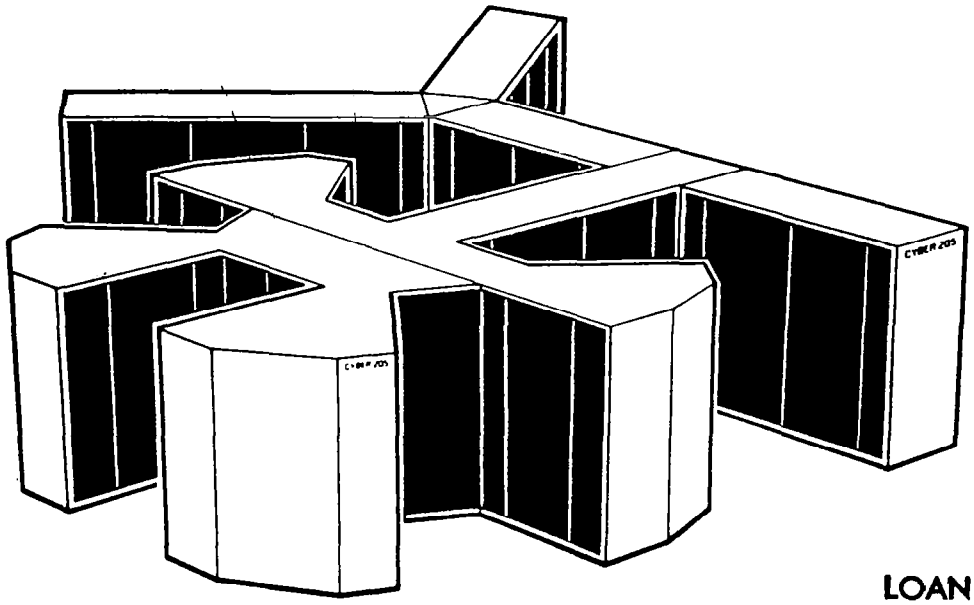
NASA Conference Publication 2295

CYBER 200 Applications Seminar



Proceedings of a seminar held in
Lanham, Maryland
October 10-12, 1983

NASA

NASA Conference Publi

CYBER 200 Applications Seminar

J. Patrick Gary, *Compiler*
Goddard Space Flight Center
Greenbelt, Maryland

Proceedings of a seminar sponsored by
NASA Goddard Space Flight Center and
Control Data Corporation and held in
Lanham, Maryland
October 10-12, 1983

NASA
National Aeronautics
and Space Administration

Scientific and Technical
Information Branch

1984

ACKNOWLEDGMENTS

We would like to publicly acknowledge and gratefully thank the large number of people who contributed their efforts toward making the CYBER 200 Applications Seminar the success that it was. First and foremost among these is Dr. Sidney Fernbach who served as Executive Chairperson in planning and convening the Seminar. Appreciation is further extended to the session chairmen, Drs. Bongiorno, Numrich, and Schneck, and to all of the presenters, including Dr. J. Decker for his after dinner speech entitled, "U.S. Government Study on Supercomputers."

A special note of thanks is offered to the behind-the-scene work of Cheryl Chase and Jean DellAmore for planning and effecting logistic arrangements, and to the receptionist/message support work of Lori Nelson, Brenda Moldawer, and Linda Ficken. Lastly, special appreciation is extended to Brenda Moldawer for the post-seminar compilation of these proceedings.

J. Patrick Gary
GSFC Host

John Zelenka
CDC Host

PREFACE

The CYBER 200 Applications Seminar, held on October 10-12, 1983, in Lanham, Maryland, under the sponsorship of NASA/Goddard Space Flight Center and Control Data Corporation, is the second of its kind. These proceedings comprise the majority of the papers presented at the meeting. Papers for the seminar were selected on the basis of showing a broad distribution of applications for which the CYBER 200 may be well suited. These ranged from problems in meteorology to problems in economics. A breakdown of the disciplines represented is shown below. Some of the papers actually could fall in more than one category, but only one is indicated for each.

	Papers
Meteorology/Oceanography	5
Chemistry	4
Math Algorithms for 205	3
Fluid Dynamics	3
Monte Carlo Methods	3
Petroleum	2
Electronic Circuit Simulation	1
Biochemistry	1
Lattice Gauge Theory	1
Economics	1
Ray Tracing	1

In the first seminar held in August 1982, it was evident that much work was yet to be done in learning to use a vector machine. At that time, only a few of the CYBER 205's had been installed. One year later, we see numerous examples of good vectorizing work carried out by still relatively inexperienced vector computer users. Clearly, in time we shall see a great deal more optimization and effective performance becoming routine.

CONTENTS

CYBER 200 Applications Seminar

Executive Chairman: Dr. Sidney Fernbach

Hosts: J. Patrick Gary, Goddard Space Flight Center
John Zelenka, Control Data Corporation

SESSION II

SESSION III

MATHEMATICAL ALGORITHMS TO MAXIMIZE PERFORMANCE IN NUMERICAL WEATHER PREDICTION

AILEEN FOREMAN

UNITED KINGDOM METEOROLOGICAL CENTER

BRACKNELL, BERKSHIRE
UNITED KINGDOM

Aileen Foreman

Mathematical Algorithms to Maximize Performance in Numerical Weather Prediction

Introduction

Numerical weather prediction models, which involve the solution of non-linear partial differential equations at points on an extensive three-dimensional grid, are ideally suited for processing on vector machines. It was logical therefore that the new global forecast model to be implemented at the Meteorological Office should be written in vector code for the Cyber 205.

In order to achive full efficiency and to reduce storage requirements the model used 32-bit arithmetic which had been found to provide high enough precision. Unfortunately, however, the trigonometrical and logarithmic functions provided by CDC could only handle 64-bit vectors and, although written in efficient scalar code, did not take advantage of the special facilities of a vector processor. It was therefore necessary to rewrite the functions in vector code to handle both 32 and 64-bit vectors. There was also no half-precision compiler available for the Cyber 205 at that time and so the functions, like the model, had to make extensive use of the "special call" syntax. This made the code more difficult to write but it allowed much greater flexibility in that it became possible to access the exponent of a floating-point number independently of its coefficient.

This paper presents a description of the techniques and it summarises the results which were achieved. One example, the logarithmic function, is treated here in detail to illustrate the general approach to the problem.

Derivation of logarithms

The coding for the logarithm function illustrates both the use of the way in which floating-point numbers are stored and the use of linked triads to gain additional speed.

To calculate $y = \log_e(x)$ we divide the range of x into two, the first of which is

a) $x \geq \sqrt{2}$ and $x < \frac{\sqrt{2}}{2}$

We first write the value of x in a way which can be related to the format of stored floating-point numbers. Thus, introducing two new unknowns n and ω, n being an integer and $\frac{1}{2} \leq \omega \leq 1$, we may write any number as $x = 2^n \omega$.

Now the Cyber 205 stores the floating-point number as

$$2^{exp} \cdot coefficient = 2^{exp} \cdot 2^{j} \cdot k$$

where the factor 2^j is introduced by normalization.

Since for logarithms, x must always be positive, for 64-bit numbers bit 17 will be on, so j = 46 and for 32-bit numbers bit 9 will be on, so j = 23.

Then relating the two, we have $n = exp + j$ and $\omega = k$

As an example, if x = 2.0 as a 64-bit normalized value

$$x = 2^{-45}.2^{46}$$

so from the above formulae

$$n = -45 + 46 = 1 \qquad \text{and} \qquad \omega = 1.0$$

Here, we can obtain the values of n and ω very easily as we can access the exponent and coefficient of a number by using special calls.

The next step is to convert the functions into a suitable form for vectorization and this involves the introduction of a new variable

$$z = \left(\frac{\omega - \sqrt{2}/2}{\omega + \sqrt{2}/2}\right)$$ which can be computed at the same time as ω .

$$\text{Then} \quad \omega = \left(\frac{1+z}{1-z}\right)\frac{\sqrt{2}}{2}$$

From the original definition

$$x = 2^{n-1/2}\left(\frac{1+z}{1-z}\right)$$

thus $$\log_e x = \left(n - \frac{1}{2}\right)\log_e 2 + \log_e\left(\frac{1+z}{1-z}\right)$$

b) For the remaining values of x , within the range $\frac{\sqrt{2}}{2} \leq x < \sqrt{2}$, the value of z is defined by:

$$z = \frac{x-1}{x+1} \qquad \text{so that} \qquad x = \frac{1+z}{1-z}$$

Then $$\log_e x = \log_e \frac{1-z}{1+z} \qquad \text{for } \frac{\sqrt{2}}{2} \leq x < \sqrt{2}.$$

In each case, the problem then becomes one of vectorizing $\log_e\left(\frac{1+z}{1-z}\right)$ which is easily done by replacing it with a truncated series which gives the required degree of precision:

$$\log_e\left(\frac{1+z}{1-z}\right) = \sum_{m=0}^{6} c_m z^{2m+1}$$

where the constants c_m are known.

Then $\log_e\left(\frac{1+z}{1-z}\right) =$

$$((((((c_6 z^2 + c_5) z^2 + c_4) z^2 + c_3) z^2 + c_2) z^2 + c_1) z^2 + c_0) z$$

Despite its complicated appearance, this reduces to eight vector operations consisting of a multiplication, six linked triads and a final multiplication by z thus

Multiplication to give z^2

First triad = V1 = $c_6 z^2 + c_5$

Second triad = V2 = $V1\, z^2 + c_4$

Third triad = V3 = $V2\, z^2 + c_3$ etc.

Tests, using the 1.5 compiler, and a range of vector lengths gave the following results, with times being expressed in units of 10^{-4} seconds.

Vector length	10	50	100	200	500	1000	2000	5000
CDC logarithms	.3	.55	.7	1.01	2.00	3.66	7.04	21.50
64-bit vector logarithm	.47	.61	.78	1.12	2.16	3.87	7.47	20.15
32-bit vector logarithm	.53	.57	.65	.82	1.34	2.20	3.99	9.66

The first point to notice here is that the full increase in speed for 32-bit vectors is only achieved with large vector lengths. Because of the overheads associated with the initiation of vector instructions, this is not unexpected and is common to all of the functions to be described. What is unexpected is that no improvement in speed was achieved for our 64-bit function when compared to the CDC function. In this respect, this function is unique among all those treated in this paper. However, the original aim of producing a 32-bit version has been successfully achieved.

Exponentials

The exponential function is derived from the standard formula

$e^x = 2^k . 2^{m/16} . 2^{f/16}$ chosen to make use of special calls. k, m and f are defined as follows:

If $n = int\left[\frac{16x}{\log_e 2}\right]$

then $k = int\left[\frac{n}{16}\right]$ and $m = n$ modulo 16 for $x \geq 0$

and $k = int\left[\frac{n}{16}\right] - 1$ and $m = 16 - n$ modulo 16 for $x < 0$

$$f = \left(\frac{16x}{\log_e 2}\right) - n$$

Now, since m is integer and $0 \leq m < 16$, the factor $2^{m/16}$ is obtained from a look-up table of 16 elements of known values, using the "special call" instruction Q8VXTOV.

Having found the integer k from the above formula, and $2^{m/16}$ from the look-up table, to obtain the value $2^k . 2^{m/16} = 2^{k+m/16}$ we add k to the exponent part of $2^{m/16}$ by using special calls.

The factor, $2^{\delta/16}$ is given by

$$2^{\delta/16} = \frac{P_3\delta^3 + \delta^2 + P_1\delta + P_0}{-(P_3\delta^3 - \delta^2 + P_1\delta - P_0)}$$

where f is obtained as above and P_0, P_1, P_3 are known constants.

Then, to obtain e^x all we need is a final multiply of $2^{\delta/16}$ by $2^{n+m/16}$

The following results were achieved, times again being given in units of 10^{-4} seconds.

vector length	10	50	100	200	500	1000	2000	5000
CDC exponential	.35	.7	.93	1.44	2.86	5.25	10.52	33.36
64-bit vector exponential	.47	.6	.78	1.14	2.29	4.15	7.97	22.75
32-bit vector exponential	.47	.56	.68	.93	1.85	3.14	5.85	14.62

Here, for a vector length of 5000 the 32-bit exponential routine is only 40% faster than the 64-bit routine because of the use of the "special call" Q8VXTOV. However the 64-bit routine has achieved a considerable speed-up over the CDC exponential.

The Hyperbolic functions

The routines to calculate the hyperbolic functions $y=\cosh x$, $y=\sinh x$ and $y=\tanh x$ use the following formula,

$$\cosh x = \frac{1}{2}\left(e^x + e^{-x}\right)$$

The calculation of e^x is as described earlier. During the calculation of e^x, little extra work is required to obtain e^{-x} which avoids the need to call the exponential routine twice.

The hyperbolic sine is given by

$$\sinh x = \frac{1}{2}\left(e^x - e^{-x}\right) \qquad \text{for} \quad |x| \geq 0.5$$

and

$$\sinh x = \sum_{m=0}^{5} \frac{x^{2m+1}}{(2m+1)!} \qquad \text{for} \quad |x| < 0.5$$

Here the two distinct cases are treated independently, so that we are dealing with shorter vector lengths, and then the results are merged together at the end of the routine. The polynomial expansion of sinh x can be performed in seven vector instructions, by using linked triads.

The hyperbolic tangent is given by

$$\tanh x = \sum_{m=0}^{5} c_m x^{2m+1} \qquad \text{for} \quad 0 \leq |x| \leq 0.12$$

$\tanh x = 1 - \frac{2}{e^{2x}+1}$ for $0.12 < |x| \leq 18.0$

$\tanh x = 1.0$ for $x > 18.0$

$\tanh x = -1.0$ for $x < -18.0$

Again, the distinct cases are treated independently so that we are dealing with shorter vector lengths, and again we can use linked triads when calculating the polynomial expansion of $\tanh x$.

The timings of the hyperbolic sine and hyperbolic tangent routines are data dependent, but some sample timings are given below. All times are expressed in units of 10^{-4} seconds.

vector length	10	50	100	200	500	1000	2000	5000
hyperbolic cosine								
64-bit vector	.55	.79	1.08	1.68	3.45	6.41	13.26	37.65
32-bit vector	.54	.69	.88	1.27	2.44	4.44	8.72	22.99
hyperbolic sinh								
64-bit vector	.75	.99	1.30	1.96	3.88	7.27	14.87	43.85
32-bit vector	.72	.87	1.07	1.48	2.74	5.00	9.47	24.38
hyperbolic tangent								
64-bit vector	.66	.87	1.15	1.68	3.33	6.01	11.79	34.83
32-bit vector	.64	.73	.89	1.21	2.30	3.66	6.87	17.76

Again, we see that for very short vector lengths we do not have a great advantage by using 32-bit vectors, but for longer vector lengths we are approaching twice the speed of the 64-bit functions. There were no CDC functions available to compare with our results.

Sines and cosines

The trigonometrical functions, $y = \sin x$ and $y = \cos x$ are calculated from the polynomial expansion of $\sin x$ so that we can make use of linked triads again. First the input argument needs to be reduced modulo 2π. This is achieved by

letting $r_1 = \frac{2|x|}{\pi}$ and $r_2 = int\left[\frac{2|x|}{\pi}\right]$

then put $z = r_1 - r_2$ so that $0 \leq z < 1$.

and $k = r_2$ modulo 4

So $\sin(x)$ is given by

$\sin x =$ $\sin z$ for $k=0$

$\sin(1-z)$ for $k=1$

$-\sin z$ for $k=2$

$-\sin(1-z)$ for $k=3$

where $\sin z = \sum_{m=0}^{8} c_m z^{2m+1}$ for 64-bit function

and the constants c_m are known.

Because the values c_7 and c_9 are too small to affect the accuracy of the 32-bit function results:

$$\sin z = \sum_{m=0}^{6} c_m z^{2m+1}$$ for 32-bit vector function

The cosine function is given by

$$\cos x = \sin\left(\frac{\pi}{2} + x\right)$$ where $\sin\left(\frac{\pi}{2} + x\right)$ is calculated as above.

If it is known that the input operand, x, is always between -2π and $+2\pi$ radians, much work can be left out of the routine;

for as above let $r_1 = \frac{2|x|}{\pi}$ and $r_2 = int\left[\frac{2|x|}{\pi}\right]$

and so $0 \le r_2 \le 3$

and $k = r_2$ modulo $4 = r_2$

and again $z = r_1 - r_2$ so $0 \le z < 1$

So for $k = r_2 = 0$, $z = r_1 - r_2 = r_1$, $\sin x = \sin(z) = \sin(r_1)$

for $k = r_2 = 1$, $z = r_1 - r_2 = r_1 - 1$, $\sin x = \sin(1 - z) = \sin(2 - r_1)$

for $k = r_2 = 2$, $z = r_1 - r_2 = r_1 - 2$, $\sin x = -\sin(z) = \sin(2 - r_1)$

for $k = r_2 = 3$, $z = r_1 - r_2 = r_1 - 3$, $\sin x = -\sin(1 - z) = \sin(r_1 - 4)$

Thus we have two sets of functions, one set to calculate the sine and cosine of any angle expressed in radians, and the other to calculate the sine and cosine of angles between -2π and $+2\pi$ radians.

The polynomial expansion of sin(z) can be calculated in ten vector instructions including eight linked triad instructions for the 64-bit function and in eight vector instructions using six linked triad instructions for the 32-bit functions.

Tests gave the following results with times given are expressed in units of 10^{-4} seconds.

vector length	10	50	100	200	500	1000	2000	5000
CDC sine	.15	.5	.64	.91	1.72	3.07	6.13	22.98
64-bit vector sine (all angles)	.49	.59	.72	.98	1.74	3.02	5.59	14.98
32-bit vector sine (all angles)	.42	.46	.52	.63	.98	1.57	2.76	6.35
64-bit vector sine (-2π to $+2\pi$)	.37	.44	.53	.72	1.27	2.20	4.07	10.04
32-bit vector sine (-2π to $+2\pi$)	.34	.37	.41	.50	.75	1.20	2.09	4.78

vector length	10	50	100	200	500	1000	2000	5000
CDC cosine	.3	.55	.68	.99	2.08	3.29	6.68	23.59
64-bit vector cosine (all angles)	.57	.60	.73	.99	1.87	3.19	5.94	16.00
32-bit vector cosine (all angles)	.69	.47	.51	.63	1.0	1.70	2.94	6.95
64-bit vector cosine (-2π to $+2\pi$)	.72	.45	.55	.74	1.42	2.40	4.45	11.14
32-bit vector cosine (-2π to $+2\pi$)	.67	.37	.41	.50	.77	1.37	2.31	5.51

Thus, we can see that we need a vector length of 500 to 1000 before our 64-bit routines for all angles are faster than the CDC supplied routines, but that our 32-bit routines for restricted angles between -2π and $+2\pi$ are over four times as fast as the CDC routines for vector lengths of 5000.

Tangents

Similarly for the trigonometrical function, $y = \tan x$ we have supplied two sets of functions, one set to calculate the tangent of any angle expressed in radians in both 64-bits and the other to calculate the tangent of angles between -2π and $+2\pi$ radians in both 64-bits and 32-bits. The tangent function is calculated using a polynomial expansion of tan(x) to make use of linked triads. The calculation is performed by first reducing the argument modulo 2π

Let $r_1 = \frac{4x}{\pi}$ and $r_2 = int\left[\left|\frac{4x}{\pi}\right|\right]$

then $z = r_1 - r_2$ so that $0 \le z < 1$

Now let $s = r_2$ modulo 8, putting $k = s$ if $0 \le s \le 3$
and $k = s-4$ if $4 \le s \le 7$

$\tan(x)$ is now given by

$$\tan(x) = \tan(z) \quad \text{for} \quad k=0$$

$$= \frac{-1}{\tan(z-1)} \quad \text{for} \quad k=1$$

$$= \frac{-1}{\tan(z)} \quad \text{for} \quad k=2$$

$$= \tan(z-1) \quad \text{for} \quad k=3$$

where $\tan(z) = \sum_{m=0}^{12} c_m z^{2m+1}$ to the required degree of precision.

Again, if it is known that the input operand is always between -2π and $+2\pi$ radians, we can write:

$$r_1 = \frac{4x}{\pi} \qquad \text{and} \qquad r_2 = \text{int}\left[\left|\frac{4x}{\pi}\right|\right]$$

and so $0 \le r_2 \le 7$

In this case $s = r_2$ modulo $8 = r_2$

Then $k = r_2$ where $0 \le r_2 \le 3$
and $k = r_2 - 4$ where $4 \le r_2 \le 7$

and the calculation continues as before.

The polynomial expansion of tan(z) is calculated in fourteen vector instructions using twelve linked triads.

The resulting timings of tests are given below, expressed in units of 10^{-4} seconds.

vector length	10	50	100	200	500	1000	2000	5000
CDC tangent	.98	.73	.91	1.47	2.61	4.71	9.33	30.80
64-bit vector tangent (all angles)	.90	.82	.99	1.35	2.55	4.48	8.40	22.67
32-bit vector tangent (all angles)	.96	.78	.90	1.14	1.92	3.21	5.59	13.29
64-bit vector tangent (-2π to $+2\pi$)	.67	.76	.93	1.25	2.36	4.14	7.74	20.64
32-bit vector tangent (-2π to $+2\pi$)	.67	.70	.80	.99	1.76	2.94	5.15	11.98

These results show that we need a vector length of only about 200 before our 64-bit tangent function for all angles is faster than the CDC routine, and that our 32-bit tangent function for restricted angles between -2π and $+2\pi$ radians is well over twice as fast as the CDC routine.

The Arctangent function

The arctangent function $y = a\tan(x)$ is again calculated from a polynomial expansion so that we can use linked triads. The calculation is performed as follows:

For $|x| \geq \sqrt{2}+1$ let $\omega = \frac{1}{|x|}$

and for $|x| < \sqrt{2}+1$ let $\omega = |x|$

Change the variable to z, defined by

$$z = \frac{\omega - a}{a + \omega a^2}$$

where, a is chosen so that z = 1.0 when $\omega = \sqrt{2}+1$

Under this condition, $a = (1-\sqrt{2}) + \sqrt{4 - 2\sqrt{2}}$, and is therefore a constant.

Then atan(x) is given by

atan(x)=atan(z)+atan(a)

Here, atan(a) is a constant and need only be calculated once, and we may replace atan(z) by the truncated series:

$$a\tan(z) = \sum_{m=0}^{4} c_m z^{2m+1}$$

For $|x| \geq \sqrt{2}+1$, $a\tan(x) = \frac{\pi}{2} - a\tan\left(\frac{1}{x}\right)$

and for $x < 0$, $a\tan(x) = -a\tan(x)$

Atan(z) can be calculated in ten vector instructions, eight of which are linked triad instructions. The results are in the range $-\frac{\pi}{2}$ to $+\frac{\pi}{2}$ (not inclusive).

The following results were achieved, times again being given in units of 10^{-4} seconds.

vector length	10	50	100	200	500	1000	2000	5000
CDC arctangent	.38	.90	1.19	1.97	4.06	7.35	16.19	46.09
64-bit vector arctangent	.48	.52	.66	.92	1.91	3.07	5.77	15.23
32-bit vector arctangent	.43	.49	.55	.69	1.10	1.79	3.34	7.27

These results are spectacular, in that the 32-bit arctangent function is over six times as fast as the CDC routine and even the 64-bit version has given a threefold increase in speed.

Derivation of arcsine and arccosine functions

The final trigometric routines to be considered calculate the arcsine and arccosine of x. The calculations are performed as follows.

for $0 \le x \le 1/2$, let $z = x$ so that asin(x) = asin(z)

and for $\frac{1}{2} < x \le 1$, let $z = \left(1 - \frac{x}{2}\right)^{1/2}$ and asin(x) = $\frac{\pi}{2} - 2\,asin(z)$

for $-1 \le x < 0$, asin(x) = asin(-x) and the same substitutions are used.

Now the new variable, z, must be between zero and 0.7 so we may write

$$asin(z) = \sum_{n=0}^{11} C_n z^{2n+1}$$ to the required degree of precision.

The arccosine function is derived from the arcsine using the substitution

$$acos(x) = \frac{\pi}{2} - asin(x)$$

The polynomial expansion of asin(z) is calculated in thirteen vector instructions, eleven of which are linked triads. The range of the results for arccosine is $\frac{-\pi}{2}$ to $+\frac{\pi}{2}$ inclusive, and for arccosine is 0 to π inclusive.

The following results were achieved, with times expressed in units of 10^{-4} seconds.

vector length	10	50	100	200	500	1000	2000	5000
CDC aecsine	.5	.67	.87	1.27	2.6	4.73	9.64	29.84
64-bit vector arcsine	.52	.61	.75	1.04	2.02	3.55	6.69	16.54
32-bit vector arccosine	.54	.51	.58	.73	1.37	2.25	3.91	9.11

vector length	10	50	100	200	500	1000	2000	5000
CDC arccosine	.26	.68	.89	1.27	2.41	4.35	9.16	28.55
64-bit vector arccosine	.51	.61	.76	1.05	1.95	3.44	6.44	18.73
32-bit vector arccosine	.48	.54	.61	.76	1.25	2.07	3.66	8.59

Here our 32-bit functions are over three times as fast as the CDC routines, for vector lengths of 5000.

Conclusion

The trigonometrical and logarithmic functions, as provided by CDC up to and including version 2.0 of the compiler are, in general, not very efficient. At the Meteorological Office, we found it necessary to hand-code these functions in vector syntax to take full advantage of the facilities of the Cyber 205. For the 32-bit versions, which have a high enough precision for most of our purposes, speed increases of up to six times were obtained and even for our 64-bit versions,

increases of up to three times are possible. However, CDC have undertaken to provide fully vectorized versions of the trigonometrical andlogarithmic functions in both 64-bits and 32-bits by release 2.1 of the compiler.

The functions described were written in the "special call" syntax because of compiler limitations and the difficulties associated with this were partly offset by the special features which were then available. Users with the 2.0 compiler could find that the extra facilities provided by the "special calls" do not overcome the difficulties involved with this syntax and that coding explicitly in the FORTRAN vector syntax achieves sufficient vectorization for their own purposes.

COMPUTER SIMULATIONS OF SPACE-BORNE METEOROLOGICAL SYSTEMS ON THE CYBER 205

MILTON HALEM

NASA/GODDARD SPACE FLIGHT CENTER

GREENBELT, MARYLAND

COMPUTER SIMULATIONS OF SPACE-BORNE METEOROLOGICAL SYSTEMS ON THE CYBER 205

M. Halem

NASA/Goddard Space Flight Center
Greenbelt, MD

ABSTRACT

The complete global specification of the state-of-the-atmosphere on a daily or more frequent basis is required for numerical weather forecasting. Although the number of atmospheric variables required are small, namely, temperature, winds, moisture and surface pressure, globally and throughout the atmosphere, no single space-borne instrument is able to meet these requirements at the desired degree of accuracy and coverage. As a result, investigators have proposed to NASA a number of composite systems with differing limitations in accuracy and coverage under different atmospheric conditions.

Because of the extreme expense involved in developing and flight testing these instruments, an extensive series of numerical modeling experiments to simulate the performance of these meteorological observing systems have been performed on the CYBER 205. The studies compare the relative importance of different global measurements of individual and composite systems of the meteorological variables needed to determine the state of the atmosphere. The assessments are made in terms of the systems ability to improve 12 hour global forecasts. Each experiment involves the daily assimilation of simulated data that is obtained from a data set we call "nature." This data is obtained from two sources: first, a long two-month general circulation integration with the GLAS 4th Order Forecast Model and second, global analysis prepared by the National Meteorological Center, NOAA, from the current observing systems twice daily. More than two dozen experiments representing different possible configurations were carried out and analyzed. The experiments extend over a typical winter month, February, and successive 12 hour forecasts are made from the analysis twice daily. Thus, statistics are compiled from a total of 56 forecasts for each experiment.

