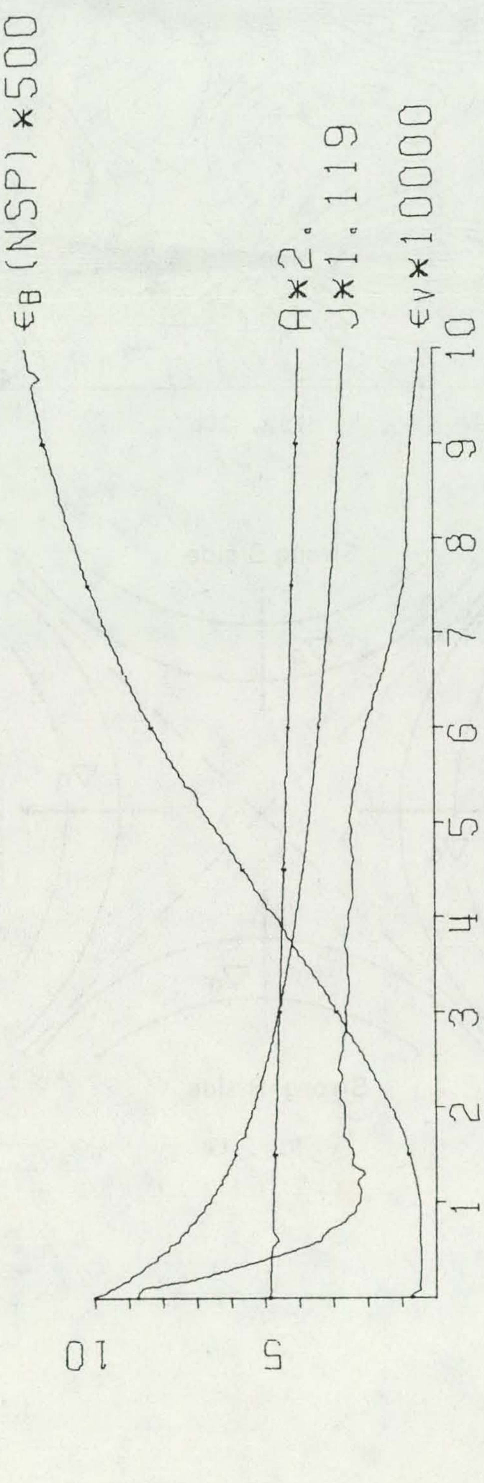

LOG V(K) SQ
0.00
-1.00
-2.00
-3.00
-4.00
-5.00
-6.00
-7.00
0.00
0.50
1.00
1.50
2.00
2.50
3.00
3.50
LOG KSQ

Fig. 7



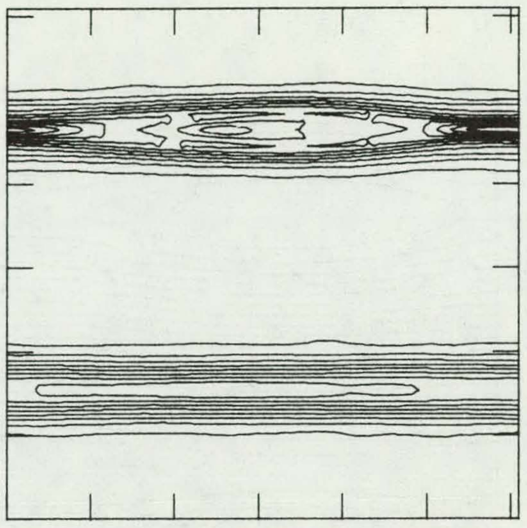


Fig. 8a

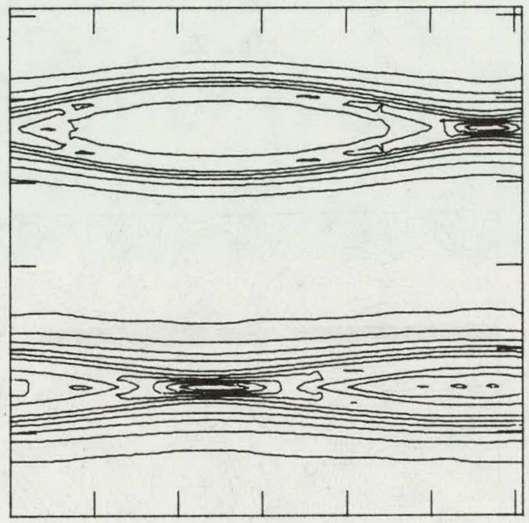


Fig. 8b

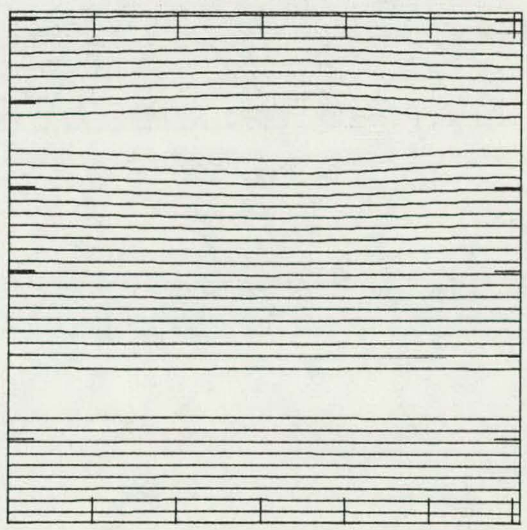


Fig. 9a

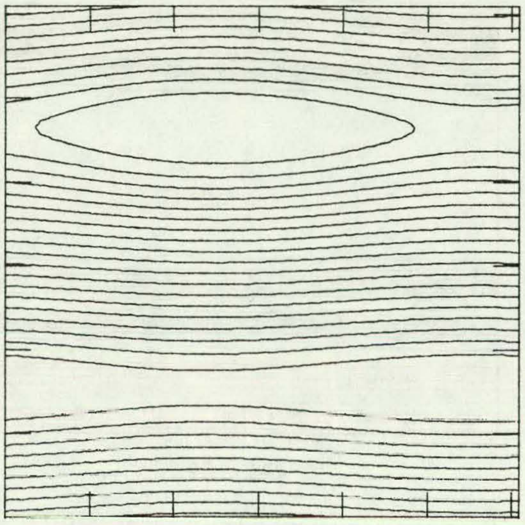


Fig. 9b

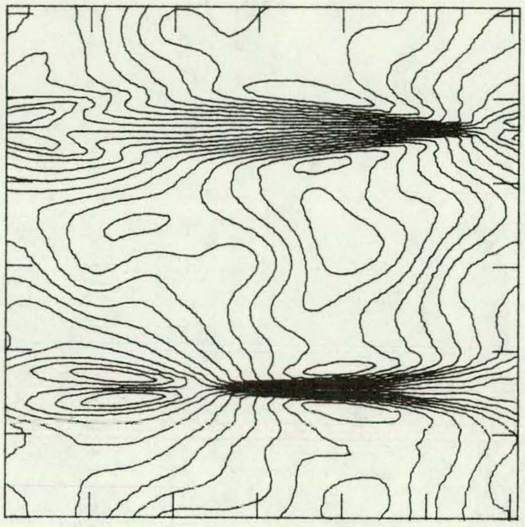


Fig. 10b

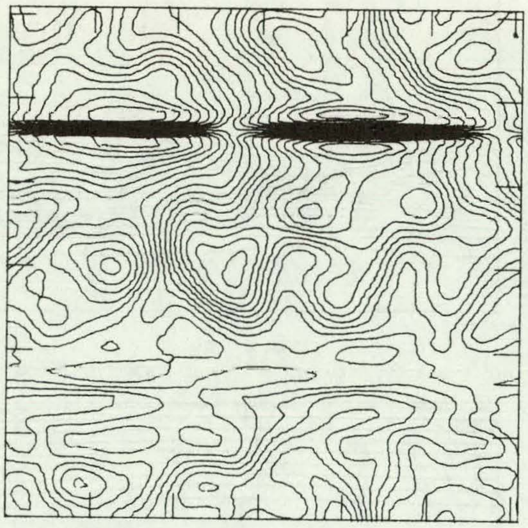


Fig. 10a

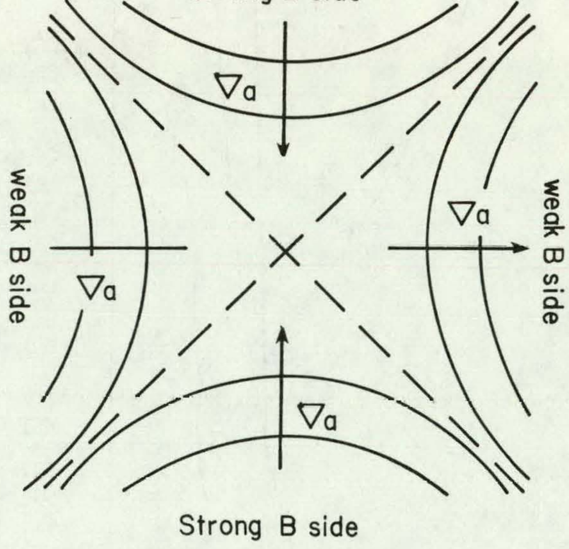


Fig. 11

NUMERICAL SOLUTION OF THE 3-D NAVIER-STOKES EQUATIONS ON THE CRAY-1 COMPUTER

J. S. Shang,* P. G. Buning,** W. L. Hankey*
M. C. Wirth,* D. A. Calahan** and W. Ames**
*Air Force Flight Dynamics Laboratory
**University of Michigan

ABSTRACT

A three-dimensional, time dependent Navier-Stokes code using MacCormack's explicit scheme has been vectorized for the CRAY-1 computer. Computations were performed for a turbulent, transonic, normal shock wave boundary layer interaction in a wind tunnel diffuser. The vectorized three-dimensional Navier-Stokes code on the CRAY-1 computer achieved a speed of 128 times that of the original scalar code processed by a CYBER 74 computer. The vectorized version of the code outperforms the scalar code on the CRAY computer by a factor of 8.13. A comparison between the experimental data and the numerical simulation is also made.

NOMENCLATURE

c	Speed of Sound
Def	Deformation Tensor
e	Specific Internal Energy $c_v T + (u^2 + v^2 + w^2)/2$
$\bar{F}, \bar{G}, \bar{H}$	Vector Fluxes, Equation (15)
L_ξ, L_η, L_ζ	Differencing Operator
M	Mach Number
P	Static Pressure
$\dot{q}$	Rate of Heat Transfer
R_{ey}	Reynolds Number Based on Running Length $\rho_\infty u_\infty x/\mu_\infty$
T	Static Temperature
t	time