This voluminous number of experiments would have taken over a year on a dedicated 24 hour per day allocation on an Amdahl V-6. The study was completed in less than a month on an as available basis on the Cyber 205 at the NASA High Speed Computing Facility.

OPERATIONAL NUMERICAL WEATHER PREDICTION ON THE CYBER 205 AT THE NATIONAL METEOROLOGICAL CENTER

DENNIS DEAVEN

NATIONAL WEATHER SERVICE

WASHINGTON, D.C.

Operational Numerical Weather Prediction on the Cyber 205 at the National Meteorological Center

Dennis Deaven

NOAA/NWS

Washington,D.C.

The Development Division of the National Meteorological Center (NMC) has the responsibility of maintaining and developing the numerical weather forecasting systems of the center. Because of the mission of NMC these products must be produced reliably and on time twice daily free of surprises for forecasters. Personnel of Development Division are in a rather unique situation. We must develop new advanced techniques for numerical analysis and prediction utilizing current state-of-the-art techniques, and implement them in an operational fashion without damaging the operations of the center.

In the past, modifications have been made to the operational job suite without adequate testing and evaluation because computational resources were not available to produce enough case studies for evaluation. Hopefully, with the computational speeds and resources now available from the Cyber 205, Development Division Personnel will be able to introduce advanced analysis and prediction techniques into the operational job suite without disrupting the daily schedule.

The operational job suite prior to the installation of the Cyber 205 contained four major components: 1. A barotropic numerical model extending over the Northern Hemisphere giving forecasters an early look at the new synoptic situation immediately after data collection at the start of the twice daily operational cycle. 2. A Limited Fine Mesh (LFM) primitive equation numerical model extending over the North American continent. The LFM is started about 1 hour 45 minutes after data collection producing numerical guidance for use by forecasters when they make their 12 to 48 hour forecasts. 3. A global primitive equation numerical model using a spectral representation to produce numerical guidance for use by forecasters in the 2 to 5 day range. This model is started at about 4 hours after each twice daily collection of atmospheric data. 4. A global data assimilation cycle is started about 10 hours after data collection and is used to produce the first guess fields for the next synoptic cycle. The data assimilation cycle consists of an optimum interpolation analysis and a global spectral model which are used to produce two six hour analysis/forecast cycles. In addition to these four major components, a Moveable Fine Mesh model is available when needed to produce forecasts of hurricane movement. The hurricane model has the capability to move with the hurricane as it forecasts the storm track for periods of 48 hours.

The operational implementation of these analysis/forecast systems on the Cyber 205 will have to proceed in a careful controlled manner so that daily production schedules are maintained. For this reason, each component of the operational suite must be carefully evaluated and tested after conversion to the Cyber 205. All components of the present system scheduled for implementation on the Cyber 205 will be converted in their present form with the current resolution and numerics in order to evaluate their performance in a parallel fashion. After about a month of successful parallel tests the component will become operational on the Cyber 205.

The National Weather Service received their Cyber 205 in May of 1983 and the first operational product appeared on August 30, 1983. The LFM was successfully implemented on the Cyber 205 and has been producing numerical guidance twice a day since that time. The final version of the LFM computer program that was implemented takes about 75 seconds of CPU time to produce a 48 hour forecast. This is about 15 times faster than the IBM/195 version of the same model. The LFM is a grid-point model containing 7 layers with 53 x 45 grid points in each layer. Five prognostic variables (pressure, temperature, moisture, and two components of wind speed) are specified at each of the 16,695 grid points. The primitive equations are solved in finite difference form for each of the prognostic variables and then advanced forward in time with an explicit

time step. Nine 400 second time steps are required for each hour of model integration which yields a total of 432 explicit time steps to produce a 48 hour prediction.

The conversion of the LFM computer code to the Cyber 205 was accomplished in about 1.5 months by a skilled meteorologist/programmer. The 2.0 FORTRAN compiler was used to produce a half precision version without resorting to Q8 special calls. The data structure of the original version of the model was changed extensively to take advantage of long vector lengths. Minimal vectorization of the radiation and moist physics was achieved with use of the vector WHERE statement.

Operational use of the Cyber 205 has shown that the system is certainly reliable and capable of achieving vendor advertised CPU speeds. With this new resource the National Weather Service should be able to improve most aspects of numerical weather prediction systems including the prediction of major precipitation events. With the increase in computing power, the National Weather Service will be able to run operational numerical guidance systems with improved analysis methods, improved model physics and increased mathematical accuracy.

OCEAN MODELLING ON THE CYBER 205 AT GFDL

MICHAEL D. COX

GFDL/NOAA
PRINCETON UNIVERSITY

PRINCETON, NEW JERSEY

Ocean modelling on the CYBER 205 at GFDL

Michael D. Cox

1. Introduction

At the Geophysical Fluid Dynamics Laboratory, research is carried out for the purpose of understanding various aspects of climate, such as its variability, predictability, stability and sensitivity. The atmosphere and oceans are modelled mathematically and their phenomenology studied by computer simulation methods. The present paper will discuss the present state-of-the-art in the computer simulation of large scale oceans on the CYBER 205. While atmospheric modelling differs in some aspects, the basic approach used is similar.

The equations of the ocean model will be presented in the following section along with a short description of the numerical techniques used to find their solution. Section 3 will deal with computational considerations and a typical solution will be presented in section 4.

2. Equations of the model

The model presented here is the multilevel numerical model described in Bryan (1969). The continuous equations will be given. A detailed description of the finite difference formulation may be found in the above work. The equations of motion are the Navier-Stokes equations written in spherical coordinates and modified by the Boussinesq approximation. Let $m=\sec\phi$, $n=\sin\phi$, $u=a\dot{\lambda}m^{-1}$ and $v=a\dot{\phi}$, where a is the radius of the earth, ϕ the latitude and λ the longitude. It is convenient to define the advection operator

$$\Gamma(\chi)=ma^{-1}[(u\chi)_\lambda+(v\chi m^{-1})_\phi]+(w\chi)_z. \tag{1}$$

The equations of motion on a sphere are

$$u_t+\Gamma(u)-2\Omega nv=-ma^{-1}(P/\rho_o)_\lambda+F^\lambda, \tag{2}$$

$$v_t+\Gamma(v)+2\Omega nu= -a^{-1}(P/\rho_o)_\phi+F^\phi, \tag{3}$$

$$\Gamma(1)=0, \tag{4}$$

$$g\rho=-P_z, \tag{5}$$

where ρ_o is unity in cgs units. The conservation equations for the temperature and salinity are

$$T_t+\Gamma(T)=F^T \tag{6}$$

$$S_t+\Gamma(S)=F^S \tag{7}$$

The terms in F contain effects of mixing as well as external driving forces. The equation of state

$$\rho=\rho(T,S,z) \tag{8}$$

is an empirically derived formula relating the local density of seawater to temperature, salinity and depth.

The set of equations (1-8) are cast into finite difference form. The prognostic equations (2,3,6,7) are solved as an initial value problem, placing all terms except the local time derivative on the right hand side and carrying out timesteps to predict new values of velocity, temperature and salinity on a prescribed mesh covering the model ocean domain. Given a certain configuration of steady wind driving and differential surface heating (both entering through the F terms), a statistical steady state is approached asymptotically in time. Time scale analysis of Eqs.(6,7) reveals that O(1000) years of integration is needed to bring the sluggish abyssal layers of the ocean model into a steady state.

3. Computational considerations

Let us consider a rectangular ocean basin model comparable in size to the N. Atlantic Ocean. It extends 60° in longitude, 65° in latitude and 4000 meters in depth. It is desirable to cover this domain with a mesh fine enough to resolve mesoscale (O(100 km)) eddies which play an important role in transporting various properties through the ocean. The minimum resolution needed for this purpose is roughly 1/3rd degree in latitude and somewhat larger, say .4 degree in longitude due to the convergence of meridians on the globe. This results in a horizontal grid space of 150x195 points. Vertically, 18 levels are needed to resolve the scales of interest. This brings the total to just over 1/2 million grid points for which Eqs.(1-8) must be evaluated each timestep.

The longest timestep which can be used without incurring numerical instability is given by the Courant-Friedrichs-Lewy condition

$$c\Delta t/\Delta x < 1 \qquad (9)$$

where c is the phase velocity of the fastest moving wave in the ocean. Since high speed external gravity waves have been filtered from this model by the condition w=0 at the surface, the fastest wave is that associated with the internal density gradients (internal gravity wave) which has a speed of roughly 3m/sec. The smallest Δx occurs at the northern wall of the model due to convergence of meridians, and is about 20 km. The resultant Δt is such that roughly 5000 timesteps are necessary to integrate one year. Therefore, 5 million timesteps, or 2.5×10^{12} grid point evaluations of Eqs.(1-8), are required to integrate this model to a steady state. Even the fastest modern day computers cannot accomplish this task in a reasonable time, although steady progress is being made. The former computer at GFDL, the Texas Instruments ASC, took 15 seconds to compute one time step on the above model. At this speed, 2.4 years of computing would be needed to reach a steady state solution. Clearly, compromises must be made in designing experiments which are achievable in a reasonable amount of computer time. This may involve reducing the domain size, or integrating for a shorter period, or both. (Interesting results may be obtained from an integration of O(10) years, particularly for the upper ocean

where time scales of adjustment are relatively short.) The greater the computational speed which can be attained, the less severe the compromises must be.

In converting the ASC ocean model to the CYBER 205, the most fundamental alteration of the code had to do with the treatment of land masses. Previously, the computation was carried out only over ocean points by making the DO loop limits functions of the placement of land. The contiguity requirement of the 205 for vectorization allows only the innermost of the three dimensional loops to vectorize in this case. An alternative method of handling land is to compute all points as if they were ocean and, at the end of the timestep, restore the land to its specified value using a masking array. Contiguity is then satisfied and vectorization is enabled through two dimensions. (The third dimension cannot be vectorized because it is cycled through memory from disc.) By using the latter technique, the typical vector length in the computation is increased from 150 in the example above (east-west dimension) to 2700 (east-west times depth dimension) resulting in a considerable decrease in the relative time spent in vector startup.

An additional time saving has been accomplished in an area of the code which is used heavily, but is inherently unvectorizable due to a recursive property. Using Q8 calls to insert machine language directly into the FORTRAN, CDC personnel have "unrolled" this loop, greatly improving on the code generated by the compiler for the equivalent FORTRAN loop.

The use of half-precision on all floating point variables has resulted in a gain of only about 15% in overall running speed, although sections of the code which are 100% vectorized increase in speed by roughly 40%. Additional work is needed to determine why the overall gain is so small considering the high degree of vectorization of the code.

Since the model above is too large to fit into core memory entirely, data is cycled through memory from disc as it is needed each timestep. If this disc transfer cannot be buffered sufficiently well, computation ceases while waiting for the I/O to finish. The result is that the computer may not be used efficiently, particularly if the other jobs running concurrently have the same difficulty. Until recently, this was a severe problem on the 205. The above model, when in the 205 alone, ran only about 15% of the wall clock time. Improved I/O schemes have been developed by CDC personnel at GFDL and currently the same model runs about 80% of the wall clock time when alone. This compares

favorably with I/O efficiencies on the ASC.

The CYBER 205 version of the model described above currently takes 4 seconds to compute one timestep, almost a factor of 4 faster than the ASC. While this speed still does not make the experiment proposed at the beginning of this section feasible, the compromises which are necessary to produce an attainable solution are much less severe than before. One such experiment will be described in the following section.

4. An ocean simulation experiment

If one wishes to study the effects of topography on the dynamics of the Gulf Stream, an argument can be made that it is not necessary to consider a domain as large as the one proposed earlier, and that several decades of integration is sufficient. Therefore, let us reduce the domain from 65 to 27 degrees in latitude and from 60 to 32 degrees in longitude. Also, for this purpose, the vertical resolution may be decreased from 18 layers to 5 layers. This produces a model which takes approximately one hour of 205 time to integrate one year of ocean time. Applying surface wind stress and differential heating similar to that of the N. Atlantic, this model has been integrated from rest a total of 20 years. The resulting temperature pattern at the second layer, centered at 212 meters depth, is shown in Fig. 1. The land mass in the northwest corner simulates the gross features of the U.S. east coast. A continental shelf and slope is also included in this solution. The simulated Gulf Stream is revealed by the tightly packed isotherms along the coast and bending out to sea at the point representing Cape Hatteras. In agreement with observations, there exist both cold and warm core "rings" which have broken from the Stream and are drifting westward. An example of the former is centered at about 70°W, 30°N and of the latter at 68°W, 37°N.

Three other experiments have been carried out in this series, altering the topography along the western boundary to study its effect on the path and behavior of the Gulf Stream.

References

Bryan, K., 1969 A numerical method for the study of the circulation of the World Ocean. J. Comput. Phys., 4, 347-376.

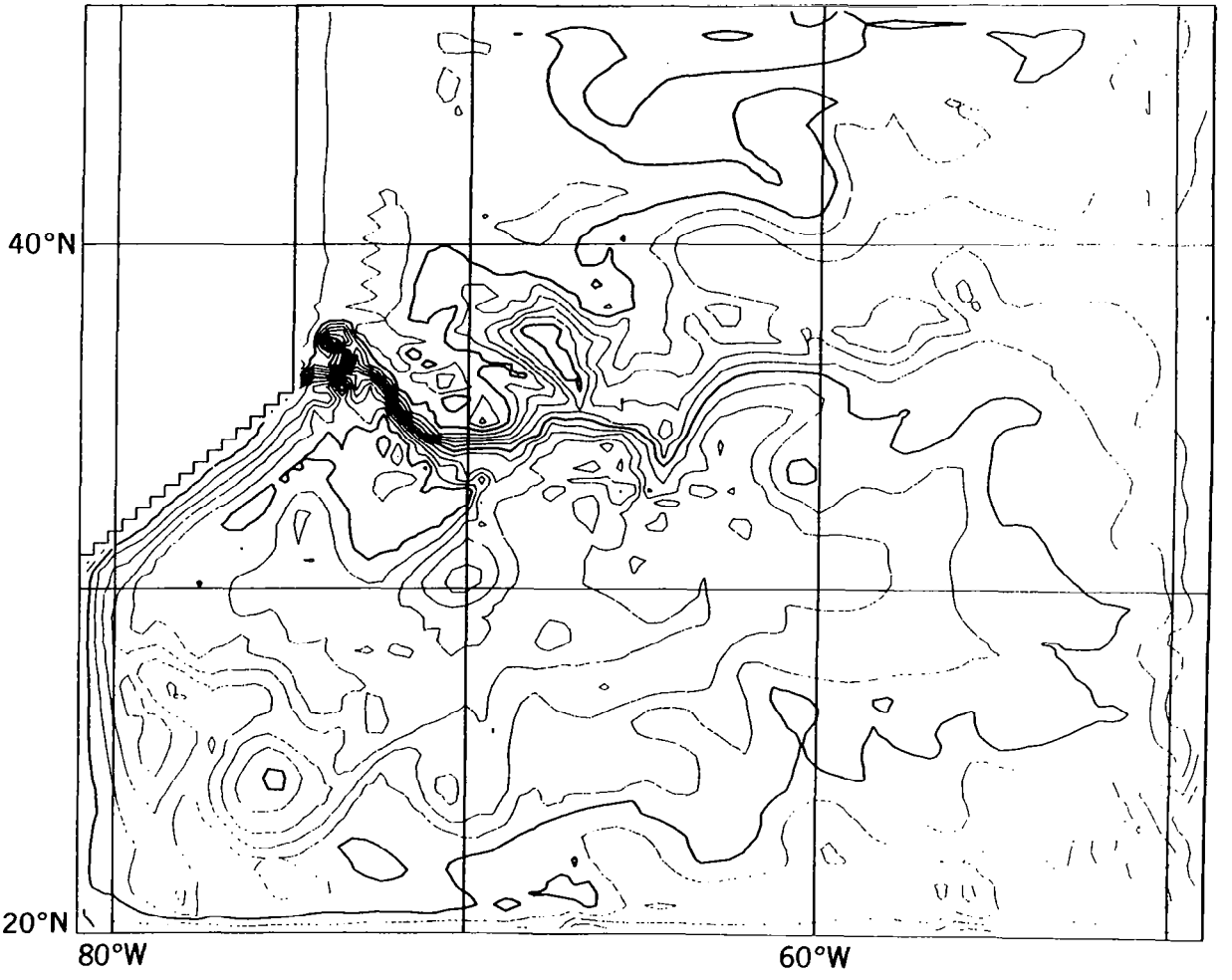


Fig. 1 Temperature at 212 meters depth. The contour interval is 1°C.

MEMORY EFFICIENT SOLUTION OF THE PRIMITIVE EQUATIONS FOR NUMERICAL WEATHER PREDICTION ON THE CYBER 205

JAMES J. TUCCILLO

SYSTEMS AND APPLIED SCIENCES CORPORATION

HYATTSVILLE, MARYLAND

Memory Efficient Solution of the Primitive Equations for Numerical Weather Prediction on the CYBER 205

James J. Tuccillo

Systems and Applied Sciences Corporation

5809 Annapolis Road

Hyattsville, MD 20784

1. INTRODUCTION

Numerical Weather Prediction (NWP), for both operational and research purposes, requires not only fast computational speed but also large memory. In this paper I will discuss a technique for solving the Primitive Equations for atmospheric motion on the CYBER 205, as implemented in the Mesoscale Atmopsheric Simulation System (MASS) (Kaplan et. al., 1982), which is fully vectorized and requires substantially less memory than other techniques such as the Leapfrog or Adams-Bashforth Schemes. The technique to be presented uses the Euler-Backard time marching scheme.

Also to be discussed will be several techniques for reducing the CPU time of the model by replacing "slow" intrinsic routines by faster algorithms which use only hardware vector instructions.

2. MODEL BACKGROUND

2.1 Description

MASS is a hydrostatic primative equation model which is run over a limited area. The model forecast the 3-dimentional structure of wind, pressure, temperature and moisture. The actual domain of coverage, along with the horizontal distribution of grid points, is depicted in Fig. 1. The characteristics of the model are listed in Table 1.

2.2 Uses and Support

The model has been applied primarily to the problem of forecasting the atmospheric environment within which severe local storms (severe thunderstorms and tornadoes) are likely to develop. It has also been applied to the problems of forecasting and investigating east coast cyclogenesis, upper level turbulence and shear, and boundary layer transport. Support for the model development has been provided by NASA/Goddard using the computational facilities of NASA/Langley (CYBER 203) and NASA/Goddard (CYBER 205)

2.3 History

The original version was implemented on a 500K word CDC STAR 100 Vector Processor at NASA/Langley in the late 70's using 64-bit FORTRAN. The availability of the SL/1 programming language at Langley, which permitted easy access to the 32-bit instruction set on the STAR 100, resulted in an effective doubling of the memory and the model was recoded with larger vectors. This allowed for an increase in the area over which the model was run while maintaining the same horizontal and vertical resolution.

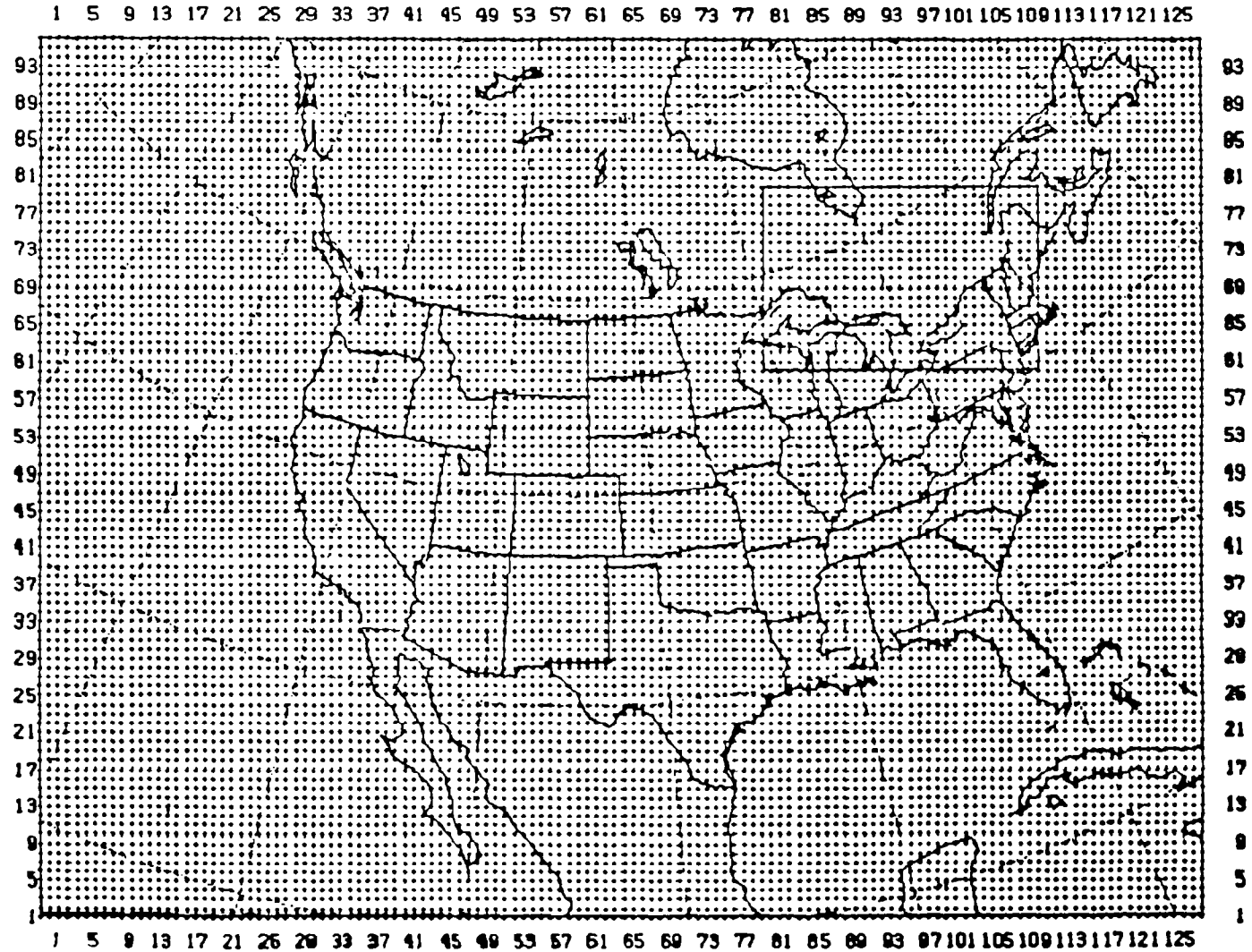


FIG. 1 DOMAIN OF COVERAGE BY MASS MODEL AND THE HORIZONTAL OF GRID POINTS DISTRIBUTION

TABLE 1 CHARACTERISTICS OF MASS MODEL

MASS (DESCRIPTION)

- o HYDROSTATIC PRIMITIVE EQUATIONS
- o TERRAIN FOLLOWING SIGMA-P COORDINATE
- o LIMITED AREA DOMAIN
- o CARTESIAN GRID ON A POLAR STEREOGRAPHIC MAP (ARAKAWA "A" GRID)
- o 4TH ORDER ACCURATE HORIZONTAL SPACE DIFFERENCING
- o 2ND ORDER ACCURATE VERTICAL SPACE DIFFERENCING
- o 2ND ORDER ACCURATE TIME DIFFERENCING
- o INITIAL DATA IS DERIVED FROM THE LFM ANALYSIS PLUS RAWINSONDES
- o INITIALIZATION IS BASED ON THE CALCULUS OF VARIATIONS
- o PHYSICS
  - LARGE SCALE PRECIPITATION
  - PLANETARY BOUNDARY LAYER
  - DRY CONVECTION
  - MOIST CONVECTION (UNDER DEVELOPMENT)
- o 50 KM GRID SPACING AT 45°N
- o 19 EQUALLY SPACED LAYERS
- o 128 X 96 COMPUTATIONAL DOMAIN
- o TIME DEPENDENT BOUNDARY CONDITIONS
- o COMPREHENSIVE INTERACTIVE DIAGNOSTIC PACKAGE ON THE FRONT END
  - VERTICAL PROFILES
  - VERTICAL CROSSECTIONS
  - CONSTANT PRESSURE SURFACES
  - TIME HISTORY
  - TRAJECTORIES
  - VERIFICATION STATISTICS

In the spring of 1980, the STAR 100 was upgraded to a 1m word CDC CYBER 203. The new machine effectively had twice the memory of the STAR 100. The area over which the model is run was again expanded and the vertical resolution was increased from 12 to 14 vertical layers.

In the spring of 1983, the model was transferred to the NASA/Goddard CYBER 205. The model was recoded in CDC FORTRAN 2.0 using 32-bit arithmetic. After being successfully benchmarked against the Langley version, the vertical resolution was again increased from 14 to 19 layers. The Goddard version of MASS on the CYBER 205 executes approximately 3 times faster than the Langley version on the CYBER 203. This can be explained by

1)Reduction in cycle time from 40 to 20 NS.

2)Linked triad instruction on the CYBER 205.

3)Faster gather/scatter instruction.

4)Coding differences.