$\bar{U}$	Dependent Variables in Vector Form $(\rho, \rho u, \rho v, \rho w, \rho e)$
$\bar{u}$	Velocity Vector
u, v, w	Velocity Components in Cartesian Frame
x, y, z	Coordinates in Cartesian Frame
ξ, η, ζ	Transformed Coordinate System, Equation (14)
ρ	Density
$\bar{\bar{\tau}}$	Stress Tensor

INTRODUCTION

In the past decade, computational fluid dynamics has become firmly established as a credible tool for aerodynamics research[1,2]. Aided by some rather crude and heuristic turbulence models, success has been achieved even for complex turbulent flows[3-8]. In spite of all these convincing demonstrations, the objective of a wide application of computational fluid dynamics in engineering design has yet to be achieved. The basic limitation is in cost effectiveness. A lower cost and systematic methodology needs to be developed[9].

The present analysis addresses one of the key objectives in obtaining efficient numerical processing. To achieve this objective, two approaches seen obvious; either develop special algorithms designed for a particular category of problems according to the laws of physics or utilize an improved computer. In the case of special algorithms, a better understanding of the generic structure of the flow field is required. In general, these attempts have been successful and have achieved an order of magnitude improvement in computing speed. On the other hand, a class of computers designed for scientific computations; the CRAY-1, STAR 100 and ILLIAC IV among others, has become available. The most significant advance in computer hardware related to computational fluid dynamics is the vector processor which permits a vector to be processed at an exceptional speed. This option gives a new perspective; i.e., a drastic reduction in computing time[10, 11, 12].

A three-dimensional time dependent Navier-Stokes code using MacCormack's explicit scheme[13] has been vectorized for the CRAY-1 computer. The selection of this particular finite differencing scheme is based on its past ability to perform a large number of successful bench mark runs[2-7], its proven shock-capturing capability, and the inherent simplicity of the basic algorithms. The Cray-1 computer was chosen because at the present time, among all the available general purpose scientific processors, it provides the highest potential floating point computation rate in both the scalar and the vector mode[14]. The combination of the selected algorithm and the CRAY-1 computer provides a bench mark for future development and a tool for current engineering evalution.

The problem selected for evaluating the CRAY-1 performance was the experimental investigation of Abbiss[15,16] of a three-dimensional interaction of a normal shock with a turbulent boundary layer in a square wind tunnel diffuser at a Reynolds number of thirty million and Mach number of 1.51. The primary purpose of the paper is to determine the computational speed of the code, although a comparison with experimental data is presented to demonstrate the validity of the solution.

GOVERNING EQUATIONS

The time dependent, three dimensional compressible Navier-Stokes equations in mass-averaged variables can be given as

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \bar{u}) = 0 \qquad (1)$$

$$\frac{\partial \rho \bar{u}}{\partial t} + \nabla \cdot (\rho \bar{u}\bar{u} - \bar{\bar{\tau}}) = 0 \qquad (2)$$

$$\frac{\partial \rho e}{\partial t} + \nabla \cdot (\rho e \bar{u} - \bar{u} \cdot \bar{\bar{\tau}} + \bar{q}) = 0 \qquad (3)$$

The turbulent closure of the present analysis is accomplished through an eddy viscosity model. The effective thermal conductivity is also defined by the turbulent Prandtl number ($Pr_t = 0.9$). The equation of state, Sutherland's viscosity law and assigned molecular Prandtl number (0.73) formally close the system of governing equations.

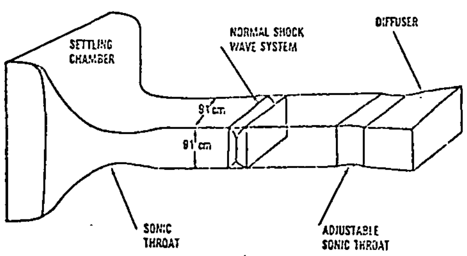


Figure 1. Flow Field Schematic

Since the wind tunnel flow field consisted of four symmetrical quadrants, only a single quadrant was computed. The boundaries of the computational domain contain two intersecting wind tunnel walls and two planes of symmetry for which the associated boundary conditions are straight forward (Figure 1). In order to develop upstream conditions equivalent to the experiment a separate computation is initiated with a free stream condition and permitted to develop a three-dimensional boundary layer along the corner region until the boundary layer duplicates the experimental observation (δ = 4.0 cm, x = 316 cm)[15]. Then, the computed flow field at this streamwise location is imposed as the upsteam condition for the interaction computation. On the wind tunnel walls, the boundary conditions are no-slip for the velocity components and a constant surface temperature. The wind tunnel wall pressure is obtained by satisfying the momentum equation at the solid surface. On the planes of symmetry, the symmetrical boundary conditions are given for all dependent variables. The normal shock wave across the wind tunnel is then specified according to the Rankine-Hugoniot conditions. The far downstream boundary condition is the well known no-change condition. In summary:

<u>INITIAL CONDITION:</u>

$$\bar{U}(0, \xi, \eta, \zeta) = \bar{U}_\infty \qquad (9)$$

<u>UPSTREAM CONDITION:</u>

$$\bar{U}(t, 0, \eta, \zeta) = \bar{U}_\infty \qquad (10)$$

<u>DOWNSTREAM</u> :

$$\left.\frac{\partial \bar{U}}{\partial x}\right|_{x \to x_L} = 0 \qquad (11)$$

<u>ON PLANES OF SYMMETRY:</u>

$$\left.\frac{\partial \bar{U}}{\partial y}\right|_{y = y_L} = 0 \text{ and } \left.\frac{\partial \bar{U}}{\partial z}\right|_{z = z_L} = 0 \qquad (12)$$

<u>ON WIND TUNNEL WALL:</u>

$$u = v = w = 0 \qquad (13a)$$

$$T_w = 313.79°\text{K} \quad \text{at } y,\ z = 0 \qquad (13b)$$

$$\nabla \cdot \bar{\bar{\tau}} = 0$$

A coordinate system transformation is introduced to improve the numerical resolution in the viscous dominated region.

$$\xi = x/x_L \qquad (14a)$$

$$\eta = 1/k \ln[1 + (e^k - 1)\, y/y_L] \qquad (14b)$$

$$\zeta = 1/k \ln[1 + (e^k - 1)\, z/z_L] \qquad (14c)$$

The governing equations in the transformed space are of the following form:

$$\frac{\partial \bar{U}}{\partial t} + \xi_x \frac{\partial \bar{F}}{\partial \xi} + \sum_i \eta_{x_i} \frac{\partial \bar{G}}{\partial \eta} + \sum_i \zeta_{x_i} \frac{\partial \bar{H}}{\partial \zeta} = 0 \tag{15}$$

where ξ_x, η_y and ζ_z are the metrics of the coordinate transformation. The definition of the conventional flux vectors F, G, and H can be found in Ref. 7.