3. EQUATION SET

The model utilizes a standard primitive equation set cast in a terrain following σ_p coordinate system. As indicated earlier, the forecasted variables are the 3-D distribution of wind, pressure, temperature and moisture. The basic prognostic equations are given below where u and v are x and y coordinate momentum, T is temperature, q is the moisture mixing ratio and Π is the pressure at the terrain minus the pressure at the top of the model.

$$\frac{\partial u}{\partial t} = -um\frac{\partial u}{\partial x} - vm\frac{\partial u}{\partial y} - \dot{\sigma}\frac{\partial u}{\partial \sigma} - m^2\frac{\partial \phi}{\partial x} - \sigma\alpha m\frac{\partial \pi}{\partial x} + fv + K\nabla^4 u + \frac{\partial u}{\partial t}\Big|_{Physics}$$

$$\frac{\partial v}{\partial t} = -um\frac{\partial v}{\partial x} - vm\frac{\partial v}{\partial y} - \dot{\sigma}\frac{\partial v}{\partial \sigma} - m^2\frac{\partial \phi}{\partial y} - \sigma\alpha m\frac{\partial \pi}{\partial y} - fu + K\nabla^4 v + \frac{\partial v}{\partial t}\Big|_{Physics}$$

$$\frac{\partial T}{\partial t} = -um\frac{\partial T}{\partial x} - vm\frac{\partial T}{\partial y} - \dot{\sigma}\frac{\partial T}{\partial \sigma} + \frac{\alpha}{C_p}\omega + K\nabla^4 T + \frac{\partial T}{\partial t}\Big|_{Physics}$$

$$\frac{\partial q}{\partial t} = -um\frac{\partial q}{\partial x} - vm\frac{\partial q}{\partial y} - \dot{\sigma}\frac{\partial q}{\partial \sigma} + K\nabla^4 q + \frac{\partial q}{\partial t}\Big|_{Physics}$$

$$\frac{\partial \pi}{\partial t} = \int_1^0 m\frac{\partial u\pi}{\partial x} + m\frac{\partial v\pi}{\partial y}\, d\sigma$$

NOTE: α = specific volume
f = coriolis parameter

Three diagnostic equations close the system and are given below where $\dot{\sigma}$ is the vertical velocity, ϕ is the geopotential energy and ω is the vertical velocity in pressure coordinates.

$$\dot{\sigma}_{k+1/2} = \dot{\sigma}_{k-1/2} + \left[\frac{\partial \pi}{\partial t} + m\frac{\partial u\pi}{\partial x}\Big|_k + m\frac{\partial v\pi}{\partial y}\Big|_k\right]\frac{\Delta\sigma}{\pi}$$

$$\phi_{k+1/2} = \phi_{k-1/2} + RT_k(1 + 0.61q_k)\ln\left(P_{k-1/2}/P_{k+1/2}\right)$$

$$\omega_k = \frac{\pi}{2}\left[\dot{\sigma}_{k+1/2} + \dot{\sigma}_{k-1/2}\right] + \sigma_k\left[\frac{\partial \pi}{\partial t} + um\frac{\partial \pi}{\partial x}\Big|_k + vm\frac{\partial \pi}{\partial y}\Big|_k\right]$$

The boundary conditions are

$$\dot{\sigma}_1 = \dot{\sigma}_0 = 0$$

$$\phi_{1/2} = \text{TERRAIN HEIGHT}$$

and the definitions for and π are

$$\sigma = \frac{P - P_{top}}{\pi} \qquad \pi = P_{sur} - P_{top}$$

the remaining variables are

m = mapscale grid transformation factor

c_p= specific heat at constant pressure

R= gas constant for dry air

P_{sur}= pressure at the terrain

P_{top}= pressure at the top of the model

χ= horizontal eddy diffusivity

4. GRID SYSTEM

The technique for solving the differential equations is to discretize the equations into finite difference form and solve them on a 3-D grid. The horizontal grid employed is the Arakawa "A" grid where all dependent variables are defined at all grid points. The vertical grid is staggered so that u, v, T and q represent layer averages defined at the mid-point of each layer and and are held at the layer interfaces. The third diagnostic variable, w, is held with u, v, T and q. This structure is represented in Fig. 2.

5. NUMERICAL TECHNIQUE

5.1 Horizontal Space Derivatives

The fourth order accurate finite difference approximation to an x-direction space derivative for an arbitrary variable ψ is given below

$$\left.\frac{\partial \psi}{\partial x}\right|_i \approx \frac{1}{12\Delta x}\left[8(\psi_{i+1}-\psi_{i-1})-(\psi_{i+2}-\psi_{i-2})\right] + O(\Delta x^4)$$

where i is a horizontal index. An analogous formula is used for y - direction derivatives.

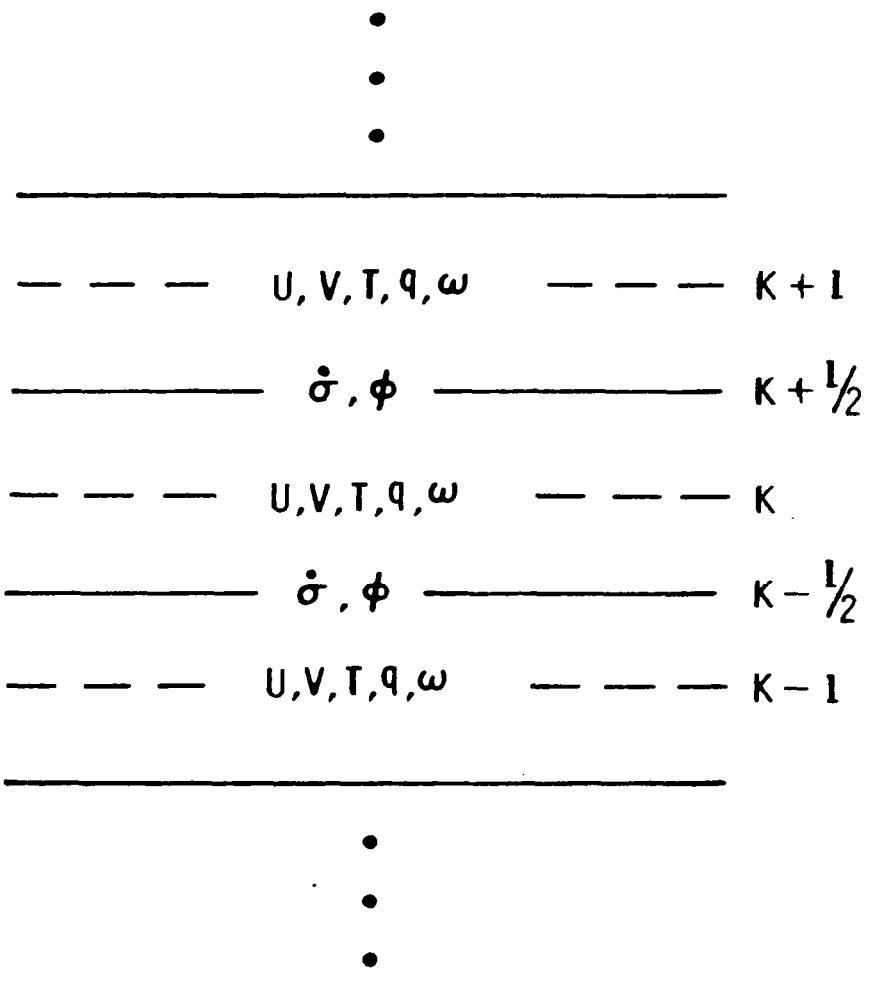


FIG. 2 VERTICAL GRID SYSTEM OF MASS

5.2. Vertical Space Derivatives

A second order accurate finite difference formula is used to approximate the vertical advection terms of the u,v, T and q prognostic equations. The representation, for an arbitrary variable ψ, is given below

$$\dot{\sigma}\frac{\partial\psi}{\partial\sigma}\Big|_k \approx \frac{1}{2\Delta\sigma}\left[\dot{\sigma}_{k+1/2}\left(\psi_{k+1}-\psi_k\right)+\dot{\sigma}_{k-1/2}\left(\psi_k-\psi_{k-1}\right)\right]$$

where k is a vertical index.

5.3 Time Derivatives

A second order accurate approximation to the time derivatives is used. The Euler-Backward Technique has the properties of frequency dependent damping and no computational mode. For an arbitrary variable ψ the finite difference representation is given as

$$\psi^* = \psi^n + \frac{\partial\psi^n}{\partial t}\Delta t \qquad \text{Prediction}$$

$$\psi^{n+1} = \psi^n + \frac{\partial\psi^*}{\partial t}\Delta t \qquad \text{Correction}$$

where n is a time level index and * refers to a intermediate time level.

This scheme requires the storage of only one time level of information (time level n) whereas other explicit schemes such as the Leapfrog Scheme requires the storage of at least two time levels (n and n-1). The penalty is that twice the computational work is required as compared with the Leapfrog scheme.

6. BASIC MEMORY REQUIREMENTS

As mentioned earlier, the Euler-Backward scheme for time marching the prognostic equations for the 3-D structure of wind, pressure, temperature and moisture requires the storage of only one time level of information. The * 'ed time level is an intermediate time level and only needs to be as deep (with respect to the vertical) as is required to solve the equations at a layer. It should be noted that only the vertical advection terms couple the model layers together and that to solve the equations at layer k requires the dependent variables at layers k+1, k and k-1. Therefore, the * 'ed time level only needs to be 3 deep (it holds the prediction values to be used during the correction step) and can be reused for the solution of each layer.

Given that the 19 model layers contain 128 x 96 grid points each, the basic memory required is

u (128, 96, 19)

v (128, 96, 19)

T (128, 96, 19)

q (128, 96, 19)

pi (128, 96)

ustar (128, 96, 3)

vstar (128, 96, 3)

tstar (128, 96, 3)

qstar (128, 96, 3)

pistar (128, 96)

If an additional layer were to be added only the u, v, T and q arrays would be increased. The ustar, vstar, tstar and qstar arrays are always dimensioned 3 deep and this is a function of the vertical advection terms which require 3 layers of storage to solve the equations.

In contrast, the Leapfrog scheme would require 2 sets of arrays dimensioned 128 x 96 x 19, therefore, there is a considerable memory savings with the Euler-Backward Scheme. A technique developed by Tuccillo (1983) shows some promise in reducing the computational work by increasing the premissable timestep.

7. METHOD OF SOLUTION

The method of solution is depicted in Fig. 3 and shows the sequence of steps required to solve the equations at all layers. Prediction is the step that advances the solution from the n to the * time level and correction is the step that advances the solution from the * to the n+1 time level. It there are NZ layers then there are 2*NZ number of steps required to advance the solution one time step. The number above each line represents the order of solution where the first step is to perform prediction for layer 1, the second step is prediction at layer 2, the third step is correction at layer 1 and so on. After correction (the 2*NZ step) at layer NZ is finished the solution has been advanced one time step.

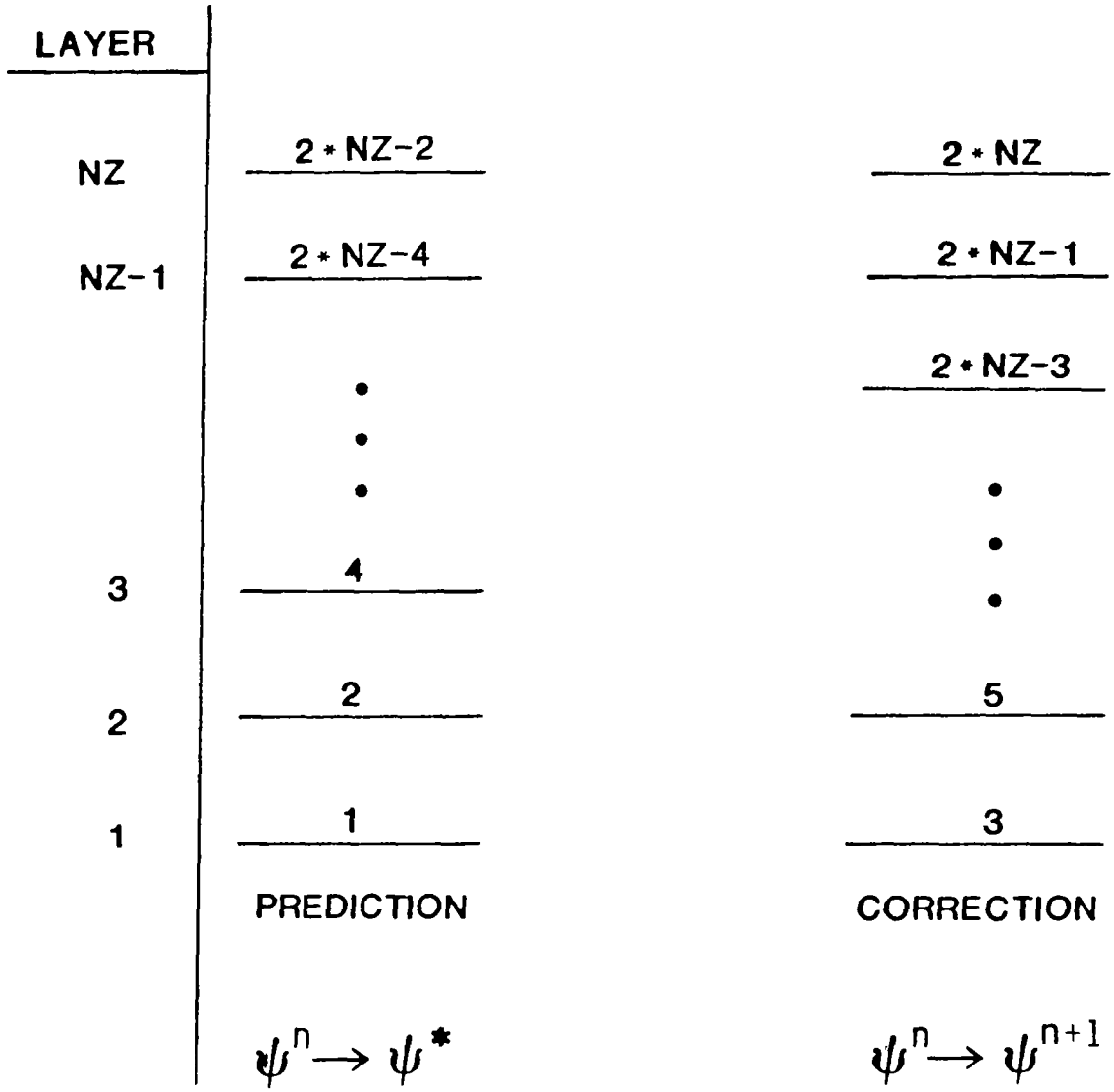


FIG. 3 SEQUENCE OF STEPS TO ADVANCE THE SOLUTION ONE TIMESTEP

The *'ed arrays are reused for each layer and the calculations for each layer are fully vectorized where the vector lengths are NX*NY or 12288. For this vector length the machine is computing at about 98% of its maximum rate.

8. BOUNDARY CONDITIONS

Since MASS is a limited area model, as opposed to a global model, the solution at the horizontal boundaries needs to be specified. The technique for specifying the boundary conditions consist of blending externally calculated values using a weighted average formula which is represented by

$$\frac{\partial \psi}{\partial t} = W \frac{\partial \psi}{\partial t}\bigg|_{INTERIOR} + (1-W) \frac{\partial \psi}{\partial t}\bigg|_{EXTERIOR}$$

where W = 0 on outer column and row

W = 0.333 on first column and row in

W = 0.666 on second column and row in

W = 1.0 on third column and row in

It should be pointed out that this technique produces an overspecification at the boundary and higher horizontal diffusion is required near the boundaries to control noise generation.

This technique is vectorized by holding the externally specified boundary tendencies in a vector and using the scatter instruction to expand them into the correct positions prior to computing the weighted average. This technique minimizes to amount of storage required.

9. PROGRAMMING TECHNIQUES

The code is completely vectorized in the horizontal. The average vector length is about 12000 which represents the number of horizontal grid points. There is a loop over the vertical layers.

Some specific techniques used during the coding are

- o 32-bit arithmetic

  Sensitivity tests have indicated that 32-bits provides enough precision. Using 32-bits effectively doubles the real memory and halves the execution time.

- o Explicitly Vectorized

  The code does not depend on automatic vectorization by the compiler. All descriptors are set up with DATA and ASSIGN statements. Special Q8 calls are used where required.

- o Diadic and Triatic Structure

  All vector statements are written in a diadic structure (triadic when linked triads are created) to minimize compiler generated dynamic space which may cause paging.

- o Subroutines are kept small enough so that the Register File is not overflowed.

  Subroutines which have more local variables then the size of the register file (approximately 200) can be inefficient since loads from memory must be executed. All subroutines are kept small enough so that the swap instruction can load all necessary local variables at entry.

- Parameter Statement Used for Vector Dimensions

  Vector dimensions are easily changed by changing parameter values. .

- Factoring of Equations to yield Linked Triads

  The sequence of instructions have been arranged to yield the maximum number of linked triads.

- Run Only in Real Memory

  No page faults are generated during the interative time marching.

- Vectors are Grouped on Large Pages

  All large vectors are placed in common and grouped on large pages using loader options.

- Bit Vectors vs. Gather/Scatter

  For those situations where control store or gather/scatter can be applied, an analysis using the nominal performance figures for each instruction was performed and the most CPU or memory efficient techniques was applied.

10. TECHNIQUES FOR REDUCING CPU TIME

A 24-hour simulation with the model requires 1312 timesteps. Each timestep requires the evaluation of 2*NZ natural logs (for 12288 grid points). This required approximately 22 mins of CPU time using the 32-bit FORTRAN VHALOG function. Since the range of arguments for the natural log function was known, a more efficient

technique was incorporated where the natural log was approximated with a series factored using Horner's Rule. The evaluation requires 11 vector instructions, nine of which are linked triads, and runs approximately 40 times faster than the FORTRAN intrinsic function. This technique reduced the CPU time spent evaluating natural logs to 30 secs.

Other techniques for reducing CPU time consist of approximating the ** FORTRAN function with series of square roots (square root in a hardware instruction) and inverting scalars to generate vector multiplies instead of vector divides.

The version of MASS implemented on the CYBER 205 at NASA/Goddard requires 13 large pages of memory and 15 minutes of CPU time (same as wall time) for a 24 hour simulation over the area depicted in Fig. 1.

11. EXAMPLE OF OUTPUT

MASS at Goddard features a comprehensive postprocessing system to produce output from the model for interpretation. The post processing system runs interactively and produces hard copies on a GOULD electrostatic plotter. Future versions of the postprocessing system will likely feature interactive color graphics which should greatly improve the usability of the modeling system as a research tool for studying atmospheric processes. Figs. 4-12 are examples of the output from three of the six postprocessing programs currently available.

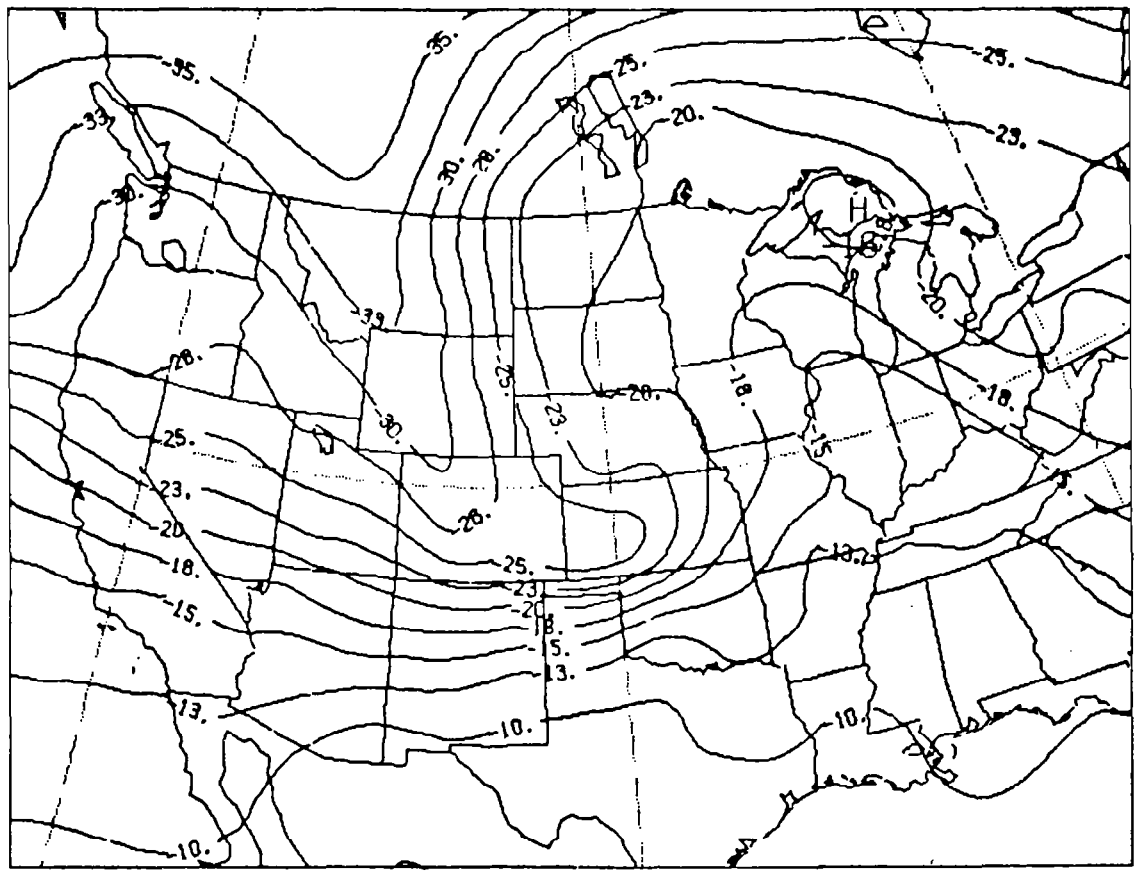


FIG. 4 MASS FORECASTS 500 MB TEMPERATURES (DEGREES CELCIUS)

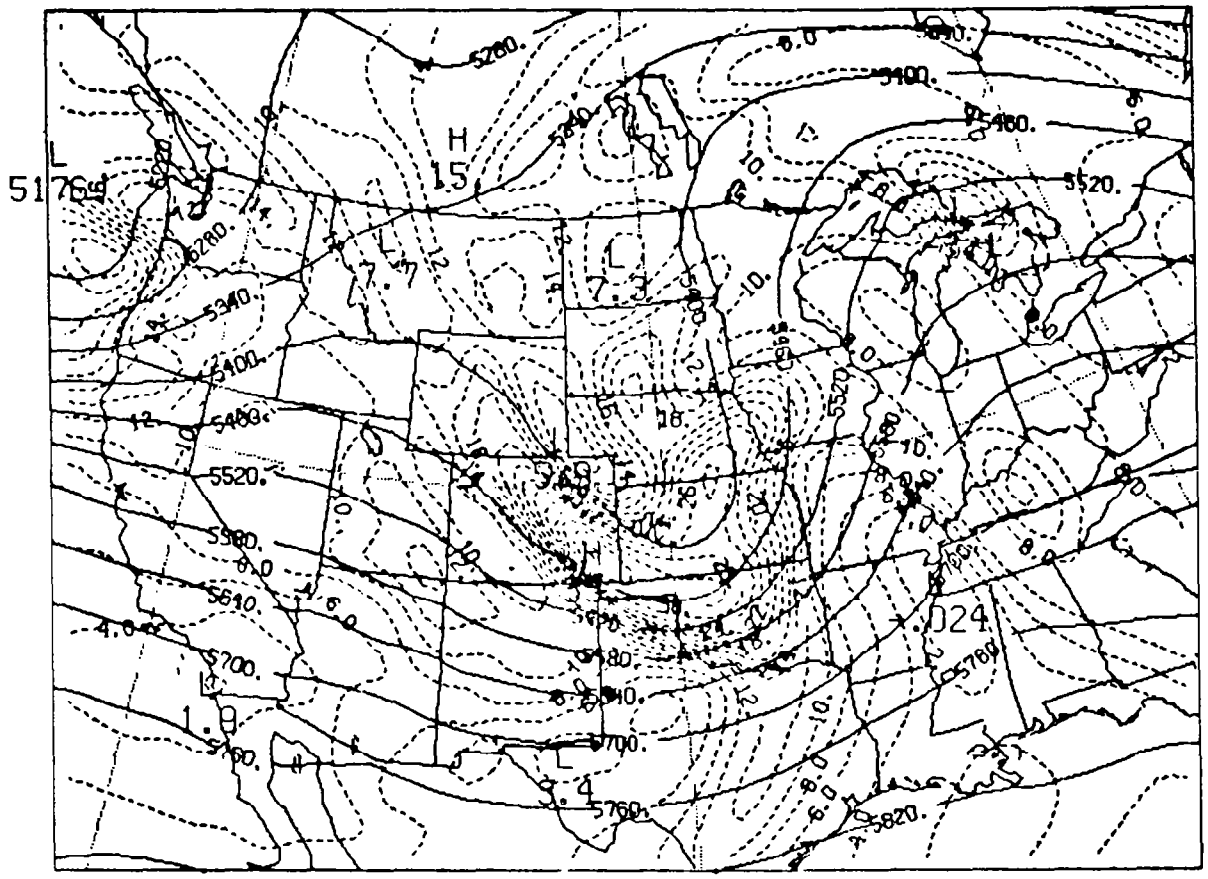


FIG. 5 MASS FORECASTED 500 MB HEIGHTS (METERS) AND VORTICITY (PER SECOND)

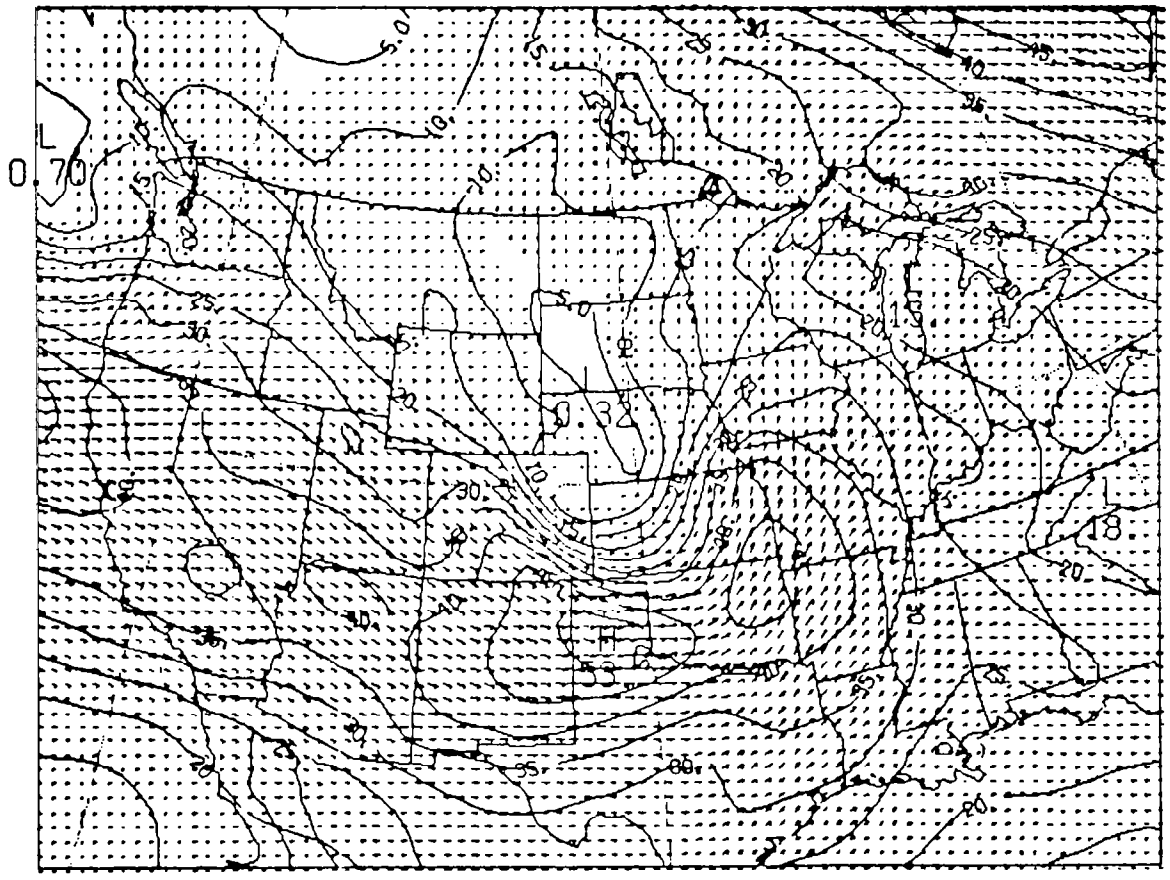


FIG. 6 MASS FORECASTED 500 MB WIND VECTORS AND ISOTACHS (METERS PER SECOND)