NUMERICAL PROCEDURE AND DATA STRUCTURE

The basic numerical method is the time-split or factorized scheme originated by MacCormack. The finite difference formulation in terms of the difference operator can be expressed as

$$\bar{U}^{n+2} = \sum_n L_\zeta\left(\frac{\Delta t}{2n}\right) \sum_m L_\eta\left(\frac{\Delta t}{2m}\right) L_\xi(\Delta t)$$

$$\sum_m L_\eta\left(\frac{\Delta t}{2m}\right) \sum_n L_\zeta\left(\frac{\Delta t}{2n}\right) \tag{16}$$

Each difference operator contains a predictor and corrector. During a specific numerical sweep, the flux vectors are approximated by a central, forward, and backward differencing scheme in such a fashion that after a complete cycle of the predictor and corrector operations all the derivatives are effectively approximated by a central differencing scheme. A graphic representation of these operations is given by Figure 2.

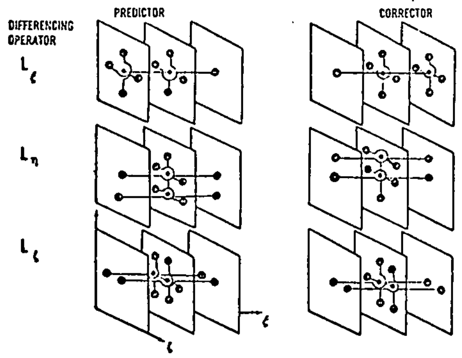


Figure 2. Grid Points Involved in the Time Step Sweep

When investigating flows with strong shock waves, it is necessary to employ numerical damping in a shock-capturing scheme. Fourth-order pressure damping was utilized which generates an artificial viscosity-like term.[17]

$$\Delta t \Delta \xi_i^3 \frac{\partial}{\partial \xi_i}\left[\frac{|u_i| + c}{4p} \frac{\partial^2 p}{\partial \xi_i^2}\right] \frac{\partial \bar{U}}{\partial \xi_i} \quad i = 1,2,3$$

The approximation of second order central differencing for the corrector step required additional grid point information beyond the immediately adjacent planes. The damping terms, however, are effective only in the presence of shock waves where the numerical resolution is degraded.

From the symmetric differencing operator sequence of predictor and corrector steps, one detects that the dependent variables in the predictor level can be completely eliminated by retaining only the three cyclic pages currently in use (Figure 3). For a flow field requiring a large amount of data storage, this reduction in memory requirement is substantial. Meanwhile, the paging process is reduced from two sweeps to one. The predictor and corrector sequence is performed within one sweep by overlapping the corrector operation during one fractional time step.

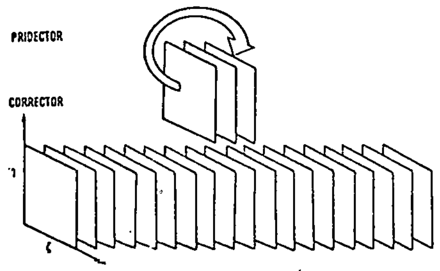


Figure 3. Data Storage and Data Flow Diagram

Once the planar or page storage is adopted, the vector length can be determined. Separate vectors are constructed for η and ζ directions, yielding vector lengths approximately equal to the number of grid points in each direction. In order to keep all solutions in the same page (η - ζ plane), the streamwise sweep (ξ sweep) is vectorized in the ζ direction.

For the present problem, the computational domain with the dimension of 356.3cm x 45.5cm x 45.5cm is partitioned into two streamwise sections of 64 pages each. Every page contains 33 x 33 grid points in η and ζ coordinates respectively. The problem is solved in two steps. The first computational section generates a three-dimensional boundary layer over a corner which becomes the in-flow boundary condition for the following shock-boundary interaction domain. Both contain 64 x 33 x 33 grid points, but a finer streamwise mesh spacing Δx = 1.27 cm was used for the interaction zone to gain a finer numerical resolution of the shock-boundary layer interaction. The ratio between the fine and coarse streamwise grid spacing is 0.3063 of the local boundary-layer thickness (4.0cm)[15]. The cross flow plane grid-point distribution, however, remains identifical between the two overlapping segments. The memory requirement for each is about 0.545 million words.

The numerical solution is considered at its steady state asymptote when the maximum difference between two consecutive time levels of the static pressure in the strong interacting zone is less than 0.2 percent. In the leading computational domain the convergence criterion is established similarly but is based on the velocity profiles instead of pressure.

TIMING RESULTS

A portion of the present effort is aimed at making internal comparisons of the relative times for various types of functional unit processing and memory loading (I/O) for the vectorized code. A knowledge of relative time expenditure information is important to provide some insight into the program execution rate. Although this type of data is code dependent, the present example is deemed typical of a large class of Navier-Stokes solvers. The timing information is measured by vector operation counts[11] and shown in Figure 4. It is obvious that the relative usage of the memory path and functional units is dominated by memory loadings (34.6%) and floating point multiplication (33.3%). Within the functional units, the relative usage of the floating point addition and multiplication has the ratio of two to three. The relative usage of the reciprocal approximation is extremely rare, i.e. less the 2%. In spite of the high percentage of memory loading, a portion of the vectorized Fortran code has achieved an execution rate of 42.9 MFLOPS[11]. Further improvements still can be made either in Fortran or assembly language versions of the present code. However, we feel an overall execution rate greater than 60 MFLOPS on this size problem is unlikely.

A basic dilemma exists for the comparative investigation; namely in the process of vectorization significant changes were made either on the amount of computation performed or on the number of subroutine calls made. The final vectorized program usually bears little resemblence to the original scalar code[7, 11]. Substantial improvement in performance of the vectorized code on a scalar machine has also been reported. However, this improvement in performance can be considered as a contribution due to the vectorization process.

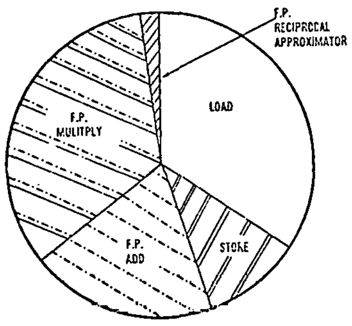


Figure 4. Vector Operation Counts in Percentage

In order to perform the comparative study, a criterion must be established. The ultimate evaluation of data processing rate is the computing time. The completely duplicated computations for an identical fluid mechanics problem are usually prohibited by the incore memory and the indexing limitations for various processors. Therefore, one has to accept the rate of data processing as the criterion. The rate of data processing is commonly defined as

RDP = CPU Time/(Total Number of Grid Points x Total Number of Iterations)

The particular rate of data processing is most suitable for numerical programs with similar algorithms and convergence rate. If the ratio between field grid points and boundary points can be maintained between two programs then the comparison is particularly meaningful.