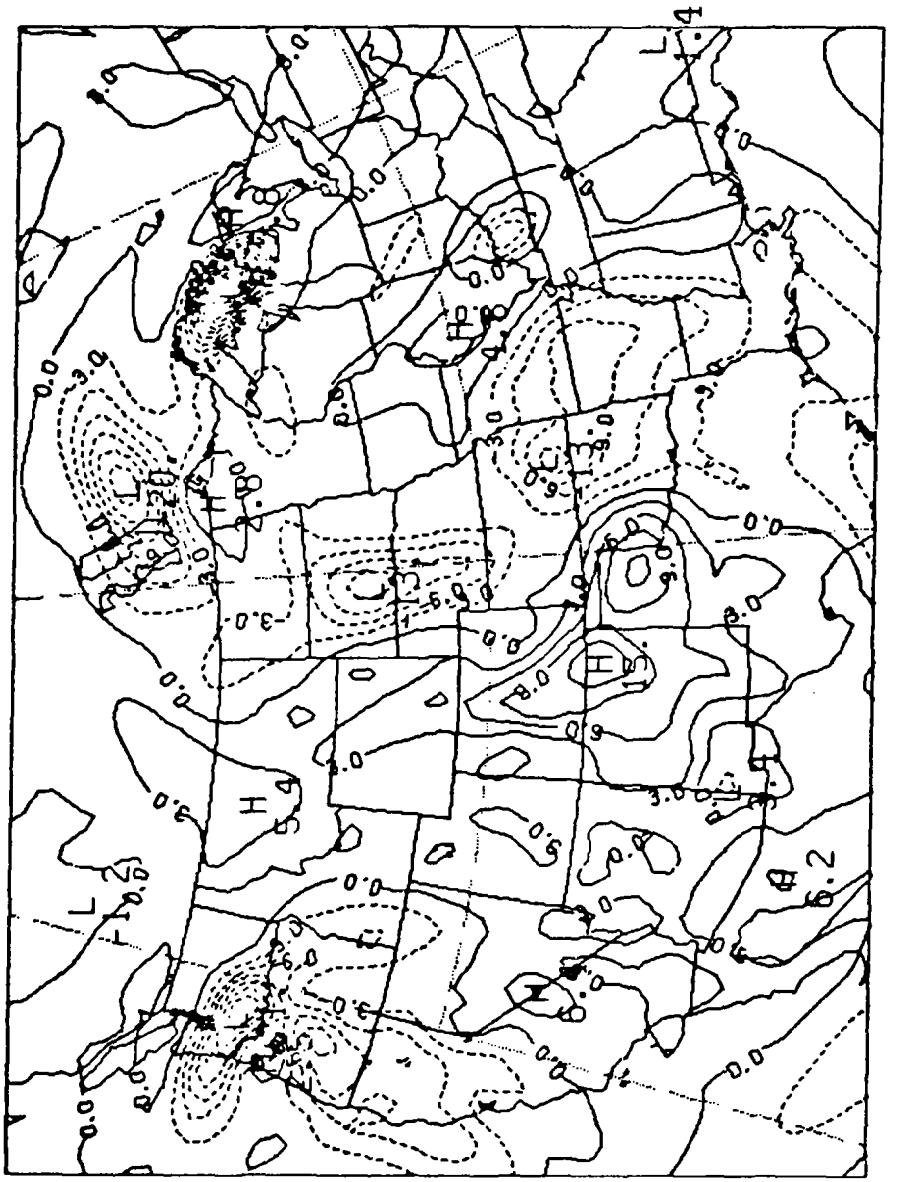


FIG. 7 MASS FORECASTED 500 MB VERTICAL VELOCITY
(MICROBARS PER SECOND)

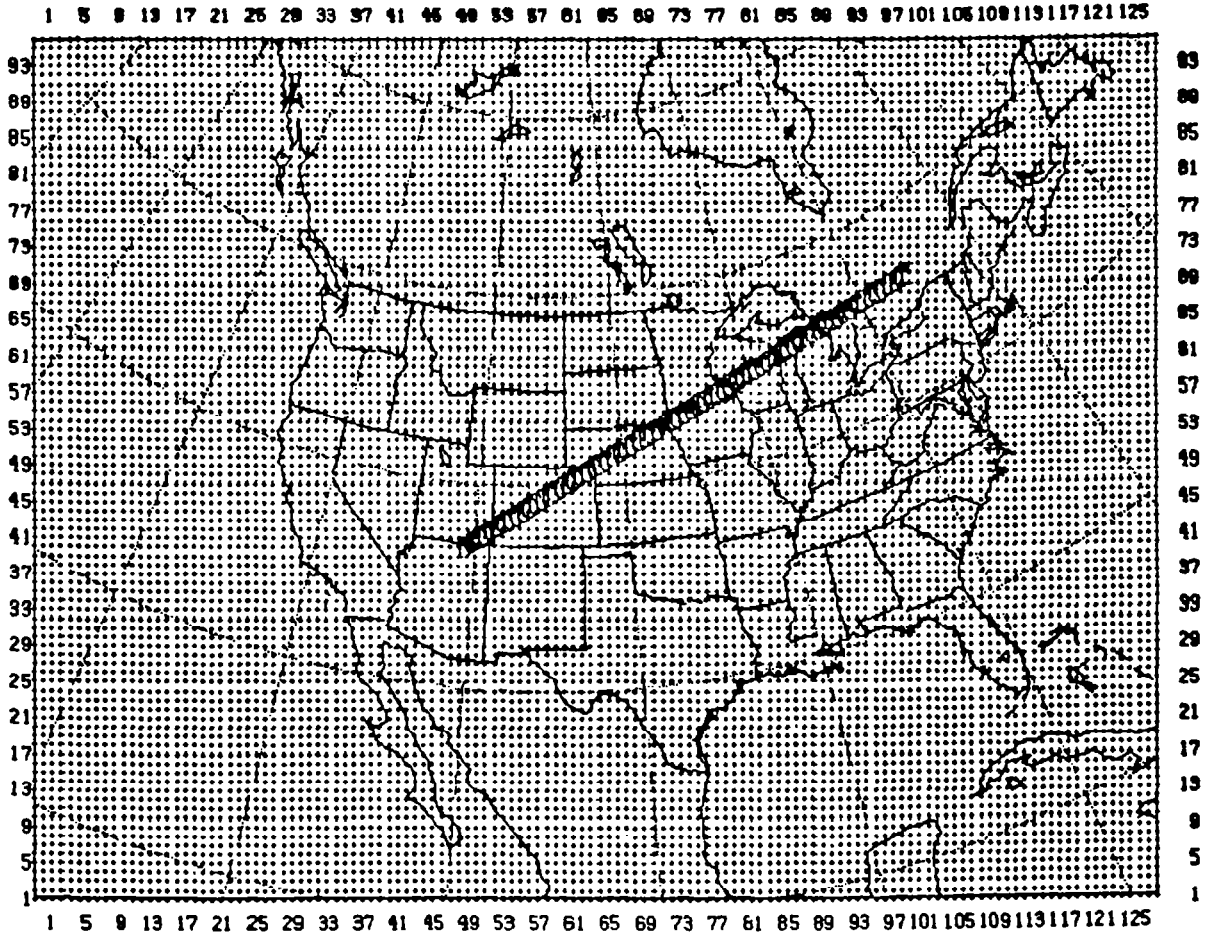


FIG. 8 VERTICAL CROSS-SECTION LOCATOR MAP

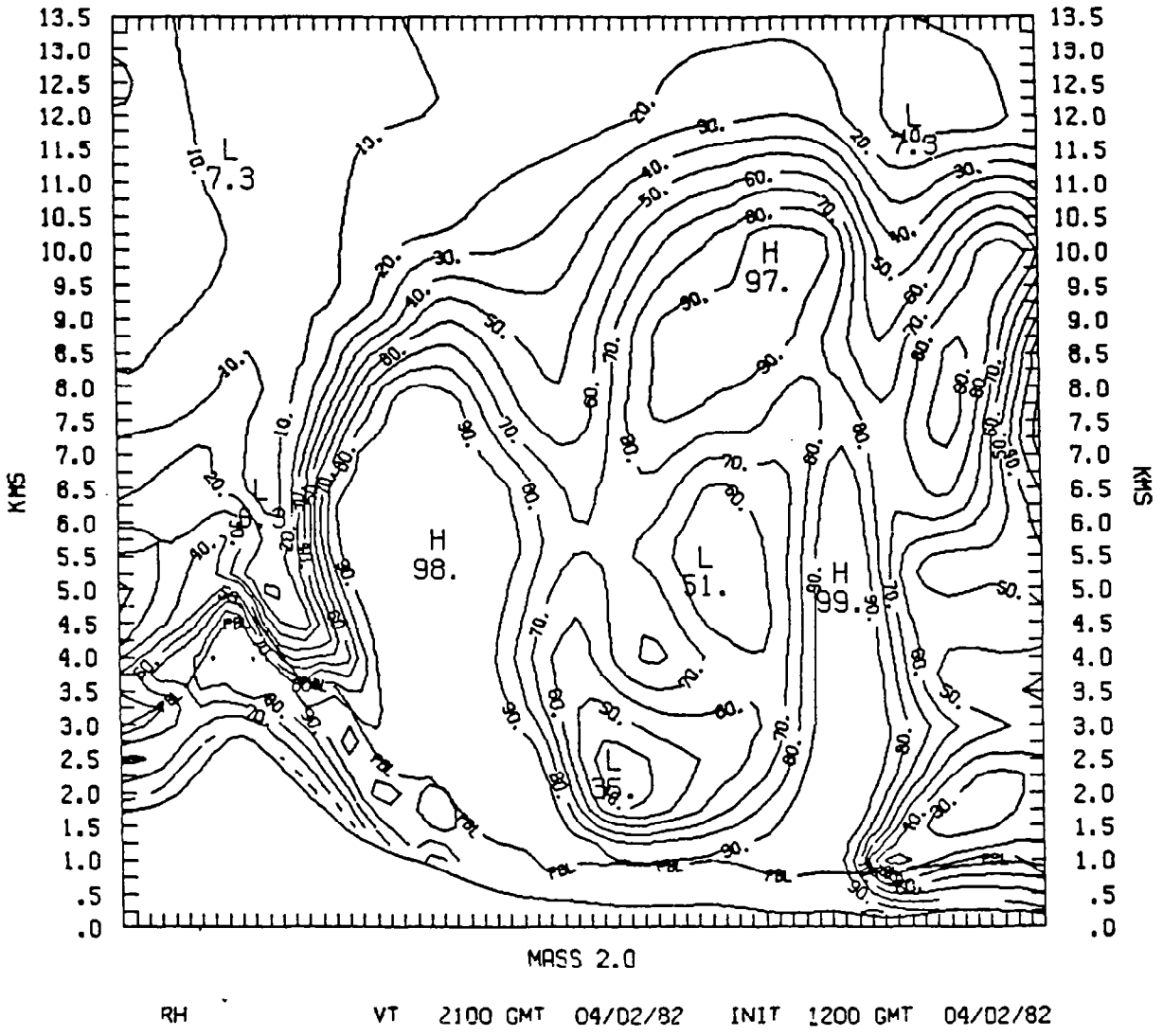


FIG. 9 MASS FORECASTED VERTICAL CROSS-SECTION OF RELATIVE HUMIDITY (PERCENT)

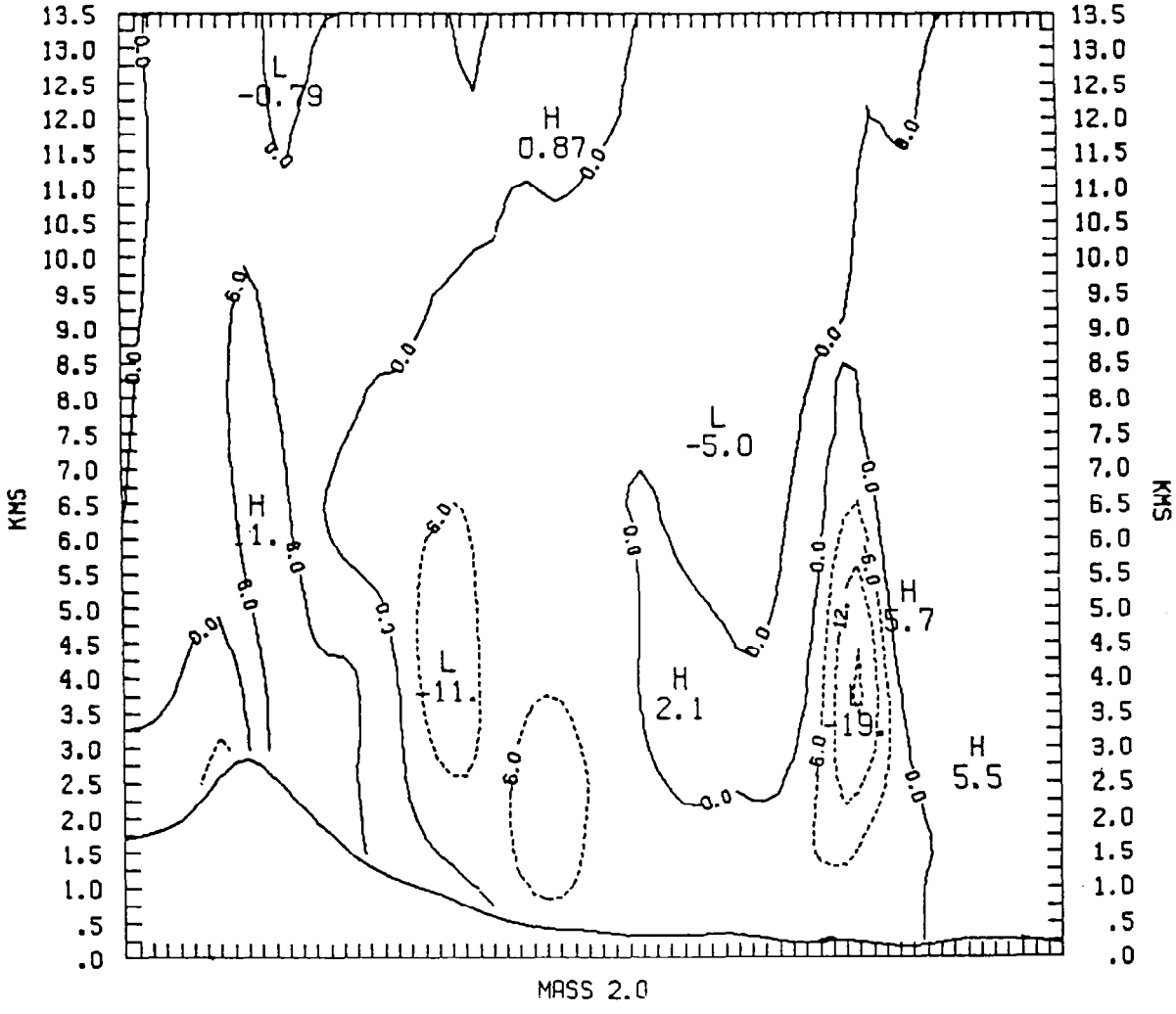


FIG. 10 MASS FORECASTED VERTICAL CROSS-SECTION OF VERTICAL VELOCITY (MICROBARS PER SECOND)

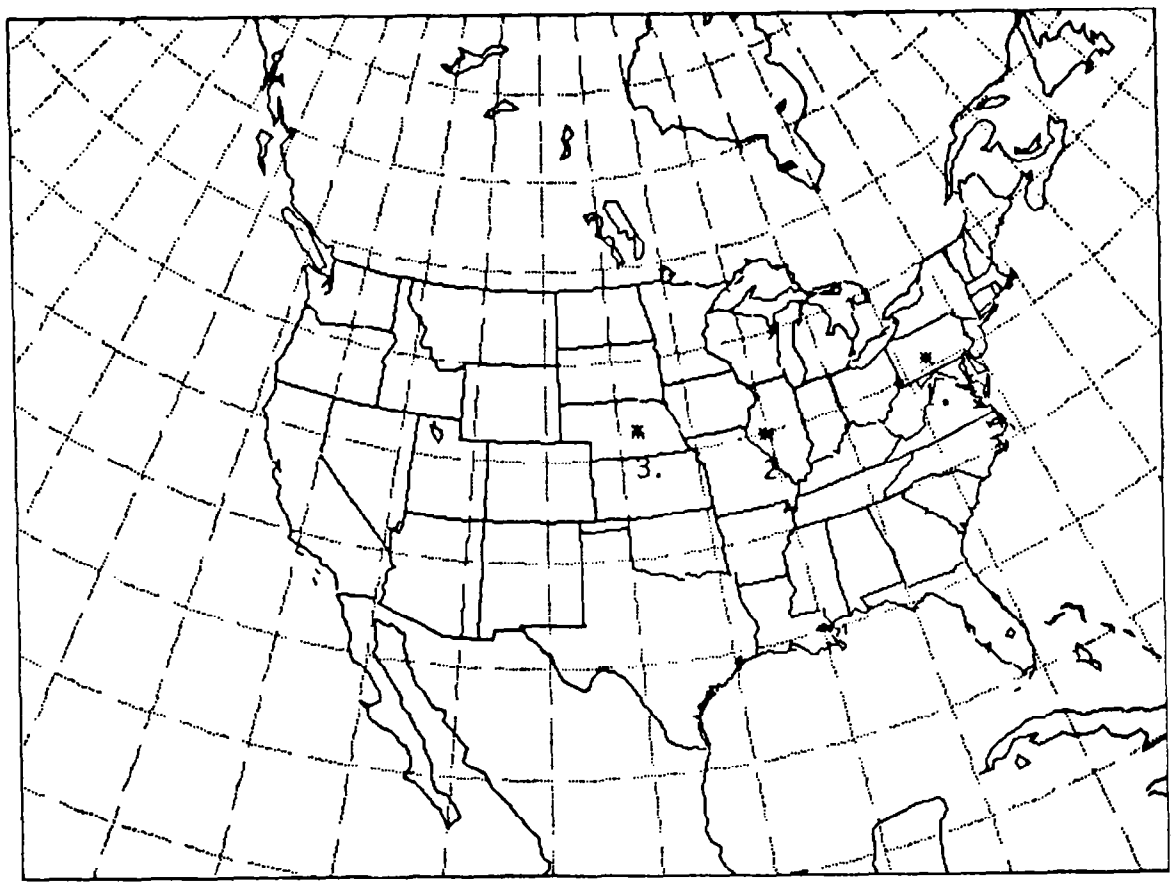


FIG. 11 SOUNDING LOCATOR MAP

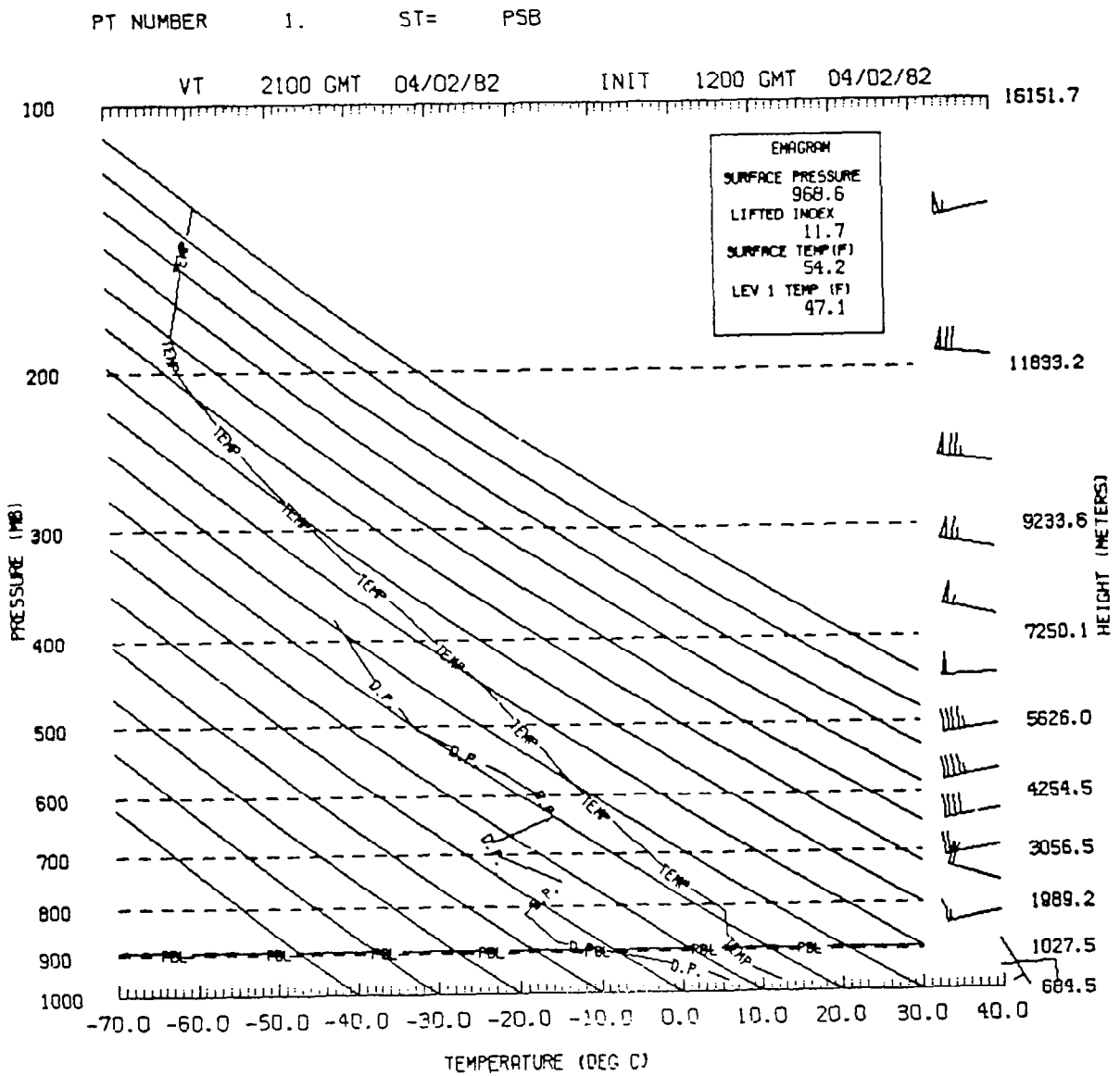


FIG. 12 MASS FORECASTED SOUNDING

12. REFERENCES

Kaplan, M.L., J.W. Zack, V.C. Wong, and J.J. Tuccillo, 1982: Initial Results from a Mesoscale Atmospheric Simulation System and Comparisons with the AVE-SESAME I Data Set. Mon. Wea. Rev., 110, 1564-1590.

Tuccillo, J.J., 1983: The Application of Pressure Gradient Force Averaging to the Euler-Backward Scheme. M.S. Thesis, Old Dominion University.

COMPUTER SIMULATION OF PROTEIN SYSTEMS

D. J. OSGUTHORPE,
P. DAUBER-OSGUTHORPE,
J. WOLFF,
D. H. KITSON
AND
A. T. HAGLER

THE AGOURON INSTITUTE

LA JOLLA, CALIFORNIA

Computer Simulation of Protein Systems

D. J. Osguthorpe, P. Dauber-Osguthorpe, J. Wolff, D. H. Kitson and A. T. Hagler
The Agouron Institute, 505 Coast Blvd. South, La Jolla, California 92037.

Introduction. Significant advances are being made in the theoretical treatment of the conformation and dynamics of biological molecules. Several recent convergent developments are responsible for opening up new fields of investigation. They include:

1. The development and application of powerful theoretical techniques taken from statistical physics such as Monte Carlo and molecular dynamics simulations to biological systems.
2. The development of powerful computational hardware such as the Cyber 205.
3. The development of interactive graphics systems.
4. The increasing availability of experimental structural and dynamic data such as the ever-growing data base of protein crystal structures, small peptide crystal structures and the structural and dynamic properties of these same molecules in solution.

These developments enabled us to undertake the project of studying ligand binding to dihydrofolate reductase (DHFR). This is an extremely important enzyme, as it is the target of several drugs (inhibitors) which are used clinically as antibacterials, antiprotozoals and in cancer chemotherapy.[1,2] DHFR catalyzes the NADPH (reduced nicotinamide adenine dinucleotide phosphate) dependent reduction of dihydrofolate to tetrahydrofolate, which is used in several pathways of purine and pyrimidine biosynthesis, including that of thymidylate.[3] Since DNA synthesis is dependent on a continuing supply of thymidylate, a blockade of DHFR resulting in a depletion of thymidylate can lead to the cessation of growth of a rapidly proliferating cell line.

DHFR exhibits a significant species to species variability in its sensitivity to various inhibitors. For example, trimethoprim, an inhibitor of DHFR, binds to bacterial DHFR's 5 orders of magnitude greater than to vertebrate DHFR's.[4,5] We were interested in studying the structural mechanics, dynamics and energetics of a family of dihydrofolate reductases to rationalise the basis for the inhibition of these enzymes and to understand the molecular basis of the difference in the binding constants between the species. This involves investigating the conformational changes induced in the protein on binding the ligand, the internal strain imposed by the enzyme on the ligand, the restriction of fluctuations in atom positions due to binding and the consequent change in entropy. X-ray crystallographic structures of DHFR from a few species, in complex with various ligands, are known,[6-8] as well as partial data about the structures in solution.[9-11] The availability of the structure, in the form of atomic coordinates for the enzyme system, is a prerequisite for performing any kind of energy calculations. In addition, due to the size of these systems as discussed below, only the availability of supercomputers such as the Cyber 205 make this project feasible.

Computational Techniques. The techniques we use to investigate the DHFR system all require the calculation of the potential energy of the molecular system. This potential energy is expressed in terms of an analytical representation of all internal degrees of freedom and interatomic distances, as in eqn. (1).

$$\begin{aligned} V = {} & \sum\{D_b[1 - e^{-\alpha(b-b_0)}]^2 - D_b\} + 1/2 \sum H_\theta(\theta - \theta_0)^2 \\ & + 1/2 \sum H_\phi(1 + s\cos n\phi) + 1/2 \sum H_\chi \chi^2 \\ & + \sum\sum F_{bb'}(b - b_0)(b' - b_0') \\ & + \sum\sum F_{\theta\theta'}(\theta - \theta_0)(\theta' - \theta_0') + \sum\sum F_{b\theta}(b - b_0)(\theta - \theta_0) \\ & + \sum F_{\phi\theta\theta'}\cos\phi(\theta - \theta_0)(\theta' - \theta_0') + \sum\sum F_{\chi\chi'}\chi\chi' \\ & + \sum\epsilon[2(r^*/r)^9 - 3(r^*/r)^6] + \sum q_i q_j / r \end{aligned} \tag{1}$$

This type of representation of the potential energy in terms of the internal (valence) degrees of freedom is called a Valence Force Field. Such valence force fields have long been used in vibrational spectroscopy in order to carry out normal mode analysis.[12] Basically the terms in equation (1) express

the energies required to deform each internal coordinate from some unperturbed "standard" value denoted by the subscript "0". The first term is a Morse potential which describes the energy required to stretch each bond from its relaxed value, b_0. The second term represents the energy stored in each valence angle when it is bent from its "standard" value, θ_0. The third term represents the intrinsic energy required to twist the molecule about a bond by a torsion angle, ϕ. The fourth term represents the energy required to distort intrinsically planar systems by χ from their planar conformation, i.e. the out of plane term. The next terms represent various couplings between internal coordinates, which are known to be necessary from studies of vibrational spectra.[13] They are the bond-bond, angle-angle, bond-angle, angle-angle-torsion and out of plane cross-term respectively. The last 3 terms describe the exchange repulsion, dispersion and coulombic interactions that occur between non-bonded atoms.

The parameters D_b , H_θ , H_ϕ , H_χ , and F_{ij} are the force constants for the corresponding intramolecular deformation, r^* and ϵ characterize the size of the atoms and the strength of the van der Waals interaction between them, while the q_i are the partial charges carried by each atom. The parameters for the functions were derived from fitting a wide range of experimental data including crystal structure, unit cell vectors and the orientation of the asymmetric unit, sublimation energies, molecular dipole moments, molecular structure, vibrational spectra and strain energies of small organic compounds.[14-19] Ab-initio molecular orbital calculations have also been used in conjunction with the experimental data to give information on charge distributions, energy barriers and coupling terms, both to supplement and confirm the results obtained from the experimental data.[20, 21]

Minimisation. Given the analytical representation of the potential energy in eqn. (1), we can minimize this energy with respect to all internal degrees of freedom, i.e. solve the equation

$$\partial E/\partial x_i = 0 \qquad i = 1, 3n \tag{2}$$

where the x_i are the cartesian coordinates of the molecule.

The minimisation results in the "minimum energy structure" of the system. Analysis of the minimum energy structure reveals the basic structural features of the system along with the interatomic forces underlying this minimum energy conformation. At the minimum, we can take second derivatives of the energy and construct the mass weighted second derivative matrix. From the eigenvalues of this matrix the vibrational frequencies may be obtained and the normal modes from the eigenvectors.[22] The conformational entropy of the system can now be calculated from the vibrational frequencies using the Einstein relations.[23] The conformational entropy of a system plays an important role in both conformational equilibria and binding.[24]

Molecular dynamics. Molecular dynamics is the numerical integration of Newtons classical equations of motion. Having specified the potential, we define the initial conditions of the system, the coordinates of the protein, inhibitor, solvent and a set of initial velocities. Once the initial conditions are given, Newtons equations of motion

$$-\delta V(\bar{r}_i \cdots \bar{r}_n)/\delta \bar{r}_i = \bar{F}(\bar{r}_i \cdots \bar{r}_n) = m_i d^2\bar{r}_i/dt^2 \tag{3}$$

are integrated forward in time, in order to compute the atomic trajectories $\bar{r}_i$ (t)...$\bar{r}_n$ (t) as functions of time. The forces are calculated from the energy expression in eqn. (1) by taking analytical derivatives. We then take a small time step, Δt, of $\approx 10^{-15}$ sec. and applying the acceleration as calculated from Newtons law (eqn. 3), we update the velocity and position of each atom, to a new velocity and position using a Gear[25] predictor-corrector algorithm or a Verlet algorithm.[26] The forces and acceleration at the new positions are then calculated and we repeat the procedure, thus tracing the trajectories of the atoms.

Calculations on the Cyber. One of the systems we are studying, the E. coli DHFR-Trimethoprim complex, is the system we have been using to develop the programs on the Cyber 205. Table I lists the no. of atoms, internal coordinates and non-bond interactions for this system, to demonstrate the

magnitude of the calculation involved.

Table I

E. coli Dihydrofolate Reductase System

	atoms
E. coli Dihydrofolate Reductase	2490
Trimethoprim	40
155 Waters	465
	2995

Internal Coordinates

Bonds	2875
Valence Angles	4785
Torsion Angles	6784
Bond-Bond cross-terms	4785
Bond-Angle cross-terms	9570
Angle-Angle cross-terms	7584
Angle-Angle-Torsion cross-terms	6784
Non-bond pairs	$\approx$1,600,000

Minimisation and molecular dynamics both require computing the energy using eqn. (1), changing the coordinates and repeating this process many times. Note that each energy calculation involves evaluating the appropriate terms in eqn. (1) for each of the internals listed in table I. Thus the last three terms in eqn. (1) need to be evaluated for each of the 1,600,000 non-bonded pairs. As the time required to compute the change in the coordinates once the energy has been calculated is small, the time required to calculate the energy determines the time to perform the minimisation, or how many steps of dynamics can be done. For a minimisation the number of iterations depends on how close to zero we require the derivatives, for a conjugate gradient minimiser previous experience indicates that about 3 times the number of atoms iterations are required to get derivatives to less than 0.05 kcal/molÅ, which is about 10,000 iterations for the protein. In molecular dynamics we would like to simulate at least 100 picoseconds, preferably a nanosecond, as this is still a very short time compared to molecular events such as binding. This requires 100,000 iterations at a 1 femtosecond timestep. Thus the speed with which the energy calculation is carried out is crucial.

Non-bond interaction calculation. Table II shows the timings of the energy routines used to compute eqn. (1) on the VAX 11/780 and the Cyber 205 for the Dihydrofolate Reductase system. The non-bond part of the calculation takes by far the major portion of the CPU time, 78% of the iteration time on the VAX, so this was vectorised first. The routine computes the non-bond energy, see eqn. (1), by calculating the interaction between all pairs of atoms, except for bonded atoms and 1-3 interactions. For a 10Å cutoff this is $\approx 1.6\times 10^6$ pairs, which is the reason this is the major time consuming portion of the energy calculation. This was implemented on the VAX by a residue neighbour list in

Table II

Comparison of the Timing of Energy Calculation routines for 1 Iteration

Routine	VAX 11/780	CYBER Vectorised Large Pages
Bonds	2.42	0.055
Valence Angles	9.06	0.13
Torsion Angles[2]	30.69	0.55
Bond-bond	5.25	0.14
Bond-Angle	11.9	0.25
Angle-Angle	16.55	0.17
Out of Plane	2.35	0.10[1]
Non-Bond	448.98	1.23
Iteration Timing[2]	573.58	2.7

1. The out of plane routine is not vectorised.
2. The iteration timing is slightly larger than the sum of all the individual routine timings as it includes the time for the minimisation routine itself.

which for each residue a list of all the residues it interacts with is stored. This neighbour list is set up prior to the non-bond calculation and has to be recalculated every so often if a cutoff is used. In the non-bond calculation a loop is performed over all the residues and for each residue the interactions of all atoms in it with all atoms of the residues in the neighbour list of this residue are computed. This routine was vectorised by calculating the interaction of 1 atom with all its neighbouring atoms as vector operations. This gives vector lengths of up to 1000 for a 10Å cutoff. A bit vector with the length of the number of atoms in the molecule is set up for each atom which indicates whether an atom interacts with this atom or not. This is a large array, $N^2/2$, where N is the number of atoms, but because of the bit addressing capability of the Cyber 205 this only takes up 70,000 words in memory. The performance improvement of this routine after vectorisation is 365 over the VAX, which includes the intrinsic scalar speed of the Cyber 205, some 14 times faster than the VAX. The vectorisation of the non-bond routine took approximately 1 month.

Valence energy calculation. The valence energy and cross-term routines take $\approx$20% of the iteration time on the VAX. These routines were vectorised next, starting with the torsion angle routine which is the next major time consuming routine, 6% of the iteration time on the VAX. The bond, valence angle and torsion angle routines already used a list of the internals in the VAX version. These were all vectorised by creating vectors for the bonds, valence angles and torsion angles, which gives vector lengths from 3000 to 9000 for the dihydrofolate reductase system, see table I. These vectorisations resulted in performance improvements of 37 to 90 over the VAX in these routines.

To date we have achieved a net gain in speed over the VAX 11/780 of 212 for the enzyme simulation study described above.

References

1. J.R. Bertino, *Arthritis and Rheumatism*, **79**, 16 (1973).
2. J. Bertino and D. Johns, in *Cancer Chemotherapy*, ed. I. Brodsky, vol. 2, p. 9, Grune and Stratton. New York (1972).
3. M. Osborn and F.M. Huennekens, *J. Biol. Chem.*, **969**, 233 (1958).
4. J. Burchall and G.H. Hitchings, *Mol. Pharmacol.*, **1**, 126 (1965).
5. B.R. Baker, in *Medicinal Chemistry 3rd ed.*, ed. A. Burger, Wiley-Interscience, New York (1970).
6. D.A. Matthews, R. Alden , J.T. Bolin, S.T. Freer, R. Hamlin, N. Xuong, J. Kraut, M. Poe, M. Williams, and K. Hoogstein, *Science*, **197**, 452 (1977).
7. D.A. Matthews, R.A. Alden, J.T. Bolin, D.J. Filman, S.T. Freer, R. Hamlin, W.G.J. Hol, R.L. Kisliuk, E.J. Pastore, L.T. Plante, N. Xuong, and J. Kraut, *J. Biol. Chem.*, **253**, 6946 (1978).
8. D.A. Matthews, R.A. Alden, S.T. Freer , N. Xuong, and J. Kraut, *J. Biol. Chem.*, **254**, 4144 (1979).
9. G. C. K. Roberts, J. Feeney, A. S. V. Burgen, V. Yuferov, J. G. Dann, and R. Bjur, *Biochemistry*, **13**, 5351 (1974).
10. J. Feeney, B. Birdsall, J. P. Albrand, G. C. K. Roberts, A. S. V. Burgen, P. A. Charlton, and D. W. Young, *Biochemistry*, **20**, 1837 (1981).
11. A. Gronenborn, B. Birdsall, E. Hyde, G. Roberts, J. Feeney, and A. Burgen, *Mol. Pharmacol.*, **20**, 145 (1981).
12. O. Ermer, *Structure and Bonding*, **27**, 161, Berlin (1976).
13. S. Califano, *Pure Appl. Chem.*, **18**, 353 (1969).
14. A.T. Hagler, E. Huler, and S. Lifson, *J. Am. Chem. Soc.*, **96**, 5319 (1974).
15. A.T. Hagler and S. Lifson, *J. Am. Chem. Soc.*, **96**, 5327 (1974).
16. A.T. Hagler, S. Lifson, and P. Dauber, *J. Am. Chem. Soc.*, **101**, 5122 (1979).
17. A.T. Hagler, P. Dauber, and S. Lifson, *J. Am. Chem. Soc.*, **101**, 5131 (1979).
18. P. Dauber and A.T. Hagler, *Accts. of Chem Res.*, **13**, 105 (1980).
19. P. Dauber-Osguthorpe, J. Wolff, and A. T. Hagler. work in progress
20. S. Lifson, A.T. Hagler, and P. Dauber, *J. Am. Chem. Soc.*, **101**, 5111 (1979).
21. A.T. Hagler and A. Lapiccirella, *Biopolymers*, **15**, 1167 (1976).
22. E.B. Wilson, J.C. Decius, and P.C. Cross, in *Molecular Vibrations*, McGraw Hill, New York (1955).
23. T.L. Hill, in *An Introduction to Statistical Thermodynamics*, Addison-Wesley, Reading, Mass. (1960).
24. P. Dauber , M. Goodman, A.T. Hagler, D.J. Osguthorpe, R. Sharon, and P.S. Stern, *Proc. of the ACS Symposium on Supercomputers in Chemistry*, **173**, 161 (1981).
25. C.W. Gear, *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice Hall, Englwood Cliffs, N.J.
26. L. Verlet, *Phys. Rev.*, **159**, 98 (1967).

VLSI CIRCUIT SIMULATION USING A VECTOR COMPUTER

STEPHEN K. MCGROGAN

CONTROL DATA CORPORATION

OAKLAND, CALIFORNIA

VLSI CIRCUIT SIMULATION USING A VECTOR COMPUTER

Stephen K. McGrogan
Senior Staff Consultant

Control Data Corporation
1 Kaiser Plaza, Suite #301
Oakland, California, 94612

ABSTRACT

Simulation of circuits having more than 2000 active devices requires the largest, fastest computers available. A vector computer, such as the CYBER 205, can yield great speed and cost advantages if efforts are made to adapt the simulation program to the strengths of the computer.

ASPEC and SPICE (1) are two widely used circuit simulation programs. ASPECV and VAMOS (5) are respectively vector adaptations of these two simulators. They demonstrate the substantial performance enhancements possible for this class of algorithm on the CYBER 205. ASPECV is in use at ISD. VAMOS is in daily production use at MOSTEK.

INTRODUCTION

Over the past decade, the design of integrated circuits has become increasingly complex. Manufacturers who once had special purpose circuits of only a few dozen components now have microprocessors and random access memory chips constructed of thousands of devices. While early circuits were readily designed and debugged by hand, the more complex circuits have necessitated computer assistance.

During one phase of computer aided design, circuit simulation programs are used. These programs are given circuit interconnection information (nodes) and device characterizations (models). After establishing initial current and voltage conditions at time zero, they simulate circuit operation by evaluating device conductances and node voltages over small increments of time. Due to the rapid response of microcircuitry to voltage changes, circuit simulation must often be performed at timesteps of a few hundred picoseconds. This small timestep may necessitate thousands of steps to simulate circuit performance for a given set of initial inputs. Many such simulations (which may each require hours on an IBM 3081 or CDC 176) are required to thoroughly explore a circuit's characteristics over a wide range of temperatures and input sets.

The speed of a supercomputer is valuable to engineers designing such large scale integrated (VLSI) circuits. These engineers are, however, unwilling to compromise simulation accuracy for speed. For this reason, various projects have investigated vector computers (2) (3) (4) for use in the transient analysis of VLSI circuits.

Two well-known and widely used circuit simulators are ASPEC, copyrighted by Mr. Frank Jenkins, and SPICE, copyrighted by the Regents of the University of California. ASPECV is the product of a technical team from the San Francisco District of Control Data Corporation Professional Services Division. This team spent approximately one man-year analyzing ASPEC in detail. Their effort included extensive conversations with the program's author and the rewriting of select areas of code for enhanced performance.

The program VAMOS was developed by Steven D. Hamm and Steven R. Beckerich of MOSTEK Corporation. VAMOS evolved from a simple installation of SPICE2 into a program in which 80 percent of the analysis routine code is vectorized. Many sections of code were radically changed due to the application of algorithmic, rather than simple syntactic, vectorization.

ARCHITECTURAL CONSIDERATIONS

ASPEC AND SPICE were initially developed for a type of computer similar to the Control Data Corporation 6400. Originally, the programs were designed to handle circuits with fewer than 600 devices. Intentional minimization of memory requirements increased central processor time. Many users modified ASPEC and SPICE for use with large-scale circuits, extending the programs into areas far beyond their design. When any design is so overextended, there are often undesireable consequences. One obvious consequence was long running time on circuits with more than 2,000 devices.

Optimum performance for both ASPEC and SPICE required retailoring program design to fit the architecture of the CYBER 205. The Cyber 205 used has two vector pipes, a 16 megabyte memory, and is capable of 200 million floating point operations per second (Megaflops) on 64 bit operands. To maximize performance, the characteristics of this hardware must be considered. Some major considerations are:

1. The CYBER 205 defines a vector as contiguous memory locations. While ASPEC has a compatible memory organization, SPICE2 linked list storage needs re-organization.

2. The scalar functional units on the CYBER 205 are pipelined. Code that cannot be vectorized can be optimized by taking advantage of inherent parallelism. Even so, the performance of scalar code will probably be substantially less than the theoretical maximum of 50 Magaflops.

3. The hardware can generate and use bit vectors, which are useful in vectorizing loops containing conditional statements. These bit vectors aid in producing routines that have no scalar code and run at full vector speed.

4. The virtual memory of the CYBER 205 provides over 2 trillion words of user memory space. Any program that repetitively uses more than the entire physical memory may, however, generate a great amount of paging delay. This fact constrains the choice of algorithms, as a fast algorithm may require additional memory.

PROGRAM DESIGN

Both ASPEC and SPICE perform their simulations by alternating modeling routines with a current matrix solution routine. The modeling routines calculate the new device conductances based on device operating points. There is one model for each type of device, such as diodes, jfets, mosfets, and bi-polar transistors. One model must simulate many different operating modes and consequently has many branches and special cases.

The matrix solution routine calculates branch currents based on the conductances calculated by the modeling routines. From these currents new node voltages are obtained. This routine uses sparse Gaussian Elimination techniques. The time required by this routine grows very rapidly and non-linearly with circuit complexity.

In SPICE, to best utilize the long vector capabilities of the CYBER 205, an interface routine was written between the vectorized analysis routines and the rest of SPICE2. This routine reorganized memory into contiguous vectors and established new element pointers. ASPEC was similarly treated. The task was less formidable as data was already in homogeneous arrays.

In both VAMOS and ASPECV, vectorization of device equations is done by long vector operations with conditional stores for the results. All devices are evaluated in all regions of operation and the results are masked together to form composite result vectors. This technique avoids the data motion overhead characteristic of other methods at a cost of extra operations in each region. For VAMOS, the data given in Table 1 shows the tremendous advantage vectorization provides. The small amount of scalar store code remaining in MOSFET contributes 19.4 of the total 25.5 seconds.

ROUTINE	SCALAR	VAMOS	RATIO
LOAD	19.9	1.8	11.1
DIODE	79.4	3.6	22.1
MOSFET	325.4	25.5	12.8

Table 1. VAMOS Routine Comparisons

In VAMOS, the vector startup time required by the CYBER 205 caused the rejection of a vectorized matrix solution method for subcircuits as used in the program CLASSIE (2). Instead, effort was expended in scalar code optimization to achieve maximum instruction overlap. As part of the preprocessing phase of the program, the row-column lookup is performed once and the indices are stored in an auxiliary array.

In addition to the VAMOS techniques, ASPECV's routine EQNSOL detects perfect alignment between rows in the matrix. As circuit size increases, the number of such rows increases dramatically. Full row-length linked triads are executed in this case.

PROGRAM PERFORMANCE

Table 2 illustrates a comparison between a scalar version and VAMOS. The scalar version was already heavily optimized. The circuit tested contained 2256 mosfets, 1312 diodes, 1774 resistors and capacitors, and had 1429 equations with 98.9 percent matrix sparcity. Overall VAMOS performance was 3 times scalar, with 4 times in transient analysis. VAMOS performed the analysis over 100 times faster than a VAX-11/780.

ROUTINES	SCALAR	VAMOS
READIN	68.4	51.9
SETUP	34.7	22.7
DC SOLUTION	47.8	19.0
TRANSIENT	503.8	126.4
OUTPUT	5.6	5.6
TOTAL	660.3	225.9

Table 2. VAMOS Program Performance Comparison

Table 3 shows the characteristics of a series of flexible circuits which can be made any size by repeating a basic circuit block. Resistors and capacitors are also present but are irrelevant to modeling time. Table 4 gives execution time for two processors running ASPEC, and the current version of ASPECV on the CYBER 205. It is projected that, with continued effort, for large circuits the CYBER 205 mosfet run times could be reduced by another factor of 2 to 3. Table 5 shows that the time to model a given device decreases with increasing circuit size, a very desireable characteristic for VLSI circuitry.

CIRCUIT	DIODES	MOSFETS	NODES	MATRIX
1	50	50	30	119
2	100	100	54	220
4	200	200	102	470
8	400	400	182	860
16	800	800	358	1718
32	1600	1600	718	3473

Table 3. Circuit Characteristics

CIRCUIT	TIME STEPS	UNIVAC 1182	CDC 176	CDC 205
1	420	30	6	3
2	622	82	16	6
4	869	208	42	15
8	1658	697	141	40
16	1658	1421	301	76
32	1658	TOO BIG	TOO BIG	158

Table 4. ASPEC/ASPECV Comparison

CIRCUIT	AVERAGE TIME (micro-secs) diode	mosfet	EFFECIENCY	VECTOR
1	9.7	39		50
2	7.1	32		66
4	5.8	28		80
8	5.2	26		89
16	4.7	25		94
32	4.5	24		97

Table 5. ASPECV Size/Efficiency

Since most circuit simulation runs produce a great deal of printed output, current simulations using ASPECV spend the majority of their time in Fortran I/O. As an example, one ASPECV circuit containing 1000 devices and 950 nodes initially ran in 980 seconds on a UNIVAC 1182 and in 141 seconds on the CYBER 205. After optimizing everything but the diode and mosfet models, the same circuit required 72 seconds on the 205. Of the 72 seconds, 39 were spent in the models. ASPECV requires only 44 seconds to simulate the same circuit. Only 6.3 seconds are required in the models: 1.3 in diodes, 5.0 in mosfets. Although the mosfet model is still several times slower than theoretically possible, further effort would yield small returns indeed. The simulation mentioned spends over 66 percent of its time in Fortran I/O routines.

CONCLUSION

Program speedups of 3 to 4 were accomplished through vectorization. Future work directed at vectorization of the remaining scalar code may result in a similar speed increase. Fortran I/O provides an effective limit to maximum attainable speed.

REFERENCES

1. L. W. Nagel, "SPICE2: A computer Program to Simulate Semiconductor Circuits," Memorandum No. ERL-M520, University of California, Berkeley, May 1975.

2. A. Vladimirescu and D. O. Pederson, "Circuit Simulation on Vector Processors," Proceedings, IEEE International Conference of Circuits and Computers, New York, October 1982.

3. J. C. May, "A Device Clustering Algorithm for Vectorized Circuit Simulation," Proceedings, IEEE International Symposium on Circuits and Systems, Newport Beach, Calif., May 1983.

4. S. McGrogan and G. Tarsy, "Vector Enhancement of a Circuit Simulation Program," Proceedings, Symposium on CYBER 205 Applications, Colorado State University, Fort Collins, Colo., August 1982.

5. S. D. Hamm and S. R. Beckerich, "VAMOS: Circuit Simulation Program for a Vector Computer," Technical Paper, MOSTEK Corporation, Carrollton, Texas, August 1983.

VECTORIZED MONTE CARLO METHODS FOR REACTOR LATTICE ANALYSIS

FORREST B. BROWN

KNOLLS ATOMIC POWER LABORATORY

SCHENECTADY, NEW YORK

VECTORIZED MONTE CARLO METHODS FOR REACTOR LATTICE ANALYSIS

F. B. Brown
Knolls Atomic Power Laboratory
Schenectady, NY

This report details some of the new computational methods and equivalent mathematical representations of physics models used in the MCV code, a vectorized continuous-energy Monte Carlo code for use on the CYBER-205 computer. While the principal application of MCV is the neutronics analysis of repeating reactor lattices, the new methods used in MCV should be generally useful for vectorizing Monte Carlo for other applications. For background, a brief overview of the vector processing features of the CYBER-205 is included, followed by a discussion of the fundamentals of Monte Carlo vectorization. The physics models used in the MCV vectorized Monte Carlo code are then summarized. The new methods used in scattering analysis are presented along with details of several key, highly specialized computational routines. Finally, speedups relative to CDC-7600 scalar Monte Carlo are discussed.

Introduction

Monte Carlo calculations fill a special and important need in reactor physics analysis -- they represent "truth" against which approximate calculational methods may be calibrated. The Monte Carlo method permits the exact modeling of problem geometry, a highly accurate mathematical model for neutron interactions with matter, and a cross section representation that is as accurate as theory and measurement permit. The precision of Monte Carlo results is primarily limited by the computing time required to reduce statistical uncertainties.

Conventional (scalar) Monte Carlo codes simulate the complete history of a single neutron by repeated tracking through problem geometry and by random sampling from probability distributions that represent the collision physics. The accumulation of data for 1,000,000 neutron histories will typically require three to seven hours of CDC-7600 CPU time. On newer computers such as the CYBER-205, scalar Monte Carlo codes may run one and one-half to two times faster (with some tailoring of the coding) because of the reduced cycle time and improved architecture of the scalar processors. Much larger gains are possible when the vector processing hardware of the CYBER-205 is utilized.

The random nature of the Monte Carlo method seems to be at odds with the demands of vector processing, where identical operations must be performed on streams of contiguous data (vectors). Early known efforts to vectorize Monte Carlo calculations for other vector computers were either unsuccessful or, at best, achieved speedups on the order of seven to ten times for highly simplified problems. Recent results for Monte Carlo in multigroup shielding applications and in continuous-energy reactor lattice analysis have demonstrated that Monte Carlo can be successfully vectorized for the CYBER-205 computer. Speedups of twenty to fifty times faster than CDC-7600 scalar calculations have been achieved without sacrificing the accuracy of standard Monte Carlo methods. Speedups of this magnitude permit the analysis of 1,000,000 neutron histories in only five to ten minutes of CPU time and thus make the Monte Carlo method more accessible to reactor analysts.

General Considerations for Vectorized Monte Carlo

Conventional scalar Monte Carlo codes may be characterized as a collection of random decision points separated by short and simple arithmetic. Individual neutron histories are simulated, one at a time. The basic idea of vectorized Monte Carlo is to follow many neutrons simultaneously through their random walks, using vector instructions to speed up the computation rates. The many conditional branches (IF...GOTO), few DO-loops, and largely random data retrieval embodied in conventional Monte Carlo codes preclude vectorization through the use of automatic vectorizing software or by a syntactic vectorization of coding. Instead, experience has shown that a comprehensive, highly integrated approach is required. The major elements of such an approach are as follows:

1. The entire cross section and geometry database must be restructured to provide a unified data layout.

2. The entire Monte Carlo code must be restructured (rewritten).

3. Deliberate and careful code development is essential.

Clever programming and machine "tricks" alone will not ensure successful vectorization of a Monte Carlo code. The key to successful vectorization of Monte Carlo is that a well-defined structure must be imposed on both the database and Monte Carlo algorithm before coding is attempted. This structure may arise simply from the reorganization of existing data/algorithms or may entail the development of special mathematics or physics. Careful and systematic development helps to preserve the structure as the vectorized code becomes more complex.

Vectorization Techniques

The principal obstacle to vectorizing a conventional scalar Monte Carlo code is the large number of IF-statements contained in the coding. Examination of sections of coding shows that, typically, one-third of all essential FORTRAN statements may be IF-tests. Careful consideration of the Monte Carlo program logic and underlying physics permits categorizing these IF-statements and associating them with three general algorithmic features of Monte Carlo codes -- implicit loops, conditional coding, and optional coding. Implicit loops are vectorized using shuffling, and conditional coding is vectorized using selective operations. This approach to vectorizing Monte Carlo is effective on the CYBER-205 and other vector computers having hardware capabilities for vectorized data handling. In successful attempts to vectorize Monte Carlo methods, 40 to 60% of all vector instructions used in actual coding were vector data handling instructions (gather, compress, bit-controlled operations, etc.).

The data-handling operations associated with shuffling and selective operations in the vectorized code constitute extra work that is not necessary in a scalar code. This extra work offsets some of the gain in speed achieved from vectorization. For vectorization to be successful, overhead from shuffling and selective operations should comprise only a small fraction of total computing time. It is thus essential that all data handling operations be performed with vector instructions. Vector computers that must rely on scalar

data handling operations are severely limited in vectorized Monte Carlo performance.

Conclusions

Continuous-energy Monte Carlo methods have been vectorized for the CYBER-205 and the speedups are large. Due to the drastic restructuring of the Monte Carlo coding and data base, the MCV code has been limited to the treatment of repeating reactor lattice geometry. This restriction has been deliberate, however, to permit an orderly and careful program of development. There are no *a priori* limitations on the methods used in vectorization that would preclude extension to more general applications. Profound changes in the methods used for reactor physics analysis are anticipated now that 1,000,000 neutron histories may be run in only five to ten minutes with the CYBER-205 vectorized Monte Carlo vs. the three to seven hours that are typical for CDC-7600 scalar Monte Carlo.

References:

F. B. Brown, "Vectorized Monte Carlo Methods for Reactor Lattice Analysis," KAPL-4163, Knolls Atomic Power Laboratory (1982).

F. B. Brown, "Development of Vectorized Monte Carlo Methods for Reactor Lattice Analysis," Trans. Am. Nucl. Soc., 43, p.377 (1982).