In Table 1, the comparison of timing results between the scalar code and vectorized code on the CRAY-1 is presented.

Table 1

The Comparison of Scalar and Vector Processing on CRAY-1

VERSION OF CODE	RDP(Sec/Pts, ITERATIONS)
Scalar	4.761×10^{-4}
Vector	4.861×10^{-5}

The vectorized program outperforms the original scalar code by a factor of 8.13. In Table 2, the timing results of the scalar code and vectorized code performance for four different computers are given.

Table 2

Comparative Timing Results

COMPUTER			$\frac{(RDP)_{CYBER\ 74}}{RDP}$
CYBER 74	Scalar	7.48×10^{-3}	1.0
CDC 7600	Scalar	1.45×10^{-3}	5.2
CRAY-1	Scalar	4.76×10^{-4}	15.7
CRAY-1	Vector	5.86×10^{-5}	127.7
CRAY-1	Assembly	5.19×10^{-5}	144.2

A brief description of each running condition for which the timing results were obtained may help with the interpretion of the data. The computations conducted on CYBER 74 and CDC 7600 with a grid point system of (17 x 33 x 33) were performed in the early phase of the present task[7]. On the CYBER 74 computer the data storage problem was overcome by a data manager subroutine in conjunction with a random access disk file. The computation carried out on CDC 7600 used large core memory for all the dependent variables. The I/O requirement is substantial, particularly for the computation performed on the CYBER 74.

FORTRAN VS. ASSEMBLY LANGUAGE

The multiple functional units and memory hieracrchy of the CRAY-1 can be difficult for the Fortran compiler (CFT) to manage efficiently. Consequently, CRAY Assembly Language (CAL) versions of a number of subroutines which account for up to 78% of the computation time were written with the aid of a simulator [18]. These kernels were also vectorized in Fortran with the CRAY-1 architecture and compiler features in mind; however, non-ANSI standard utility functions [19] and unusual Fortran constructs [20] were not employed. The principle timing results follow.

1) Among 9 kernels, assembly language speedups ranged from 11% to 29% with vector lengths of 33 (= a grid dimension).
2) An overall speedup of 14.2% was achieved (Table 2), including the common 22% Fortran.
3) A detailed simulator-produced evaluation of a subroutine which accounts for ≃ 20% of the total computation time is given in [21]. The execution rate of ≃ 50 MFLOPS is 1/3 of the maximum practical rate of the processor. However, the memory path is busy 70% of the time for the Fortran code for a vector length of 63, and up to 90% for the CAL code, indicating the memory bound nature of the algorithm on the CRAY-1. Indeed, the 90% busy time is viewed as an excellent indicator of the optimality of the CAL code.

A more detailed comparative study of this code is given in [21].

COMPARISONS WITH EXPERIMENTAL DATA

In Figure 5, several velocity profiles across the wind tunnel at a Reynolds number of 3.0×10^7 are presented. This location represents the flow field condition at the end of the leading segment of the computational domain which is also the upstream condition for the following interaction zone. The present results agree reasonably well with the data of Seddon[16]. The data, however, were collected at a Reynolds number one decade lower than the present condition and at a slightly different Mach number (1.47 v.s. 1.51). At the range of Reynolds numbers considered, the Reynolds number dependence should be scaled out by the boundary layer thickness. An independent boundary-layer calculation using the exact simulated condition was performed that verified this contention. It was found that the difference in magnitude of velocity is a few percent. The present result underpredicts the measured boundary layer thickness[15] by about eight percent

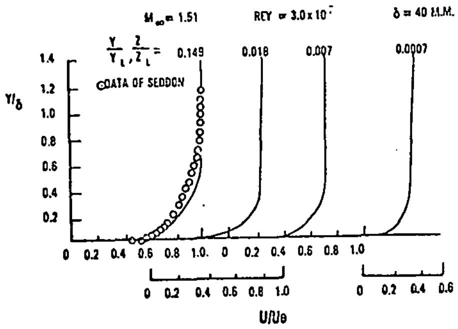


Figure 5. Velocity Profiles Along the Tunnel Wall

A direct comparison of several velocity distributions between the data of Abbiss et al[15] and the present calculation is presented in Figure 6 for the interaction region. The data are displayed for fixed x/δ and y coordinates away from the corner domain. The coordinate x is taken in the streamwise direction along the tunnel floor and y normal to the floor. Excellent agreement between the data and calculation is observed for the regions either deeply imbedded within the boundary layer or completely contained in the inviscid domain. The maximum discrepancy between data and calculation is in the lambda wave structure. The maximum desparity between data and calculations is about 10 percent.

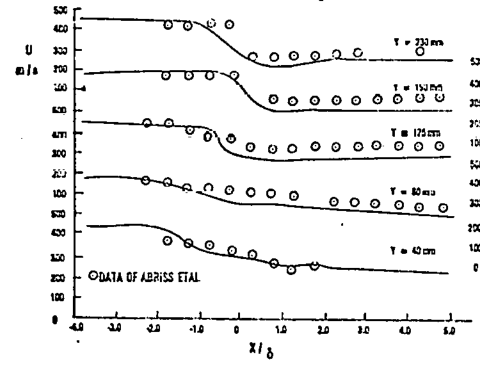


Figure 6. Comparison of the Flow Field Velocity in the Interactive Region

In Figure 7 the Mach number contour is presented in an attempt to compare with the flow field structure given by Abbiss et al[15] in Figure 8. The bifurcation of the normal shock wave is clearly indicated. The calculation nearly duplicates all of primary features of the experimental observation. However, a difference can be discerned in the dimension of the embedded supersonic zone between the experimental observation and calculation. The local supersonic zone emanates from the expansion due to the total pressure difference between the normal shock and the lambda shock structure and the rapid change in the displacement surface. A few percent disparity in predicting the magnitude of velocity lead to the distinguishable discrepancy in the definition of the embedded supersonic zone. A similar observation may be made for the work of Shea[22] in his investigation of the two-dimensional normal-shock wave turbulent boundary layer interaction.