VIBRATIONAL RELAXATION OF DIATOMIC MOLECULES IN SOLIDS AT LOW TEMPERATURES

LAWRENCE L. HALCOMB
AND
DENNIS J. DIESTLER

DEPARTMENT OF CHEMISTRY
PURDUE UNIVERSITY

WEST LAFAYETTE, INDIANA

VIBRATIONAL RELAXATION OF DIATOMIC MOLECULES IN SOLIDS AT LOW TEMPERATURES

Lawrence L. Halcomb* and Dennis J. Diestler

Department of Chemistry

Purdue University

West Lafayette, Indiana 47907

Prepared for delivery at the joint NASA/Goddard-CDC Symposium on CYBER 205 Applications, Lanham, Maryland, October 11-12, 1983.

Abstract

A miscroscopic dynamical treatment of chemical systems comprising both light particles that require a quantal description and heavy ones that may be described adequately by classical mechanics has recently been presented [J. Chem. Phys. 78, 2240 (1983)]. The application of this ''hemiquantal'' method to the specific problem of the vibrational relaxation of a diatomic molecule embedded in a one-dimensional lattice is presented. The vectorization of a CYBER 205 algorithm which integrates the 10^3-10^4 simultaneous ''hemiquantal'' differential equations is examined with comments on optimization. Results of the simulations are briefly discussed.

*David Ross Fellow

I. Introduction

A microscopic dynamical description of a chemical system composed of both light particles that require a quantal description and heavy ones that may be described adequately by classical mechanics has been proposed recently [J. Chem. Phys. 78, 2240 (1983)]. The description consists of a self-consistent set of ''hemiquantal'' equations (HQE) arrived at by taking a partial classical limit of Heisenberg's equations of motion for the system. In form, the HQE appear to consist of Heisenberg's equations for the light particles coupled to Hamilton's equations for the heavy particles. The coupling is self-consistent in that there is an instantaneous feedback between the light and heavy subsystems, with total energy and probability of presence of the quantal subsystem being conserved.

This paper will focus on the numerical solution of the HQE on the CYBER 205 for the special case of a diatomic molecule embedded in a cold, one-dimensional lattice. In Section II, we detail the model and specific form of the HQE, while the CYBER 205 algorithm and steps taken to optimize performance are included in Section III. Results of the simulations and some discussion of their physical significance are presented in Section IV.

II. Model and Equations of Motion

Figure 1 depicts the physical situation, i.e. a single diatomic molecule BC occupying a substitutional site in an otherwise pure one-dimensional lattice of atoms A; the end atoms of the lattice are assumed free. So that the normal modes of the lattice are known analytically, the mass of BC is taken to be equal to that of A. The heavy, classically behaving degrees of freedom are considered to be the displacements (u_i) of the lattice atoms, including the

center of mass of BC, from their equilibrium positions. The internal vibration (q) of BC is treated quantally and, for simplicity, as a harmonic, two-state system. We assume that only nearest-neighbor atoms interact with one another: A-A interactions are harmonic; A-B and A-C interactions are approximated by Morse potentials.

Under these conditions, the HQE take the form

$$\dot{c}_i(t) = -i\hbar^{-1}[\varepsilon_i c_i(t) + \sum_j V_{ij}(\{u_k(t)\})c_j(t)]$$

$$\dot{u}_i(t) = p_i(t)/m_A \qquad (1)$$

$$\dot{p}_i(t) = -\frac{\partial}{\partial u_i} U(\{u_j(t)\}) + \sum_{jk} c_j^*(t)c_k(t)F_{ijk}(\{u_m(t)\}) .$$

Here c_i is the occupation probability amplitude for quantal state i; p_i is the momentum conjugate to u_i; U is the harmonic part of the potential, i.e.

$$U = \frac{k}{2}\{\sum_{i=1}^{n-2}(u_{i+1} - u_i)^2 + \sum_{i=n+1}^{N-1}(u_{i+1} - u_i)^2\}, \qquad (2)$$

where N is the number of lattice atoms. F is the quantal force defined by

$$F_{ijk} = \partial V_{ij}/\partial u_k \qquad (3)$$

where

$$V_{ij}(\{u_k\}) = \langle i|V_{AB} + V_{AC}|j\rangle \ , \tag{4}$$

and the Morse potential V_{AB} is explicitly

$$V_{AB} = D_{AB}\{\exp[-a_{AB}(u_n - u_{n-1} + L - \gamma_B q)] - 1\}^2 \tag{5}$$

with a similar expression for V_{AC}.

Since the c_i are complex, the HQE consist of 2N+4 coupled first-order ordinary differential equations. Given initial conditions appropriate to the physical situation, we can integrate these numerically by standard techniques. Our principal problem now is to develop and optimize an algorithm appropriate to the CYBER 205.

III. CYBER 205 Algorithm

The HQE [Eqs. (1)] can be cast in terms of the vector differential equation $\dot{\mathbf{X}} = f(\mathbf{X}(t))$, defined by

$$\begin{aligned} \dot{x}_1(t) &= f_1(x_1, \ldots, x_n), \ x_1(0) = x_1^o \ , \\ &\vdots \qquad\qquad \vdots \\ \dot{x}_n(t) &= f_n(x_1, \ldots, x_n), \ x_n(0) = x_n^o \ . \end{aligned} \tag{6}$$

The vector $\mathbf{X}$ can be written as

$\mathbf{X} = [\mathbf{C}, \mathbf{U}, \mathbf{P}]$ where, for example,

$$\mathbf{C} = [C1, C2, C3, C4] \ . \tag{7}$$

From experience, we have found the HQE extremely well-behaved. Therefore, they can be handled with a relatively simple differential equation solver. We employ the familiar fourth-order Runge-Kutta algorithm (RK4) which, for our case, is summarized by the following equations:

$$\begin{aligned} \mathbf{K}_1 &= T\, f(\mathbf{X}) \\ \mathbf{K}_2 &= T\, f(\mathbf{X} + \mathbf{K}_1/2) \\ \mathbf{K}_3 &= T\, f(\mathbf{X} + \mathbf{K}_2/2) \\ \mathbf{K}_4 &= T\, f(\mathbf{X} + \mathbf{K}_3) \end{aligned} \tag{8}$$

$$\mathbf{X}[(n+1)T] = \mathbf{X}(nT) + (\mathbf{K}_1+\mathbf{K}_4)/6 + (\mathbf{K}_2+\mathbf{K}_3)/3$$

where T is an appropriately chosen time step. Our choice of RK4 is guided by several considerations; it is quite stable, self-starting and easily coded for the CYBER 205. In addition, we need no direct method of estimating truncation error since we can calculate total energy and probability of the system as a check. Eventually, the RK4 algorithm will be used to calculate input values for a more sophisticated predictor-corrector routine.

Since our simulations require widely varying amounts of memory, we would like to assign storage at execution time. Clearly, the vector pipelines are used more efficiently if the entire derivative vector is manipulated at once. If we are to deal almost exclusively on the dynamic stack, we need a method of parsing the vector **X** into subvectors **C,U,P** which can then be handled independently. This ''breaking up'' is accomplished by building descriptors using SHIFT and OR operations on an integer equivalenced to a descriptor which points to an area in dynamic space. The subroutine BREAKUP is presented in the Appendix. This routine allows the RK4 mainline to allocate storage dynamically while permitting the derivative routine to access each subvector individually.

We now concentrate on the vector function subprogram that calculates the derivative f(X). In our model, the four probability amplitudes must be accessed individually each time the function is called. Rather than waste a vector instruction to store the subvector C in a temporary array, it is faster and more convenient to use the following sequence of hardware calls to load them directly into registers:

```
ASSIGN TEMP, C
CALL Q8LOD (TEMP,, C1)
CALL Q8IX(TEMP, 64)
CALL Q8LOD(TEMP,, C2), etc.
```

The constants needed to calculate the potential and force functions are computed in advance and passed via labeled common. By reviewing an assembly listing of the program, one can minimize the number of loads necessary to access these constants. The evaluation of $\dot{U}$ is easily done by a vector multiplication with a stored reciprocal mass.

$\dot{P}$ can be conveniently calculated by evaluating the derivative of a fully harmonic potential U'. Thus we have

$$-\frac{\partial}{\partial u_i} U'(\{u_j\}) = k(-2u_i+u_{i-1}+u_{i+i}) \text{ where}$$

$$u_o = u_1 \ , \ u_{N+1} = u_N, \tag{9}$$

which can be effected by two vector additions and two vector multiplications as follows:

$$-\frac{\partial}{\partial \mathbf{U}}U'(\{u_j\}) = \text{UTEMP}(1;N) = K*(-2.*\text{UTEMP}(1;N) + \text{UTEMP}(0;N) + \text{UTEMP}(2;N))$$

where UTEMP is a temporary array set to the current values of **U**. Finally, $\dot{\mathbf{P}}$ is obtained by replacing the n-1, n, and n+1 elements of UTEMP by the proper values reflecting the Morse potentials at the diatomic. To accomplish this, it is necessary to access the five displacements $\{u_i,\ i = n-2,\ n+2\}$. Alternatively, descriptors could be built to define the necessary vectors on **U** and the values stored in UTEMP. In this case, hardware calls would be required to set the first and last elements of UTEMP, to access the five elements of **U** around u_n, and to store values in the three middle positions.

The conservation of total energy and probability gives us two necessary criteria to check the accuracy of the numerical solution. The total energy is given by

$$E = U(\{u_i\}) + \mathbf{P}\cdot\mathbf{P}/(2m_A)$$

$$+ |c_o|^2\varepsilon_o + |c_1|^2\varepsilon_1 \qquad (10)$$

$$+ |c_o|^2 V_{oo} + 2\mathrm{Re}\{c_o^* c_1\}V_{10} + |c_1|^2 V_{11}$$

while total probability is simply

$$P = |c_o|^2 + |c_1|^2, \qquad (11)$$

which must remain unity. These checks were made every 1000 iterations using values calculated in the first pass through the derivative routine. To calculate U' [Eq. (9)], the following code is used:

```
ASSIGN TEMP,.DYN. N-1
TEMP= Q8VDELT(U;TEMP)
EU= (K/2)* Q8SDOT(TEMP,TEMP).
```

In Table I, sample iteration times and estimates of floating point operations per second are given. The timings are for loops without I/O or accuracy checks. The results of several simulations are presented in the next Section.

IV. Results of Simulations

Our simulations all take the diatomic to be in its excited state and the lattice to be at 0K initially. This means that all elements of $\mathbf{X}(0)$ are zero, except the real component of $c_1(0)$, which is unity. The time step size is .01 ω^{-1}, where ω is the transition frequency of the diatomic. The quantity of principal interest here is $|c_1|^2$, the probability of the diatomic being excited. The physical constants for the system, which are chosen roughly to mimic HCl in Ar, are listed in Table 2. The only variable quantities are ω and N. The transition frequency is chosen low in order to observe relaxation on the time-scale of the simulation.

Figure 1 displays plots of $|c_1|^2$ versus time for a sampling of simulations. Frames (a)-(c) demonstrate the effect of increasing the diatomic's transition frequency ω, (cm^{-1}) holding the number of lattice atoms fixed. It appears that the rate of loss of energy from the diatomic increases with increasing frequency up to a point. In fact, frame (c) suggests that the diatomic evolves to a metastable state in which it loses no further energy. To test this hypothesis, we increased the number of lattice atoms to N = 2000. The result, shown in frame (f), bears this notion out. For purposes of comparison, we

include a simulation for a smaller lattice (N = 200). Here we see the effect of a pulse, which bounces back and forth, interfering with the monotonic relaxation of the diatomic.

V. Conclusion

These simulations represent the first application of a new description of the dynamics of chemical processes. Most previous approaches employ long-time asymptotic approximations, in which the coupling between the subsystems is weak and the decay is therefore very slow on the time scale of molecular motions (10^{-14}s). The advancement of ultrafast laser spectroscopy now allows chemists to monitor directly fast relaxation processes (10^{-12}s). In this regime, the coupling is more significant, and accurately solving the equations of motion becomes crucial. The HQE can be used for this purpose. However, any practical implementation will require a vector processor, such as the CYBER 205. Our calculations would be essentially impossible on Purdue University's 6500/6500/6600 system, for example. The calculations would take 50–100 times longer, even if the storage for the vectors were available.

The main feature of our CYBER 205 algorithm is a mainline that assigns storage at execution time. The vector function subprogram that evaluates the derivative can access the subvectors individually while the mainline processes the entire vector. This is accomplished by building the appropriate descriptors using the BREAKUP subroutine (see Appendix).

Some preliminary results were presented in Section IV. Future research will deal with the actual mechanism of energy exchange between the two subsystems. Also planned are some N-state models with applications in surface chemistry.

Acknowledgements

We would like to thank Purdue University for providing computer time on the CYBER 205 and the Purdue Research Foundation for financial support of this work. Also, we would like to thank Daniel Severance and the Purdue University Computing Center User Services Group, especially David Seaman, for helpful discussions.

Table I. Increase of calculation speed with increase of number of equations

Equations	Iteration Time	Mega FLOPS
24	.157 ms	6.1
204	.204 ms	22.8
804	.256 ms	37.9
2004	.671 ms	69.3
4004	1.19 ms	77.7
10004	2.75 ms	83.8
20003	5.37 ms	85.6

Table II. Parameters of model system

D_{AB}	=	9.25×10^{-15} ergs	D_{AC}	=	1.24×10^{-14} ergs
a_{AB}	=	1.83×10^{8} cm^{-1}	a_{AC}	=	1.66×10^{8} cm^{-1}
k	=	814 ergs/cm^2	m_A	=	6.64×10^{-23} g
m_B	=	1.67×10^{-24} g	m_C	=	5.88×10^{-23} g

Appendix

```
      SUBROUTINE BREAKUP(X,NSUB,LENSUB,DESSUB,NDIM)
      IMPLICIT INTEGER(A-Z)
C
C     BREAKUP- TAKES A DESCRIPTOR (X) AND MANUFACTURES OTHER
C        DESCRIPTORS [DESSUB(N)] THAT POINT TO SUBVECTORS OF
C        LENGTHS LENSUB(N) WHICH COMPRISE THE VECTOR POINTED
C        TO BY X
C
C     ARGUMENTS:
C           X- DESCRIPTOR TO BE 'BROKEN UP'
C           NSUB- NUMBER OF SUBVECTORS
C           LENSUB- ARRAY CONTAINING THE SUBVECTOR LENGTHS
C           DESSUB- ARRAY CONTAINING THE RESULTING DESCRIPTORS
C           NDIM- DIMENSION OF LENSUB AND DESSUB
C
      DESCRIPTOR D,X,DESSUB(NDIM)
      DIMENSION LENSUB(NDIM)
      EQUIVALENCE (D,DTEMP)

         ASSIGN D,X
         ADD= SHIFT( SHIFT( DTEMP,16 ), -16)
         DO 100 N=1,NSUB
            LENGTH= SHIFT( LENSUB(N),48 )
            DTEMP= OR( ADD,LENGTH )
            ASSIGN DESSUB(N),D
            ADD= ADD + 64*LENSUB(N)
  100    CONTINUE

      RETURN
      END
```

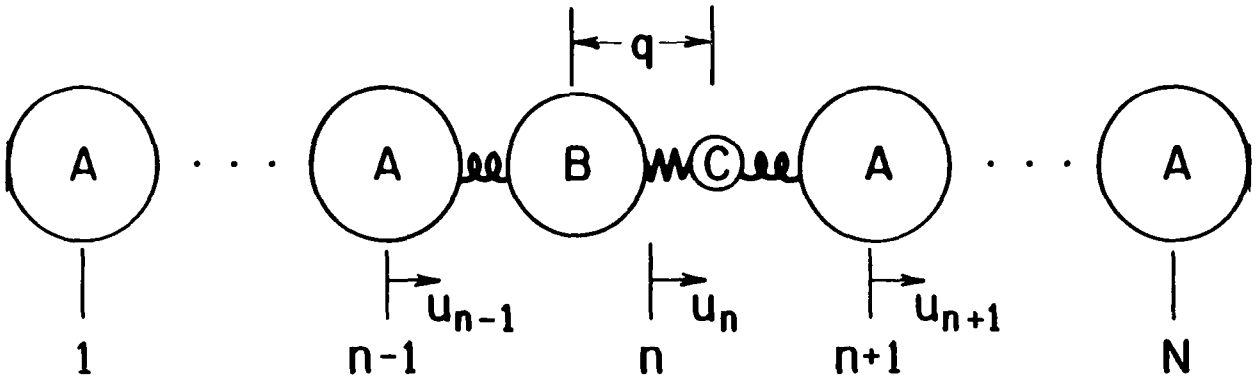


Figure 1. One-dimensional model of a substitutional diatomic molecule BC in an otherwise pure lattice of atoms A.

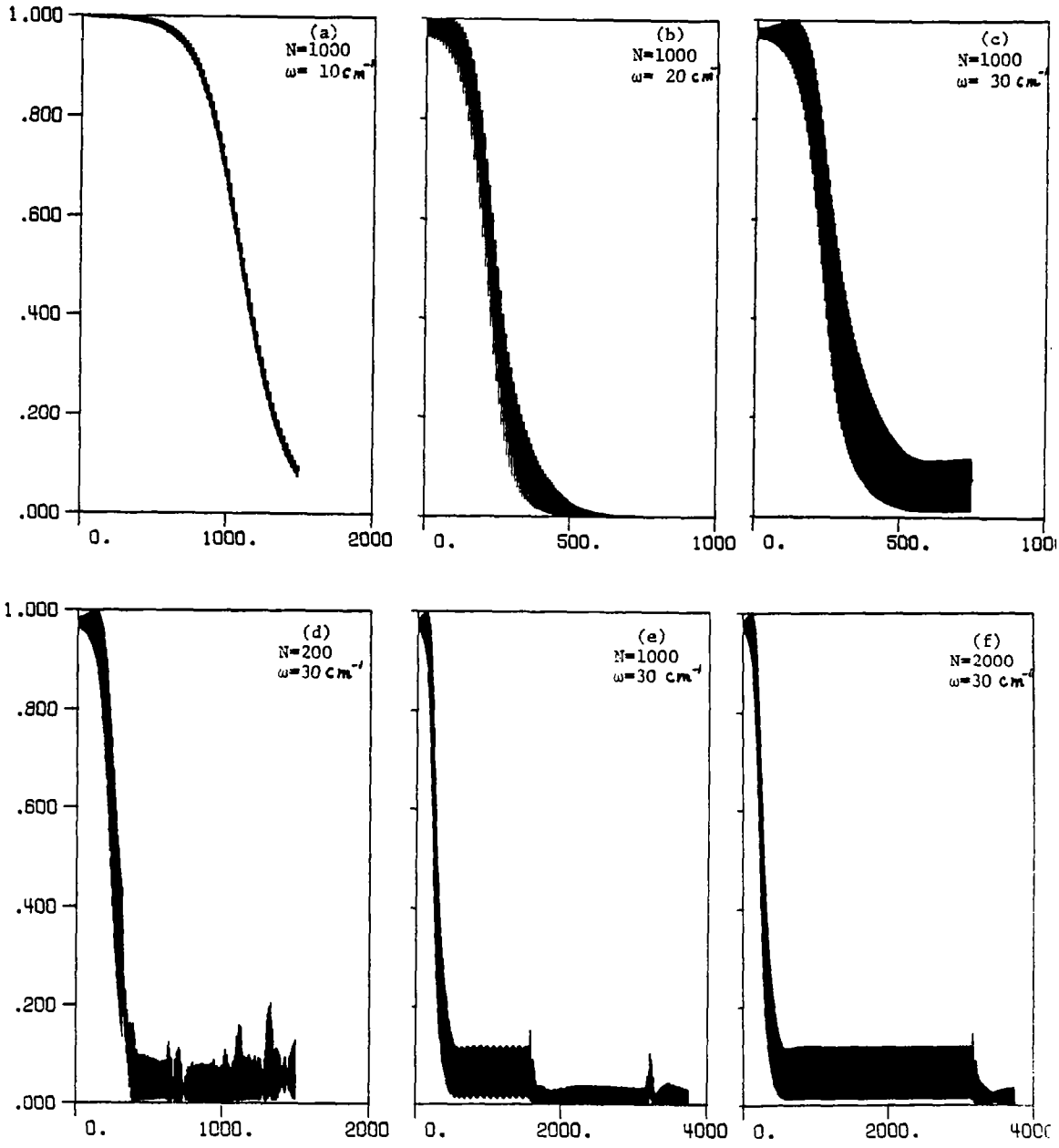


Figure 2. Plots of probability of finding diatomic in the excited state versus time for a selection of simulations of the system defined by parameters of Table 2. Time is in units of 0.18 picoseconds.

CHEMICAL APPLICATION OF DIFFUSION QUANTUM MONTE CARLO

PETER J. REYNOLDS
AND
WILLIAM A. LESTER, JR.

MATERIALS AND MOLECULAR RESEARCH DIVISION
LAWRENCE BERKELEY LABORATORY
UNIVERSITY OF CALIFORNIA

BERKELEY, CALIFORNIA

CHEMICAL APPLICATION OF DIFFUSION QUANTUM MONTE CARLO*

Peter J. Reynolds and William A. Lester, Jr.+
Materials and Molecular Research Division
Lawrence Berkeley Laboratory
University of California
Berkeley, California 94720

The diffusion quantum Monte Carlo (QMC) method gives a stochastic solution to the Schrodinger equation. This approach has recently been receiving increasing attention in chemical applications as a result of its high accuracy. However, reducing statistical uncertainty remains a priority because chemical effects are often obtained as small differences of large numbers. We give as an example the singlet-triplet splitting of the energy of the methylene molecule CH_2.

We have implemented the QMC algorithm on the Cyber 205, first as a direct transcription of the algorithm running on our VAX 11/780, and second by explicitly writing vector code for all loops longer than a crossover length C*. We discuss the speed of the codes relative to one another as a function of C*, and relative to the VAX. Since CH_2 has only eight electrons, most of the loops in this application are fairly short. The longest inner loops run over the set of atomic basis functions. We discuss the CPU time dependence obtained versus the number of basis functions, and compare this with that obtained from traditional quantum chemistry codes and that obtained from traditional computer architectures. Finally, we discuss some preliminary work on restructuring the algorithm to compute the separate Monte Carlo realizations in parallel--potentially allowing vectors of unlimited length.

*This work was supported in part by the Director, Office of Energy Research, Office of Basic Energy Sciences, Chemical Sciences Division of the U. S. Department of Energy under Contract No. DE-AC03-76SF00098, Director's Program Development Fund, Lawrence Berkeley Laboratory, and the Control Data Corporation.

+Also Department of Chemistry, University of California, Berkeley, California.

1. BACKGROUND

In recent years Monte Carlo methods have been increasingly applied to quantum-mechanical problems. Quantum Monte Carlo (QMC) methods fall into two major categories. Variational QMC[1] is a method of evaluating expectation values of physical quantities with a given (generally optimized) trial wave function Ψ_T. The procedure in effect amounts to evaluating a ratio of two integrals, although the actual Monte Carlo procedure is generally more sophisticated. The second major category of QMC is the "exact" type.[2] In these latter approaches the Schrödinger equation is actually "solved". It is not necessary to already have a highly accurate wave function in order to compute the expectation values. Properties of interest are in effect "measured" as the system evolves under the Schrödinger equation. When a stationary state is obtained, averages of the measured quantities give the desired expectation values.

Only recently have chemical calculations by exact QMC methods been carried out.[3,4] We will discuss here one such QMC method -- the fixed-node, diffusion QMC -- which we have been using in calculating molecular energies. In Section 2 we present the basic theory. Section 3 describes the algorithm. The implementation of this algorithm on the Cyber 205, its optimization, and results, are discussed in Section 4.

2. BASIC THEORY[4]

The Schrödinger equation may be rewritten in imaginary time, and with a constant shift in the zero of energy in the following form:

$$\frac{\partial \Psi(\underset{\sim}{R},t)}{\partial t} = [D\nabla^2 - V(\underset{\sim}{R}) + E_T]\ \Psi(\underset{\sim}{R},t) \quad . \tag{1}$$

Here $D \equiv \hbar^2/2m_e$, $\underset{\sim}{R}$ is the three-N dimensional coordinate vector of the N electrons, and $V(\underset{\sim}{R})$ is the potential energy (the Coulomb potential for a molecular system). Equation (1) is simply a diffusion equation combined with a first-order rate process, and thus may be readily simulated. The function $\Psi(\underset{\sim}{R},t)$ plays the role of the density of diffusing particles. These particles undergo branching (exponential birth or death processes) according to the rate term $[E_T - V(\underset{\sim}{R})]\ \Psi(\underset{\sim}{R})$. Thus, the number of diffusers increases or decreases at a given point in proportion to the density of diffusers already there.

The steady-state solution to Eq. (1) is the ground-state eigenfunction of the Schrödinger equation. Furthermore, the value of E_T at which the population of diffusers is asymptotically constant gives the energy eigenvalue E_0. The lowest eigenstate, however, is that of a Bose system. In order to treat a Fermi system, such as a molecule, we need to impose anti-symmetry on $\Psi(\underset{\sim}{R})$. A method which does this, and at the same time allows us to sample more efficiently (to reduce our statistical error), is importance sampling with an

anti-symmetrized importance function Ψ_I. The zeros (nodes) of Ψ_I become absorbing boundaries for the diffusion process, maintaining the anti-symmetry. A simple form for Ψ_I which gives the necessary anti-symmetry is a Slater determinant of molecular orbitals multiplied by a symmetric function of the coordinates.

To implement importance sampling, one simply multiplies Eq. (1) by Ψ_I and rewrites it in terms of a new probability density $f(\underset{\sim}{R},t)$ given by

$$f(\underset{\sim}{R},t) \equiv \Psi_I(\underset{\sim}{R})\,\Psi(\underset{\sim}{R},t). \tag{2}$$

The resultant equation for f can be written as

$$\frac{\partial f}{\partial t} = D\nabla^2 f + [E_T - E_L(\underset{\sim}{R})]f - D\nabla\cdot[fF_Q(\underset{\sim}{R})]\ . \tag{3}$$

The local energy $E_L(\underset{\sim}{R})$ and the "quantum force" $F_Q(\underset{\sim}{R})$ are simple functions of $\Psi_I(\underset{\sim}{R})$. Eq. (3), like Eq. (1), is a generalized diffusion equation, now with the addition of a drift term, due to the effect of F_Q. It is Eq. (3) that we solve stochastically. Using a Green's function approach, our diffusers are made to follow a "random walk" (Markov chain) in such a way that their asymptotic distribution is given by the steady-state solution, $f_\infty(\underset{\sim}{R})$, of Eq. (3). Properties of interest (such as the energy) are measured during the "walks", and are thus averages over the distribution $f_\infty(\underset{\sim}{R})$.

3. ALGORITHM

We present here an outline of the algoritnm for performing diffusion QMC. For more detail see Ref. 4. This algorithm is not structured specifically for the architecture of the Cyber 205. We will return to this point in the next section.

(0) Initialization. First generate an ensemble of N_c configurations of the N-electron system. Typically $N_c \approx 100\text{-}500$. These coordinates may be chosen randomly, or more efficiently from the distribution $|\Psi_I(\underset{\sim}{R})|^2$. This initial distribution is $f(\underset{\sim}{R}, t=0)$.

(1) Loop over blocks. In each block:

(2) Repeatedly loop over the ensemble until the time in each configuration has reached the chosen target time. For each member of the ensemble compute the inverse of the Slater matrix. Then:

(3) Loop over the electrons. Compute F_Q for the current electron. Move to

$$r' = r + D\tau F_Q + \chi \qquad (4)$$

where τ is the discrete time-step size, and χ is a 3-dimensional Gaussian random variable with a mean of zero

and a variance of $2D\tau$. This corresponds to the diffusive motion. If the electron crosses a node, eliminate the configuration from the ensemble and continue loop (2) over the ensemble. Otherwise update the Slater matrix and its inverse, and continue loop (3).

After all electrons in the current configuration have been moved, advance the time associated with this new configuration R' by τ. Calculate $E_L(\underset{\sim}{R}')$. Also calculate the branching factor, or multiplicity.

$$M = \exp\left(-\tau\{[E_L(\underset{\sim}{R}) + E_L(\underset{\sim}{R}')]/2 - E_T\}\right). \qquad (5)$$

Return M copies of this configuration to the ensemble. This branching, or birth and death process, corresponds to the rate term in Eq. (3). Weight all averages by M. Continue loop (2). After all members of the ensemble have reached the target time, the current block is finished. Use $\langle E_L \rangle$ to update E_T. Store $\langle E_L \rangle$ and the other averages. "Renormalize" the ensemble back to its original size N_c. (This is necessary because the population grows or shrinks exponentially. Although we have endeavored to make the exponent close to zero [cf Eq. (5)], asymptotically at large time the population will either vanish or overflow the allocated storage.) Reset all averages to zero. Continue loop (1) for the desired number of blocks.

(4) Average over blocks.

4. CYBER 205 IMPLEMENTATION.

The problem we chose to study is the singlet-triplet energy splitting of the methylene molecule, CH_2. CH_2 is fairly typical of the molecules we have been studying by QMC, in terms of the number of electrons and the number of nuclei. As a result, most of the inner loops in this application are quite short. The longest inner loop runs over the set of atomic basis functions. With this in mind, we present our results on the relative performance of the Cyber 205 and the VAX 11/780. To compare with the CDC 7600, we note that our code runs almost exactly ten times faster on the 7600 than on the VAX.

We have implemented the QMC algorithm on the Cyber 205, initially by simply transcribing our working program from the VAX to the Cyber. The major impediment at this stage was the lack of unformatted I/O on the Cyber and, even worse, its inability to handle logical records longer than 137 bytes. After rewriting these portions of the code, the program finally ran.

With automatic vectorization both on and off, the Cyber ran approximately 16 times the speed of the VAX. Apparently, any speed-up from vectorization of the longer loops was lost to the start-up time for vectorizing the short loops. It seemed clear that explicit vectorization was required. Thus, as our next step, all long inner loops of constant length were written explicitly in vector

syntax, while short constant-length loops were left as DO loops. Most loops in the code, however, are of variable length. These were all recoded in the form:

```
IF (length .GT. C*)      THEN

        [Vector code]

ELSE

        [Scalar code]

END IF.
```

We present in Figure 1 our performance results as a function of the crossover length C*. At values of C* greater than 26 the scalar section of code is always being executed, and thus the curve flattens out. For C* less tnan approximately 16, it appears that vector start-up time hinders performance. The optimum crossover point appears to be around 16. The lowest of the three curves corresponds to the implementation described above. Subroutine calls are quite costly on the Cyber 205. Thus in the middle curve we show the result of removing two short subroutines (both written in IF-THEN-ELSE form) and substituting vector code directly into the calling

programs. The speed-up is fairly dramatic, providing a peak speed of close to 20 times the VAX (up from 17).

Interestingly, although the compliler recognizes that A**2 should be replaced by A*A, inside of vector code A**2 calls the float-to-an-integer-power routine. Needless to say, this is costly. Essentially, changing one line of vector code from A**2 to A*A led to the improvement shown in the top curve. Clearly the improvement is most pronounced for small C*, where this line of code is being executed more frequently.

As mentioned earlier, the longest inner loop is over the number of atomic basis set functions, n. Traditional quantum chemistry codes scale as n^4 or n^5. Thus increasing the size of the basis set can be very costly. In our QMC approach, the algorithmic dependence on n is linear. In Fig. 2 we plot the relative run times as a function of basis set size on both the VAX (upper curve) and the Cyber 205 (lower curve). Both curves are indeed fairly linear in n. However, the slope for the Cyber is almost flat. This smaller slope is due to an increase in the vector length rather than an increase in the number of machine instructions being executed. The result is a speed enhancement of 30 over the VAX (up from 20) by n=50.

Although a factor of 30 over the VAX (or equivalently a factor of 3 over the 7600) is certainly good, it is nowhere near our hoped

for performance. This can be explained by the fact that even loops of length 50 are relatively short on the Cyber 205. Possibly more important, however, is that the relatively long inner loops constitute only a fraction of the code being executed. Thus, truly high speed for this kind of application requires on architectural rewrite of the code.

Looking over the algorithm (cf Sect. 3) it is clear that the entire structure is highly parallel. This is a fairly general characteristic of Monte Carlo codes. Thus, on a parallel processor the loop (1) over blocks can be eliminated, and each block can be computed independently on a separate processor. There is no communication required between processors until the very end, when [step (4)] the average over blocks is computed.

For a truly efficient Cyber 205 algorithm, however, loop (1) is too short to vectorize, generally ranging between 10 and 100. Loop (2) is much more desirable to vectorize, with $N_c \approx 100\text{-}500$. To do so, this loop must be made innermost in the new algorithm. In other words, the entire ensemble must be treated in parallel. Furthermore, the vector length is dynamic, since at each time-step the birth and death process modifies the ensemble size. We are currently developing this fully vector code for future implementation. This code appears to have great potential for fully exploiting the vector capabilities of the 205.

Finally in Table 1, we present our results on the singlet-triplet energy splitting of methylene, and compare these results with theory and experiment.

5. ACKNOWLEDGMENTS

We would like to thank the Control Data Corporation for a grant of computer time on the Cyber 205 at Colorado State University. We also thank the Institute for Computational Studies at Colorado State University and the Center for Advanced Vector Technology for their orientation program and assistance. Finally, we thank Steve McGrogan for many helpful comments on the code, and Robert Barnett for preparing the figures.

REFERENCES

1. W. L. McMillan, Phys. Rev. 138, A442 (1965); D. M. Ceperley, G. V. Chester, and M. H. Kalos, Phys. Rev. B 16, 3081 (1977).
2. See for example, M. H. Kalos, Phys. Rev. 128, 1791 (1962), and M. H. Kalos, D. Levesque, and L. Verlet, Phys. Rev. A 9, 2178 (1974).
3. J. B. Anderson, J. Chem. Phys. 63 1499 (1975); 65, 4121 (1976); 73, 3897 (1980); 74, 6307 (1981).
4. P. J. Reynolds, D. M. Ceperley, B. J. Alder, and W. A. Lester, Jr., J. Chem. Phys. 77, 5593 (1982).

TABLE 1.

The ground-state ($^3B_1$) and first-excited state ($^1A_1$) energies of methylene.

Method	$^3B_1$-energy (hartrees)	$^1A_1$-energy (hartrees)
Hartree-Fock	-38.9348	-38.8944
CI-SD	-39.1071	-39.0956
CI-SDQ (est.)	-39.122	-39.105
QMC	-39.128±0.004	-39.108±0.004
Experimental	-39.148	---

$^1A_1 - {}^3B_1$ energy (kcal/mole)	
CI	9.9-11.3
Expt	8.5-19.6
QMC	12.3±3.4

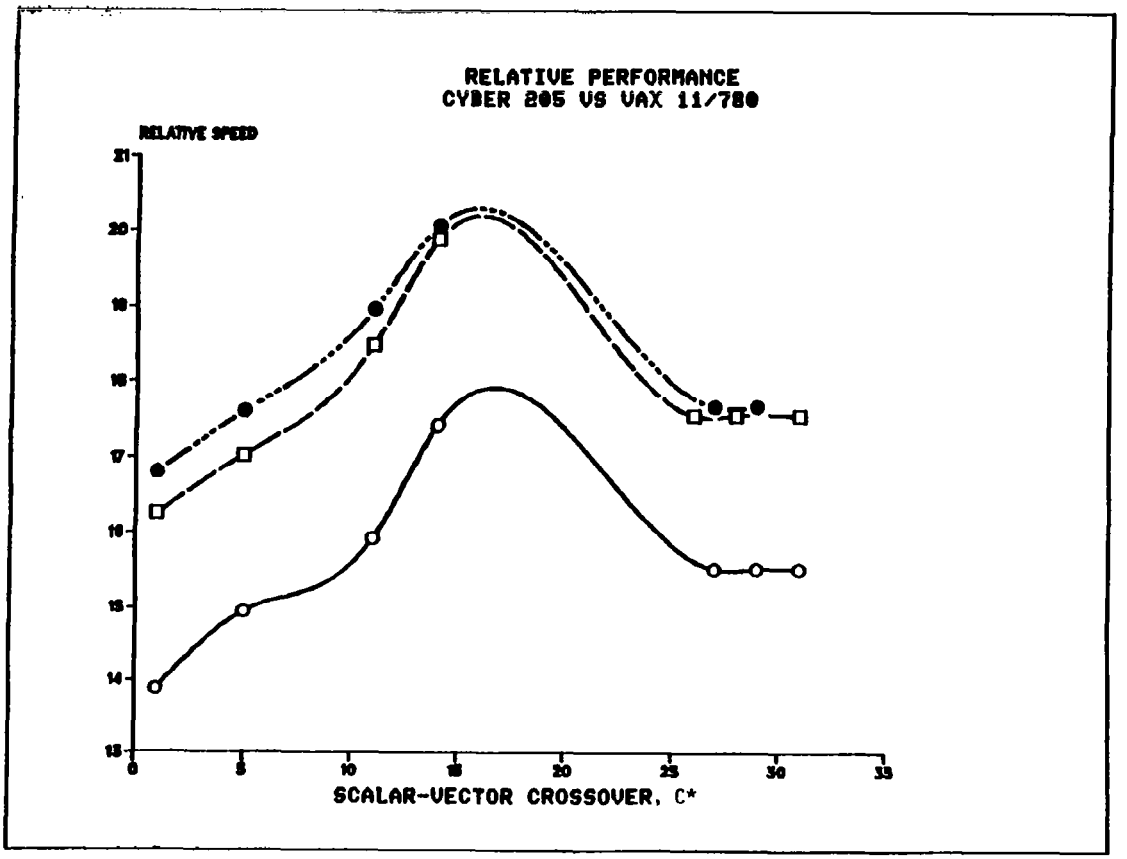


Figure 1. Relative speeds of the Cyber 205 and the VAX 11/780 for quantum Monte Carlo calculations of the ground-state energy of CH_2. The crossover point C* is the vector length below which variable-length loops are run in scaler mode. Thus, for large C* these loops are all run in scaler mode,whereas for very small C*, vector start-up time hinders performance. The three curves correspond to different degrees of hand-optimization of the code. See text for details. Note that the curves interpolating the data points are simply polynomial fits to the data. The actual curve for a particular molecule is a set of steps at the values of the various loop lengths that occur in the problem. The fits can be considered an "average" behavior for this type of calculation.

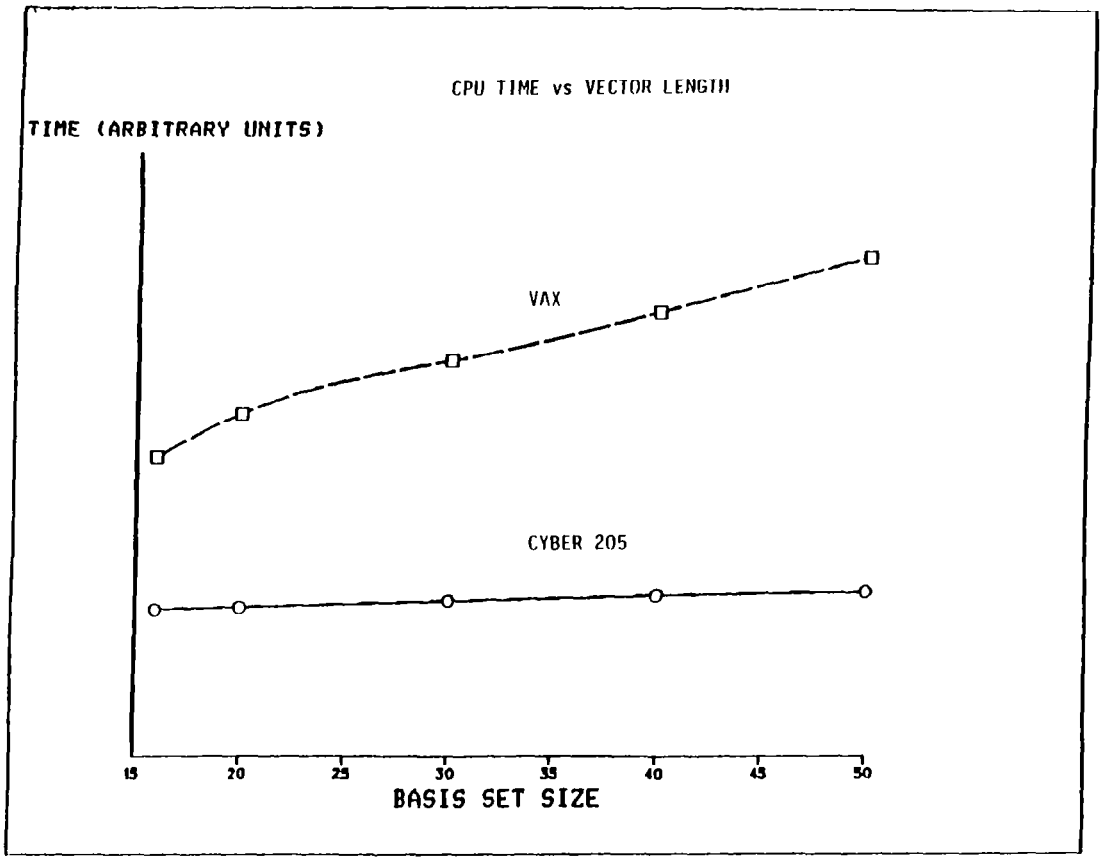


Figure 2. CPU time versus the number of atomic basis set functions, n. Conventional codes scale as n^{λ} with $\lambda \approx 4\text{-}6$ while QMC scales simply as n. Both the VAX and Cyber show this n dependence clearly. However, the slope for the Cyber is almost zero. At n=16 the Cyber is 20 times the speed of the VAX while at n=50 the Cyber is 30 times faster.

A HIGHLY OPTIMIZED VECTORIZED CODE FOR MONTE CARLO SIMULATIONS OF SU(3) LATTICE GAUGE THEORIES

D. BARKAI
CONTROL DATA CORPORATION
INSTITUTE FOR COMPUTATIONAL STUDIES AT CSU
FORT COLLINS, COLORADO

K. J. M. MORIARTY
DEPARTMENT OF MATHEMATICS, STATISTICS
AND COMPUTING SCIENCE
DALHOUSIE UNIVERSITY
HALIFAX, NOVA SCOTIA, CANADA
AND
DEPARTMENT OF MATHEMATICS
ROYAL HOLLOWAY COLLEGE
ENGLEFIELD GREEN, SURREY, U.K.

AND

C. REBBI
DEPARTMENT OF PHYSICS
BROOKHAVEN NATIONAL LABORATORY
UPTON, NEW YORK

A Highly Optimized Vectorized Code for Monte Carlo Simulations of SU(3) Lattice Gauge Theories*

D. Barkai
Control Data Corporation
at the
Institute for Computational Studies at CSU
P.O. Box 1852
Fort Collins, Colorado 80522

K.J.M. Moriarty
Department of Mathematics, Statistics
and Computing Science
Dalhousie University
Halifax, Nova Scotia B3H 4H8, Canada
and
Department of Mathematics
Royal Holloway College
Englefield Green, Surrey TW20 OEX, U.K.

and
C. Rebbi
Department of Physics
Brookhaven National Laboratory
Upton, New York 11973

Abstract

New methods are introduced for improving the performance of the vectorized Monte Carlo SU(3) lattice gauge theory algorithm using the CDC CYBER 205. Structure, algorithm and programming considerations are discussed. The performance achieved for a 16^4 lattice on a 2-pipe system may be phrased in terms of the link update time or overall MFLOPS rates. For 32-bit arithmetic it is 36.3 μsec/link for 8 hits per iteration (40.9 μsec for 10 hits) or 101.5 MFLOPS.

September 1983

*Talk presented at the International Conference "Parallel Computing 83" at the Free University of Berlin, 26-28 September 1983 and at the Joint NASA/Goddard-CDC Symposium on CYBER 205 Applications, held at Lanham, Maryland, 11-12 October 1983.

1. *Introduction*

Many important results for quantum field theories in general and, in particular, for the gauge theory of strong interactions known as Quantum Chromodynamics (QCD) have been obtained by formulating the dynamics on a space-time lattice. The lattice version of a quantized gauge field theory, as proposed by Wilson [1], has the properties of introducing an ultraviolet cut-off independently of any perturbative expansion and of preserving manifest gauge invariance. It permits a variety of investigations by non-perturbative techniques, strong-coupling expansions [2] and Monte Carlo (MC) simulations [3] being the most notable ones. Monte Carlo simulations, indeed, have probably produced the most important results for QCD, being able to probe the structure of the theory in the domain where the transition between the strong-coupling behavior at large distances and the asymptotically-free behavior at small separation takes place.

Numerical methods must be used to explore the vary crucial domain of intermediate couplings, since there are no known analytical techniques for solving or even efficiently approximating gauge theories throughout that region. On the other hand the fact that quantum fluctuations on a finite lattice extending for n sites in four dimensions are given by integrals of a dimensionality $4n^4n_g$ (n_g is the number of independent parameters in group space), which can easily exceed 2,000,000, leaves importance sampling, i.e. Monte Carlo simulations, as the only calculational possibility.

Monte Carlo calculations are of a numerical nature, and quite demanding on computational resources. The simulation of a system with SU(3) gauge group (i.e. the system underlying QCD) on a lattice extending for n sites in each of the four space-time dimensions requires storage of $4n^4$ link variables, i.e. $4n^4$ SU(3) matrices, and the systematic

replacement, or "upgrading", of each of these matrices with new, updated values, for several hundred or several thousand sweeps of the whole lattice. One MC iteration is defined as a sweep of the lattice, i.e., one upgrade per link variable. A computation involving M MC iterations thus implies $4Mn^4$ individual upgrades of SU(3) matrices. The upgrading of each SU(3) matrix requires approximately 4,150 elementary arithmetic operations and 180 table look-ups (if 10 attempts at changing the link variable are made for each upgrade). For a lattice large enough for obtaining physically meaningful results, the amount of computation needed for a Monte Carlo simulation of QCD becomes extremely high.

Because of the aforementioned difficulties, Monte Carlo simulations of QCD have been generally limited to lattices of rather small extent, a lattice of 8^4 sites already representing a large lattice with respect to the scale of most calculations. On the other hand, with the progress in the field, it has become apparent that one must definitely analyze larger systems to develop confidence in the numerical results. This need may be understood on physical grounds. If 2 GeV is considered as a universal energy for the effects of asymptotic freedom to begin manifesting themselves, one would like the lattice spacing to be smaller than $(2\text{GeV})^{-1}$ (and the corresponding ultraviolet cut-off larger than 2GeV) i.e. smaller than 0.1fm. Conversely, if the goal of the computations is to determine hadronic structure, the extent of the lattice should be larger than the typical size of a hadron. Taking this size to be (minimally) 1 fm, it becomes apparent that the parameter n ought to be larger, if possible substantially larger, than 10. With, e.g., n = 16 and M = 1000 the calculation of a MC simulation requires more than 10^{12} operations not a small task even for the largest machines currently available.

The number of the data elements involved, and the amount of computations needed for manipulating this data, makes it worth while to investigate ways for vectorization of the code.

The purpose of this article is to illustrate the vectorization and implementation on the CDC CYBER 205 of a code for Monte Carlo simulations of the SU(3) lattice gauge theory. (For previous implementations of vectorized code see Ref.4.) As will be discussed in more detail in the final section of this paper, the characteristics and performance are such that 1 MC iteration of a 16^4 lattice can be done in 10.72 seconds (corresponding to an upgrade time of 40.9 μsec per SU(3) link variable). Thus, 16^4 and larger lattices can be considered for meaningful simulations of QCD. While we describe in this article the program for the basic Monte Carlo algorithm, we are currently using it, together with other vectorized codes, for a reevaluation on a large lattice, of several quantities of theoretical and phenomenological interest in QCD. The results of these investigations will be presented separately [5]. Here we proceed with a description of the computational algorithm and an outline of its vectorization in Sect. 2, with a more detailed account of the program in Sect. 3 and a summary of performance data in Sect. 4.

2. The Monte Carlo Algorithm

We consider a hypercubical lattice of n_s sites in each of the three spatial directions and n_t sites in the temporal one. The dynamical variables of the SU(3) gauge theory are 3×3 unitary-unimodular complex matrices, which are associated with the $4n_s^3n_t$ links of the lattice. We denote by U_x^μ the matrix associated with the

oriented link coming out of the lattice site of (integer) coordinates $x \equiv (x_1,x_2,x_3,x_4)$ in the direction μ (μ=1,2,3,4). The goal of the Monte Carlo algorithm is to produce a stochastic sequence of configurations of the system $C^{(i)}$, (a configuration being defined as the collection of all U_x^μ), such that the probability $P(C)$ of encountering any configuration C in the sequence approaches, after a reasonable equilibriation time, the distribution

$$P(C) \propto \exp\{-S(C)\} \quad , \tag{2.1}$$

where S is the action of the configuration C in that sequence. S is given by a sum over plaquette variables p, a plaquette being an oriented square of the lattice defined by the origin x and two directions μ and ν :

$$S = \sum_p S_p = \beta \sum_p (1 - \frac{1}{3} \text{Re Tr } U_p) \quad , \tag{2.2}$$

where

$$U_p \equiv U_x^{\mu\nu} = U_x^{\nu +} U_{x+\hat{\nu}}^{\mu +} U_{x+\hat{\mu}}^{\nu} U_x^{\mu} \quad , \tag{2.3}$$

β is the coupling parameter and $\hat{\mu}$, $\hat{\nu}$ stand for unit lattice vectors in the μ and ν directions, respectively. When Eqn. 2.1 is satisfied, quantum mechanical expectation values of observables $\mathcal{O}$, defined rigorously as averages over all possible configurations, namely

$$\langle \mathcal{O} \rangle = Z^{-1} \int \prod_{x,\mu} dU_x^\mu \, \mathcal{O}(U) \exp[-S(U)] \qquad (2.4)$$

with

$$Z = \int \left(\prod_{x,\mu} U_x^\mu \right) \exp[-S(U)] \quad , \qquad (2.5)$$

can be approximated by averages taken over the configurations generated by the Monte Carlo algorithm:

$$\langle \mathcal{O} \rangle \simeq \frac{1}{N} \sum_{i=N_0+1}^{i=N_0+N} \mathcal{O}(C^{(i)}) \quad . \qquad (2.6)$$

N_0 represents the number of initial configurations discarded in order to allow for the stochastic sequence to reach equilibrium.

In our code we implement the MC algorithm following the method of Metropolis et al [6]. Each individual dynamical variable U_x^μ is replaced by a new one $\tilde{U}_x^\mu$ according to the following procedure:

i) a new candidate matrix $U_x^{\mu\,\prime}$ is obtained from U_x^μ by group multiplication:

$$U_x^{\mu\,\prime} = R_k U_x^\mu \quad ,$$

where R_k is an $SU(3)$ matrix randomly selected from a prepared set $\{R_1, \ldots, R_M\}$ of M matrices, to be discussed later.

ii) the change in action, ΔS induced by the variation $U_x^\mu \to U_x^{\mu\,\prime}$ is calculated:

$$\Delta S = S(U_x^{\mu\,\prime},\ldots)-S(U_x^\mu,\ldots); \qquad (2.7)$$

iii) a pseudorandom number r with uniform distribution between 0 and 1 is generated and

$$\tilde{U}_x^\mu = U_x^{\mu\,\prime} \text{ if } r < \exp(-\Delta S) \ ,$$

$$\tilde{U}_x^\mu = U_x^\mu \text{ otherwise.}$$

The steps i) to iii) define what is called a "hit" on one of the link variables. These steps are repeated N_h (number of hits) times. This completes the upgrading of one (link) variable U_x^μ. One MC iteration (or one sweep of the lattice) is executed when all the variables have been processed in this manner.

A crucial consideration for the whole algorithm and also for its vectorization is that the calculation of the variation of the action ΔS involves only a few of the dynamical variables apart from U_x^μ itself, namely those defined on the remaining links of the six plaquettes which share the link between x and $x+\hat{\mu}$. It is convenient to be slightly detailed at this point and to introduce some terminology. Given the link from x to $x+\hat{\mu}$ there are three "forward" plaquettes incident on it, namely those with vertices

$$x,\ x+\hat{\mu},\ x+\hat{\mu}+\hat{\nu} \text{ and } x+\hat{\nu} \ ,$$

(ν taking the three values $\neq \mu$) and three "backward" plaquettes, namely those with vertices

$$x, \quad x+\hat{\mu}, \; x+\hat{\mu}-\hat{\nu} \text{ and } \quad x-\hat{\nu} \quad ,$$

(see Fig. 1).

We shall define as the "force" acting on U_x^μ the sum of the expressions

$$F_{f,x}^{\mu\nu} = U_{x+\hat{\mu}}^{\mu\dagger} \, U_{x+\hat{\nu}}^{\mu} \, U_x^{\nu} \tag{2.8}$$

(corresponding to the forward plaquettes) and

$$F_{b,x}^{\mu\nu} = U_{x+\hat{\mu}-\hat{\nu}}^{\nu} U_{x-\hat{\nu}}^{\mu} U_{x-\hat{\nu}}^{\nu\dagger} \tag{2.9}$$

(corresponding to the backward plaquettes) over the three values of $\nu \neq \mu$

$$F_x^\mu = \sum_{\nu\neq\mu} (F_{f,x}^{\mu\nu} + F_{b,x}^{\mu\nu}) \quad . \tag{2.10}$$

One can easily convince oneself that of the terms contributing to the action in Eqn. 2.2 all those containing U_x^μ can be written in the form

$$\beta[1 - \tfrac{1}{3} \, \mathrm{ReTr}(F_x^{\mu\dagger} U_x^\mu)] \quad , \tag{2.11}$$

and therefore

$$\Delta S = -\frac{\beta}{3}\,\mathrm{ReTr}[F_x^{\mu\dagger}(U_x^{\mu\prime} - U_x^{\mu})] \quad . \tag{2.12}$$

Thus, we become aware of two fundamental facts:

i) once the force F_x^{μ} is calculated, the N_h subsequent hits on the link variable U_x^{μ} can be done without any further recourse to the values of other U variables.

ii) several upgradings can be done in parallel, provided only that the forces F_x^{μ} required for the computation do not involve any of the U_x^{μ} variables that are simultaneously upgraded.

While point i) is relevant for any MC simulation, point ii) acquires particular importance if one wants to write a vectorized code. Indeed, as we shall show, all U_x^{μ} variables with fixed μ can be separated into two sets such that the forces for one set only involve elements of the other. Then, all the U_x^{μ} variables belonging to one set can be grouped together in an array and upgraded simultaneously. Finally one proceeds to upgrade the elements of the other set (the red-black or checkerboard algorithm [4]). We will see in the next section that the ability to separate the link variables into two independent sets is a key to efficient vectorization.