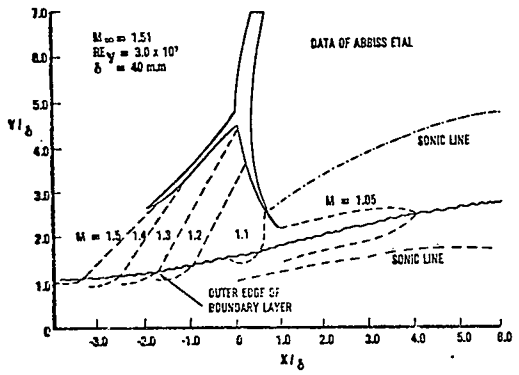


Figure 7. Experimentally Measured Flow Field Structure in the Plane of Symmetry

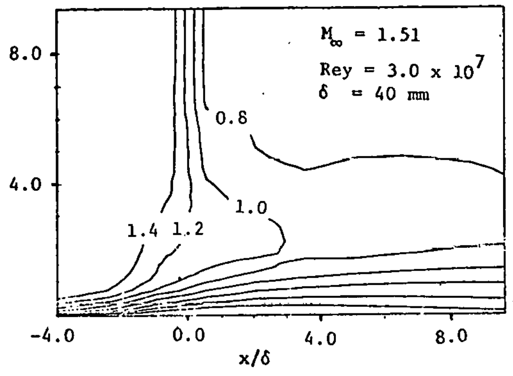


Figure 8. Computed Number Contour in the Plane of Symmetry

In Figure 9, the velocity distribution parallel to the wind tunnel side is given. A reverse flow is observed beneath the lambda shock wave system. The separated flow region begins about three boundary-layer thickness upstream of the normal shock and terminates at five boundary layer thickness downstream. The length of the separated domain is similar to the measurement of Seddon[16] and the numerical simulation by Shea[22].

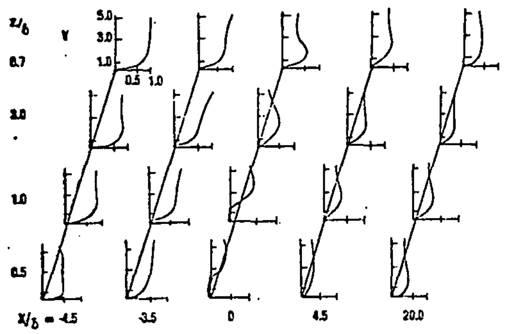


Figure 9. Computed Velocity Field in the Interaction Region

The entire flow field structure is presented in Figure 10 in terms of density contours at various streamwise locations. The shear layer over the corner region, the strong inviscid-viscous domain, and the subsequent readjustment of the flow field are easily detectable. A clear indication of substantial growth of the shear layer over the wind tunnel wall is also obvious.

CONCLUSIONS

A three-dimensional time dependent Navier-Stokes code using MacCormack's explicit scheme has been vectorized for the CRAY-1 computer achieved a speed of 128 time that of the original scalar code processed by a CYBER 74 computer. The vectorized code outperforms the scalar code on the CRAY-1 computer by a factor of 8.13.

The numerical simulation for a turbulent, transonic, normal shock-wave boundary-layer interaction in a wind tunnel has been successfully performed using a total 139,400 grid points. The numerical result indicates sufficient resolution for engineering purposes. Additional increase in speed by up to an order of magnitude through algorithm requirement also seems attainable.

ACKNOWLEDGEMENT

The authors wish to acknowledge the assistance of S. Arya and E. Sesek of the University of Michigan in preparation of the CRAY-1 program. This work was prepared in part under the auspices of Grant AF AFOSR Grant 75-2812. The authors also wish to express their appreciation to Cray Research, Inc. and Lawrence Livermore Laboratory for the use of their computer facility.

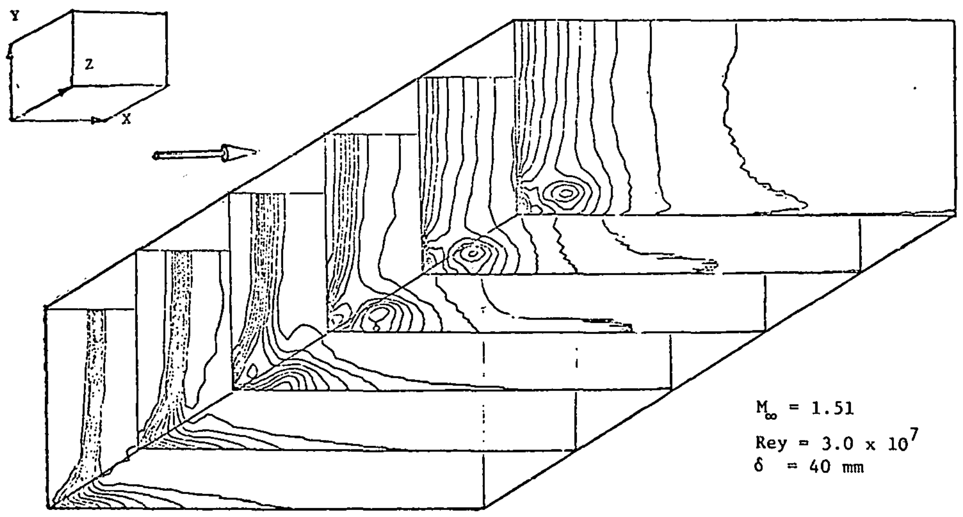


Figure 10. Perspective View of Density Contours

REFERENCES

1. Chapman, D. R., Drydent Lectureship in Research Computational Aerodynamics Development and Outlook, AIAA Paper 79-0129, January 1979.

2. Peyret, R. and Viviand, H. "Computation of Viscous Compressible Flows Based on the Navier-Stokes Equations," AGARDograph, No. 212, September 1975.

3. Knight, D. D., "Numerical Simulation of Realistic High-Speed Inlets Using the Navier-Stokes Equations," AIAA J., Vol. 16, June 1978.

4. Levy, L. L. "Experimental and Computational Steady and Unsteady Transonic Flow About a Thick Airfoil," AIAA J., Vol. 16, June 1978.

5. Mikhail, A. G., Hankey, W. L. and Shang, J. S., "Computation of a Supersonic Flow Past An Axisymmetric Nozzle Boattail with Jet Exhaust," AIAA Paper 78-993, July 1978.

6. Hung, C. M. and MacCormack, R. W. "Numerical Solution of Three-Dimensional Shockwave and Turbulent Boundary-Layer Interaction," AIAA J., Vol. 16, No. 10, October 1978.

7. Shang, J. S., Hankey, W. L. and Petty, J. S., "Numerical Solution of Supersonic Interacting Turbulent Flow Along a Corner," AIAA Paper 78-1210, July 1978.

8. Pulliam, T. H. and Lomax, H., "Simulation of Three-Dimensional Compressible Viscous Flow on the Illiac IV Computer," AIAA Paper 79-0206, January 1979.

9. "Future Computer Requirements for Computational Aerodynamics," A Workshop held at NASA Ames Research Center, Oct 406, 1977, NASA Conference Proceeding 2032.

10. J. S. Shang, Buning, P. G., Hankey, W. L., and Wirth, M. C., "The Performance of a Vectorized 3-D Navier-Stokes Code on the CRAY-1 computer, AIAA Paper 79-1448, 1979.

11. Buning, P. G., "Preliminary Report on the Evaluation of the CRAY-1 as a Numerical Aerodynamic Simulation Process," Presented at AIAA 3rd Computational Fluid Dynamics Conference, Open Forum, June 1977.

12. Smith, R. E. and Pitts, J. I., "The Solution of the Three-Dimensional Compressible Navier-Stokes Equations on a Vector Computer," Third IMACS International Sympsium on Computer Methods for Partial Differential Equations," June 1979, Lehigh University, PA, and Private Communication.

13. MacCormack, R. W., "Numerical Solutions of the Interactions of a Shock Wave with a Laminar Boundary-Layer," Lecture Notes in Physics, Vol. 8, Springer-Verlag, 1971.

14. Calahan, D. A., "Performance of Linear Algebra Codes on the CRAY-1, "Proceedings SPE Symposium on Reservior Simulation, Denver, CO, 1979.

15. Abiss, J. B., East, L. F., Nash, C. R., Parker, P., Pike, E. R. and Swayer, W. G., "A Study of the Interaction of a Normal Shock-wave and a Turbulent Boundary Layer Using a Laser Anemometer," Royal Aircraft Establishment, England, TR 75151, February 1976.

16. Seddon, J., "The Flow Produced by Interaction of Turbulent Boundary-Layer with a Normal Shock Wave of Strength Sufficient to Cause Separation," Royal Aircraft Establishment, England, Rand M 3502, March 1960.

17. MacCormack, R. W. and Baldwin, B. S., "A Numerical Method for Solving the Navier-Stokes Equations with Application to Shock-Boundary Layer Interactions," AIAA Paper 75-1, January 1975.

18. Orbits, D. A., "A CRAY-1 Simulator," Report #118, Systems Engineering Laboratory Univ. of Michigan, September 1, 1978.

19. CRAY-1 Fortran (CFT) Reference Manual, Pub. #2240009, Cray Research, Inc., 1978.

20. Higbie, Lee, "Speeding Up Fortran (CFT) Programs on the CRAY-1," Technical Note Pub. #2240207, Cray Research, Inc., 1978.

21. Ames, W. G., Arya, S. and Calahan, D. A., "An Evaluation of the Fortran Compiler on the CRAY-1," Report #134, Systems Engineering Laboratory, University of Michigan, October 1, 1979.

22. Shea, J. R., "A Numerical Study of Transonic Normal Shock-Turbulent Boundary Layer Interactions," AIAA Paper 78-1170, July 1978 and Private Communication.

ATTENDEES OF SCIE MEETING

September 12-13, 1979

Larry Ablow
SRI International

Bill Alzheimer
Sandia Laboratory, Livermore

G.W. Anderson
Sandia Laboratory, Livermore

Arvind
Massachusetts Institute of Technology

John Avila
NASA/AMES

Pat Bailey
EPRI

Robert Barton
Lawrence Livermore Laboratory

Marsha Berger
Stanford University

Stanley A. Berger
University of California, Berkeley

Carl Berkowitz
Pacific Northwest Laboratories

Richard Blaine
International Business Machines

David L. Book
Naval Research Laboratory

Jay P. Boris
Naval Research Laboratory

Dennis Brockway
Los Alamos Scientific Laboratory

Ingrid Bucher
Los Alamos Scientific Laboratory

Pieter Buning
NASA/AMES

Bill Buzbee
Los Alamos Scientific Laboratory

D. A. Calahan
University of Michigan

David Ceperley
Lawrence Berkeley Laboratory

Simon W. Chang
Jaycor

W. J. Cody
Argonne National Laboratory

Mark Cotnoir
Systems Development Corporation

Keith L. Derstine
Argonne National Laboratory

Ronald J. Detry
Sandia Laboratory, Albuquerque

Jack Dongarra
Argonne National Laboratory

Paul Dubois
Lawrence Livermore Laboratory

Floyd E. Dunn
Argonne National Laboratory

Jim Edwards
General Electric

Ted Einwohner
Lawrence Livermore Laboratory

Raymond Ellis
Sandia Laboratory, Livermore

Kenneth Eppley
Lawrence Livermore Laboratory

Albert M. Erisman
Boeing Computer Services Company

Vance Farber

Mike Farmwald
Lawrence Livermore Laboratory

Michael D. Feit
Lawrence Livermore Laboratory

Sidney Fernbach
Lawrence Livermore Laboratory

Horace Flatt
International Business Machines

Kirby W. Fong
Lawrence Livermore Laboratory

Bengt Fornberg
California Institute of Technology

Paul Fredrickson
Los Alamos Scientific Laboratory

Alex Friedman
University of California, Berkeley

Verlan K. Gabrielson
Sandia Laboratory, Livermore

Fred Gama-Lobo
Los Alamos Scientific Laboratory

W. Morven Gentleman
University of Waterloo

William E. Gifford III
AFWL, Kirkland

Eric Gilbert
Lawrence Livermore Laboratory

Dick Giroux
Lawrence Livermore Laboratory

Joseph Grcar
University of Illinois

Anne Greenbaum
Lawrence Livermore Laboratory

John Greenstadt
International Business Machines

William Gropp
Stanford University

Louis A. Hageman
Bettis Atomic Power Laboratory

Karen Haskell
Sandia Laboratory, Albuquerque

Gerald Hedstrom
Lawrence Livermore Laboratory

Leland C. Helmle
Informatics - PMI

Richard Hickman
Lawrence Livermore Laboratory

H. Richard Hicks
ORNL

Lee Higbie
Cray Research

Lee E. Hollingsworth
Sandia Laboratory, Albuquerque

Stu Hopkins
NASA/AMES

Alphonse Iacoletti
Sandia Laboratory, Albuquerque

Stan Jensen
Lockheed Research Lab

Thomas L. Jordan
Los Alamos Scientific Laboratory

Ralph G. Jorstad
Boeing Computer Services Company

Lou Just

John Kammerdiener
Los Alamos Scientific Laboratory

Alan Karp
International Business Machines

Michael J. Kascic
Control Data Corporation

Michael J. Keskinen
Naval Research Laboratory

John Killeen
Lawrence Livermore Laboratory

John Kimlinger
Lawrence Livermore Laboratory

David A. Kloc
Air Force Weapons Laboratory, N.M.

Jay Lambiotte
NASA/Langley Research Center

Richard Lancaster
IPC

B. Langdon
Lawrence Livermore Laboratory

Barbara Lasinski
Lawrence Livermore Laboratory

Robert Lee
Lawrence Livermore Laboratory

Charles Leonard
Sandia Laboratory, Livermore

Robert Lyczkowski
Lawrence Livermore Laboratory

Steve McCormick
Colorado State University

Michel McCoy
Lawrence Livermore Laboratory

B. Edward McDonald
Naval Research Laboratory

Brandan McNamara
Lawrence Livermore Laboratory

Lynn D. Maas
Los Alamos Scientific Laboratory

Neil Madsen
Lawrence Livermore Laboratory

Mary-Ann Mahaffy
Los Alamos Scientific Laboratory

J. Manickam
Princeton University

Alan Mankofsky
Cornell University

Thomas Manteuffel
Sandia Laboratory, Livermore

Barry Marden
Sandia Laboratory, Albuquerque

Alex Marusak
Los Alamos Scientific Laboratory

Ken Marx
Sandia Laboratory, Livermore

Lawrence Marx
NASA/Goddard

William Mattheaus
William and Mary

T. C. Michels
Lawrence Livermore Laboratory

Gordon J. Miller
Sandia Laboratory, Livermore

R. H. Miller
University of Chicago

Barry N. Moore
Austin Research Associates

James R. Morris
Lawrence Livermore Laboratory

Paul J. Nikolai
Wright-Patterson AFB

Joseph Oliger
Stanford University

Carl Edward Oliver
Air Force Office Science Research

David Orbits

Arthur Ortega
Sandia Laboratory, Livermore

Sam Paolucci
Sandia Laboratory, Livermore

Merrell Patrick
Duke University

Stu Patterson Jr.
Cray Research

Larry Patzer
Air Force Weapons Lab, N.M.

Charles Pfefferkorn
NASA/AMES

William G. Poole, Jr.
Boeing Computer Services Company

Jeffrey P. Quintenz
Sandia Laboratory, Albuquerque

John Rettberg
International Business Machines

Clifford Rhoades
Air Force Weapons Laboratory, N.M.

Garry Rodrigue
Lawrence Livermore Laboratory

Tim Rudy
Lawrence Livermore Laboratory

Lawrence E. Rudsinski

Steven Sackett
Lawrence Livermore Laboratory

P. G. Saffman
Caltech

Theodore Salvi
Air Force Weapons Laboratory, N.M.

Ahmed Sameh
University of Illinois

James G. Sanderson
Los Alamos Scientific Laboratory

Melvin Scott
Sandia Laboratory, Albuquerque

J. S. Shang
Wright-Patterson AFB

Richard Simari
Air Force Weapons Laboratory, N.M.

Bruce F. Smith
NASA/AMES

Major Leonard Stans
Air Force Weapons Laboratory, N.M.

Kenneth G. Stevens
NASA/AMES

A. Stewart
NASA/AMES

Robert Stoeckly
Mission Research Corporation

William Sutcliffe
Lawrence Livermore Laboratory

Tokihiko Suyehiro
Lawrence Livermore Laboratory

Marvin Theimer
Stanford University

J. W. Thomas
Colorado State University

John Tomlin
NASA/AMES

Samuel Thompson
Sandia Laboratory, Albuquerque

D. S. Trent
Pacific Northwest Laboratories

Phillip J. Trosin
Informatics, Inc.

Carol Tull
Lawrence Livermore Laboratory

Walter H. Vandevender
Sandia Laboratory, Albuquerque

Richard Varga
Kent State University

Doug Vaughan
Lawrence Livermore Laboratory

L. E. Voelker
Sandia Laboratory, Livermore

H. H. Wang
International Business Machines

Osaki Watanuki
University of California, L.A.

V. Watson

David Wexter
System Development Corporation

Paul P. Whalen
Los Alamos Scientific Laboratory

Greg Wojcik
Weidlinger Associates