3. The Vectorized Implementation of the Algorithm

The previous discussion has demonstrated that Monte Carlo lattice gauge theories are worthy candidates for vector processing. Until recently, however, people were doubtful as to whether the vector capabilities of current

supercomputers can be effectively utilized for such applications. The main source for this skepticism is the inherent conflict between random access to data, an integral part of a Monte Carlo process, and the strict order of data elements required for pipelined computations. In other words, unless data can be "gathered" at rates comparable to computation rates no efficient vectorization can be achieved.

One of the major strengths of the CDC CYBER 205, and what makes it a particularly powerful Monte Carlo machine, is the ability to order a random collection of data by means of a vector instruction, namely, the "Gather" instruction. This instruction is equivalent to a series of random, or, indirect "load" operations on a serial computer. The Gather instruction uses a vector of integers as an "index-list" pointing to the elements to be fetched. These elements are stored in the order they have been encountered into an output vector. The result rate for the Gather operation is one element every 1.25 cycles (a cycle, or clock-period on the CDC CYBER 205 is 20 nonoseconds). For a comparison, note that the floating-point arithmetic rate, excluding division, is one element every cycle per pipe for 64-bit operands. The CYBER 205 hardware also supports 32-bit operations with twice the result rate for vector floating-point operations. For example, on a two pipe machine 32-bit arithmetic is performed at a rate of 5 nsec per result, or 200 MFLOPS.

The effective utilization of the computational tools build into the vector processor is closely related to the data structure, as are most of the important algorithmic decisions. It is, therefore, appropriate, at this point, to discuss the memory requirements. A 3×3 complex matrix is represented by 18 real numbers. The constraints

of being unitary and unimodular reduce the number of independent parameters to 8, but such a minimal representation of the U_x^μ variables implies a substantial increase in the computational complexity. To obtain optimal performance it is useful to keep all the 18 values representing the real and imaginary parts of the elements of U_x^μ . For a lattice with $n_s = n_t = 16$ a configuration will be defined by $18 \times 4 \times 16^4 =$ 4.718592 million values, which may be more than can be put in the fast memory of many computer systems. Fortunately, the sequential nature of the MC algorithm suggests that only a fraction of the variables need to be in memory at any one time. The others can be kept on disk. The factors which determine an optimal size for the partition between variables in memory and on disk are the following:

i) the partition should not make the code unnecessarily complicated;

ii) the I/O operations should not take longer than the actual computations;

iii)sufficiently long vectors should be available.

On the basis of the above requirements we decided to upgrade one space at a time, i.e. to upgrade all the $4n_s^3$ variables U_x^μ with fixed time coordinate x_4 , and then to proceed to the next x_4 etc. We shall refer to this procedure as time-slicing and to the collection of variables with fixed time coordinate x_4 as one time-slice of the system. If the variables with a given $x_4 = t$ are being upgraded, the calculation of the force requires knowledge of the U_x^μ with $x_4 = t-1$, $x_4 = t$ and $x_4 = t+1$. Thus 3 time slices need to be in memory throughout this stage of the calculation. As a matter of fact, since I/O operations can proceed independently from CPU operations, it

is possible to achieve concurrency of I/O and CPU operations if extra memory buffer space is allocated for holding the $x_4 = t-2$ slice (to be written out), and the $x_4 = t+2$ slice (to be read in). The conventional way of implementing concurrent I/O is to allocate space for two more slices. The resulting five slices in memory act as a circular buffer as shown in Fig. 2. However, the virtual memory hardware on the CDC CYBER 205, and the supporting software provide the capability to swap data between disk and memory. Hence, the memory area of one slice only is needed to write out the $x_4 = t-2$ slice, and read in the $x_4 = t+2$ slice. Consequently, the total memory requirements for the link variables are thus $4 \times n_s^3 \times 4 \times 18$ locations. Allowing for some additional work-space we find that lattices with $n_s = 16$ can be considered in a machine with 2m words (16m bytes) in full precision (64-bit words) and $n_s = 20$ in half precision (32-bit words). The length in time does not constitute a problem any longer and lattices with any n_t may be simulated.

With the slicing mechanism in place we now turn to vectorization aspects of the code. In Sec. 2, the Red-Black ordering was introduced. The motivation for this choice merits some discussion. The computation involves, mainly, matrix multiplications. This operation is easily vectorized, but the matrices concerned are 3×3 matrices, and the resulting vectors are going to be 3 elements long. For efficiently vectorized code one needs to seek longer vectors. This results from the observation that the timing formula for a vector instruction may be written as

$$(\text{Start-up} + \alpha \cdot N) \text{ cycles} \qquad (3.1)$$

where the start-up time is a constant, independent of the vector length. It amounts to aligning the input and output streams, filling up the pipelines up to the point where the first result is available and storing the last result. The start-up time is also independent of the number of pipelines and whether 64-bit or 32-bit arithmetic is performed. On the CDC CYBER 205 it amounts to about 50 cycles, or 1 μsec. The "$\alpha \cdot N$" term is known as the "stream time". N is the number of elements in the vector, so that the stream time is proportional to the vector length. α is a constant associated with the number of pipelines and the arithmetic mode. Table 3.1 contains the α values for some relevant circumstances. It is now obvious that high performance is achieved by minimizing the number of "start-ups" as a consequence of using longer vectors, or, increasing N for each vector operation.

The SU(3) matrices are too small as an object for vectorization; however, there are n_s^3 such matrices in every time slice. One cannot use all of these link values simultaneously because -

i) updating each link requires all its immediate neighbors, and

ii) the correct convergence of the Metropolis process depends upon using "new" values as soon as they are available.

The Red-Black (checker-board) ordering resolves this apparent recursive relationship. The separation of the U_x^μ variables into two sets, for each value of μ and at fixed x_4, is achieved by putting in the two sets all the variables belonging to links originating from odd and even sites, respectively, i.e. with $x_1 + x_2 + x_3 = 1$ or 0 (mod 2). This assures the independence of the forces F_x^μ from the variables U_x^μ being upgraded. On a lattice with $n_s = 16$ the above separation gives a vector length of $n_s^3/2 = 2048$, sufficiently large to insure almost optimal

performance (in fact, 91% and 95% in 32-bit and 64-bit arithmetic, respectively). The calculation of the force F_x^μ requires knowledge of the U_x^μ variables associated with links neighboring the one under consideration. Because of boundary conditions, which we take to be periodic, the variables which enter the calculation of F_x^μ will not, in general, have a simple location-index relative to U_x^μ in the array of dimension $n_s^3/2$. This is easily remedied by the introduction of auxiliary integer-valued arrays, where the indices of the various neighbors of U_x^μ are prestored. The Gather instruction plays a crucial role in the way these index arrays are used. When F_x^μ is evaluated, all the needed variables are gathered into temporary arrays, so that the indices of all elements entering into the computation of F_x^μ are the same, and this proceeds in a fully vectorized manner.

Once the F_x^μ's are determined the algorithm for the upgrading of all the U_x^μ (in the same set) is straightforward and completely vectorizable. The matrices R which are used for finding the new candidates $U_x^{\mu\,\prime}$, are Gathered according to an array of indices extracted at random from a table. The table contains M $SU(3)$ matrices which have a distribution centered around the identity of the group and are obtained in the following fashion. For each value of i between 1 and $M/2$ (M must be even) an eight component vector V_k with approximately gaussian distribution and $\langle V_k^2 \rangle = 1$ is pseodoramdomly generated. The fourth-order approximation to R_i is given by

$$R_i^0 \simeq 1 + iA - \frac{A^2}{2} - \frac{iA^3}{3!} + \frac{A^4}{4!} \simeq \exp(iA) \quad , \tag{3.2}$$

where

$$A = b \sum_k V_k \lambda_k \quad . \tag{3.3}$$

λ_k are Gell-Mann's matrices (i.e., a set of generators of the Lie algebra of SU(3)) and b is a real parameter specifying the spread of the distribution. The final value for R_i is obtained by normalizing R_i^0 to a unitary-unimodular matrix. In general, if we denote the three columns of an SU(3) matrix by $\vec{r}_1, \vec{r}_2$ and $\vec{r}_3$ the constraint of being unitary and unimodular is expressed by

$$\begin{aligned} |\vec{r}_1|^2 &= |\vec{r}_2|^2 = 1 \\ \vec{r}_1 \cdot \vec{r}_2^* &= 0 \\ \text{and} \qquad \vec{r}_3 &= (\vec{r}_1 \times \vec{r}_2)^* \quad . \end{aligned} \tag{3.4}$$

Given a matrix R^0 with the first two columns $\vec{r}_1^0$ and $\vec{r}_2^0$, with $\vec{r}_1^0 \times \vec{r}_2^0 \neq 0$, we shall define as the normalized form of R^0 the matrix R with columns

$$\begin{aligned} \vec{r}_1 &= \vec{r}_1^0 / \sqrt{|\vec{r}_1^0|^2} \\ \vec{r}_2 &= \{\vec{r}_2^0 - \vec{r}_1 (\vec{r}_1^*, \vec{r}_2^0)\} / \sqrt{|\vec{r}_2^0 - \vec{r}_1 (\vec{r}_1^* \cdot \vec{r}_2^0)|^2} \\ \text{and} \qquad \vec{r}_3 &= (\vec{r}_1 \times \vec{r}_2)^* \end{aligned} \tag{3.5}$$

The reason for the nonmenclature is due to the fact that, if R^0 differs slightly from a unitary-unimodular matrix, e.g. as a consequence of roundoff errors, then R is an $SU(3)$ matrix close in value to R^0 Thus, the approximately unitary-unimodular matrix R_i^0 obtained by truncated exponentiation in Eqn. 3.2 is converted to a proper SU(3) matrix R_i by normalization. The last $M/2$ matrices are obtained by

$$R_{\frac{M}{2}+i} = R_i^{+} \qquad (1 \leq i \leq \frac{M}{2}) \tag{3.6}$$

so as to insure that, together with any given matrix R_i , the inverse should also belong to the table.

The procedure for normalizing the $SU(3)$ matrices of the random table, as described above, is also applied, every few iterations, to the link matrices. This is done to insure that the group symmetry of the matrices is preserved regardless of rounding errors which may be introduced by the hardware after many arithmetic operations. This renormalization process is particularly important when the computations are performed using low precision arithmetic. It gives us confidence, which was also tested and verified, in using 32-bit arithmetic for our calculations on the CDC CYBER 205.

Once $U_x^{\mu'}$ is determined, using the table of random $SU(3)$ matrices, the action difference is obtained by calculating, separately,

$$\mathrm{ReTr}(F_x^{\mu+} U_x^{\mu}) \quad \text{and} \quad \mathrm{ReTr}(F_x^{\mu+} U_x^{\mu'})$$

(notice that $\mathrm{ReTr}(A^{+}B)$ is the vector product of the arrays containing

the real and imaginary parts of A and B) , forming an array with exp(-ΔS) , comparing with an array of pseudorandom numbers and accepting or rejecting the change, via a masking operation, according to the outcome of the vectorized comparison between the random numbers and the exponentiated action differences. These steps are repeated for a prefixed number of hits before commencing the upgrade of the other set or the variables corresponding to different directions.

The conditional acceptance of elements in a vector, or, the masking operation referred to above, is handled through the usage of a "bit-vector" (the CDC CYBER 205 is bit addressable and the software allows the Fortran user to use this feature). It is exploited as a part of the vector instruction, and inhibits storing results where zeros are encountered in the bit-vector.

The reader should by now realize that many thousands of random numbers are required for each iteration. The conventional congruent method for generating random numbers is recursive, and may be described by

$$y_{i+1} = (a \cdot y_i) \bmod (b) \qquad (3.7)$$

where a is the "multiplier" and b is determined so as y_{i+1} will be approximately the lower half of the coefficient of the product $a \cdot y_i$ The nature of this calculation suggests that in order to produce N random numbers one has to repeat it serially N times. There is, however, a way to reproduce the same sequence of N numbers in parallel, using vector instructions [7]. Define a new multiplier by

$$A = (a^N)\mathrm{mod}(b)$$
$$= (\ldots(a*a)\mathrm{mod}(b)*a)\mathrm{mod}(b)\ldots*a)\mathrm{mod}(b) \qquad (3.8)$$

and let

$$\underline{Y}_1 = (y_1, y_2, \ldots, y_N) \qquad (3.9)$$

be the vector containing the first N random numbers.
Then

$$\underline{Y}_{i+1} = (A*\underline{Y}_i)\mathrm{mod}(b) \qquad (3.10)$$

reproduces the same sequence of random numbers one gets with a repeated application of Eqn. 3.7 (the computation of Eqn. 3.10 requires only 3 vector operations on the CDC CYBER 205).

To conclude this section, let us discuss the way matrix multiplication is done, being the most time-consuming aspect of the computation . First, the reader will remember that we do not vectorize the matrix multiplication as such, but, rather, perform the operations on many matrices in parallel, where for each matrix the "scalar" sequence of operations is followed.

When computing the products of two SU(3) matrices, one need not evaluate all the columns of the result, since the third column of the product matrix (which is again unitary-unimodular) is related to the first two by Eqn. 3.4. In the code we have exploited this fact whenever possible. It is particularly advantageous when several SU(3) matrices must be multiplied together, since one may limit the calculations to two columns out of three in all intermediate products and simply reconstruct the third column of the final result as shown in Eqn. 3.4.

Finally, all complex arithmetic has been done in terms of real variables, separating real and imaginary parts (which would also result in a more efficient code for a scalar machine), and we have used the identity

$$(A+iB)(C+i\,D) = (A+B)(C-D)-BC + AD + i(BC+AD) \qquad (3.11)$$

to perform the product of two complex matrices in terms of three real multiplications and five real matrix additions. Using complex arithmetic the product of two matrices would require four real multiplications and two additions. Due to the fact that matrix multiplication requires $2N^3$ operations, where N is the dimension of the matrix, and matrix addition requires only N^2 operations, our method pays off even for $N = 3$.

A schematic outline of the flow of the calculations is shown in Fig. 3.

4. Performance and Timings

The figures quoted here are based on runs executed on a two-pipe, 2m 64-bit words CDC CYBER 205. They apply to a 16^4 lattice ($n_s = 16$, $n_t = 16$), SU(3) gauge theory with 10 hits per link upgrade (unless stated explicitly otherwise). We present performance figures for both 64-bit and 32-bit arithmetic operations. In both modes the exponentiation and the generation of random numbers were carried out using 64-bit arithmetic. It should be noted here that due to our slicing mechanism the 32-bit version requires real memory of only 852,000 words (64-bit words, or 6.8m bytes), so it actually fits comfortably on a 1m words system. With these parameters

the code performs at 98% CPU utilization. The 64-bit version requires, of course, twice as much memory.

In Table 4.1 we give the percentage of the execution time for the two arithmetic modes spent in the force (F_x^μ) and the Metropolis updating calculations. It becomes clear from these figures why it is worth while using a single force computation for a number of attempts at updating (rather than the one attempt proposed by the original Metropolis method).

It should be added here the normalization procedure discussed in Sec. 3, performed every 5 iterations adds only 0.74% and 0.59% in 64-bit and 32-bit modes, respectively, to the total execution time.

Table 4.2 presents a percentage breakdown of the code by operation type. The reader will notice that the Gather, random number generation and the exponentiation operations are more heavily weighted in the 32-bit mode compared with that of the 64-bit mode. These three types of operations perform at the same rate in both modes. The last two execute in 64-bit mode in both versions of the code. The Gather instruction performs at the same rate regardless of whether the operands are 64-bit or 32-bit variables. This is because the performance of the Gather operation is driven by memory access (and not by computation complexity). The matrix multiplication, being made up of floating-point operations only, executes at near peak rate of 95 MFLOPS and 182 MFLOPS for the 64-bit and 32-bit modes,respectively. The effect of vectorizing the random number generator can be illustrated by noting that this operation amounted to 6% (64-bit) and 11% (32-bit) of the total time when it was not vectorized. The "action" involves taking the real part of the trace of products of SU(3) matrices (purely floating-point operations). The "acceptance" is the portion of the code where the conditional acceptance

of new U_x^μ matrices occurs under the control of a bit-vector created for that purpose.

The actual time for one iteration of the 16^4 lattice with 10 hits is 16.27 secs. (64-bit) and 10.72 secs. (32-bit). This amounts to a substained performance rate of 66.8 MFLOPS (64-bit) and 101.5 MFLOPS (32-bit). Another way, commonly used by physicists, to express the performance of Monte Carlo lattice gauge theories implemented on a computer system, is the link update time, i.e., the time needed to update one link of the lattice once. This measure is useful for comparisons since it is independent of the lattice size. The link update times (in μsecs.) for our implementation are given in Table 4.3. These figures may be compared to a link update time of about 1,100 μsecs on the CDC 7600 computer system with a highly optimized code.

Acknowledgements

We would like to thank Control Data Corporation for awarding time on the CDC CYBER 205 at the Institute for Computational Studies at Colorado State University where the code described in the text was developed. One of the authors (K.J.M.M.) would like to thank Dalhousie University for the award of a Visiting Fellowship which made his visit to Fort Collins, Colorado possible. This research was also carried out in part under the auspices of the US Department of Energy under contract No. DE-AC02-76CH00016.

Table 3.1. Stream rate proportionality factor (α) .

No. of pipes \ Arithmetic mode	64-bit	32-bit
2	1/2	1/4
4	1/4	1/8

Table 4.1. Breakdown by percentage of sections of code.

	64-bit	32-bit
force	43.49	42.46
update	56.40	57.40

Table 4.2. Breakdown by percentage of the main operation types.

operation type	64-bit	32-bit
matrix multiplication	58.33	47.05
Gather	20.78	29.27
random number generator	0.95	1.83
exponentiation	7.43	11.72
action	5.93	4.70
acceptance	3.62	3.01

Table 4.3. The upgrades times for a link (in μsecs).

number of hits	64-bit	32-bit
10	62.1	40.9
8	55.1	36.3

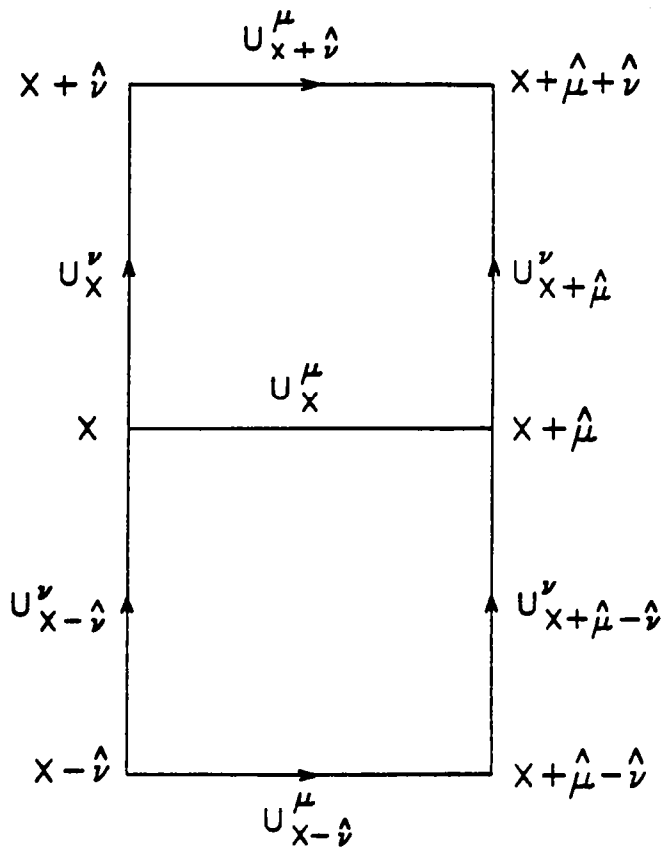


Figure 1. "Forward" (upper half) and "backward" (lower half) plaquettes in the μ-ν plane, where $x \equiv (x_1,x_2,x_3,x_4)$ is a point in our four-dimensional lattice. This is one out of three such planes which can be formed in a four-dimensional space.

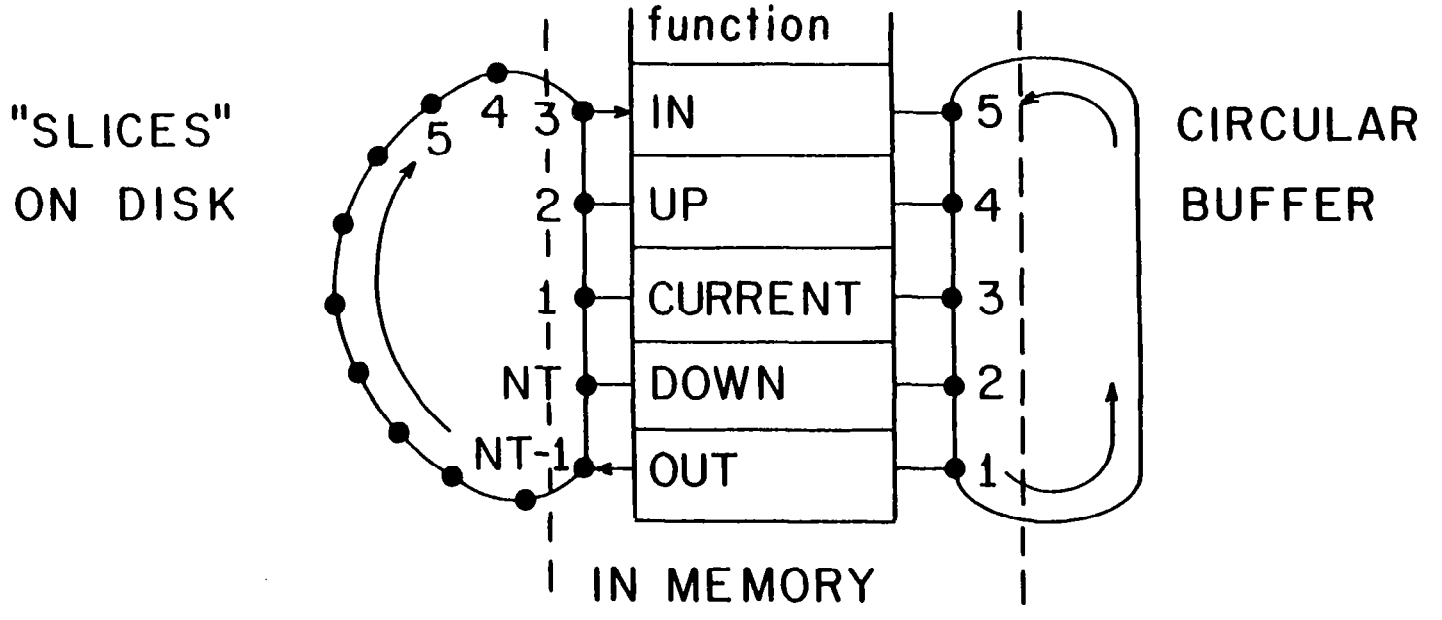


Figure 2. The I/O scheme. In our implementation the "in" and "out" boxes occupy the same physical memory.

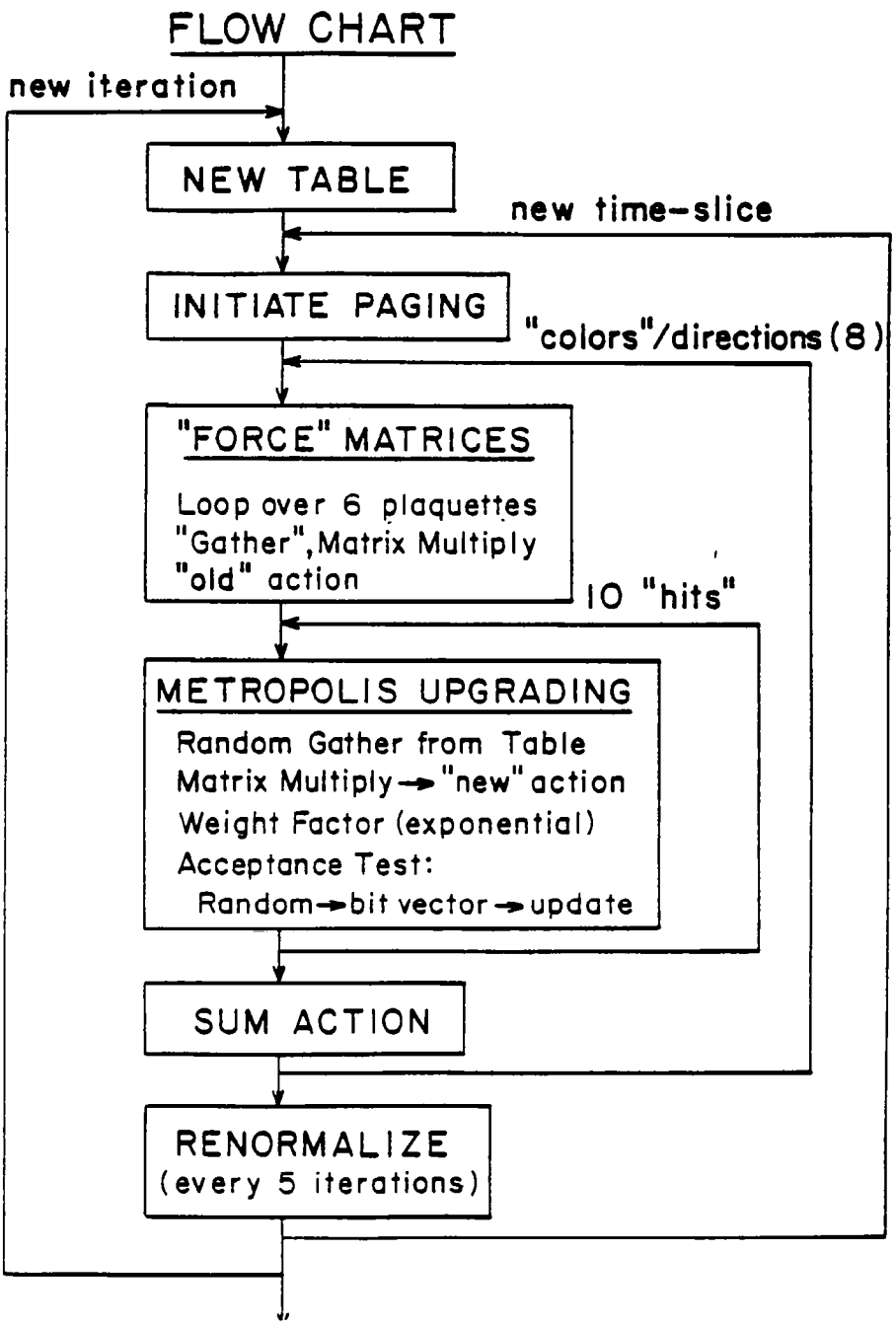


Figure 3. Schematic description of the computational process.

References

[1] K.G. Wilson, Phys. Rev. D10, 2445 (1974).

[2] J.-M. Drouffe and J.-B. Zuber, Phys. Reports (to be published).

[3] M. Creutz, L. Jacobs and C. Rebbi, Phys. Reports 95, 201(1983).

[4] D. Barkai and K.J.M. Moriarty, Comput. Phys. Commun. 25, 57(1982); 26, 477(1982); 27, 105(1982); D. Barkai, M. Creutz and K.J.M. Moriarty, Comput. Phys. Commun. 30, 13(1983).

[5] D. Barkai, K.J.M. Moriarty and C. Rebbi, (to be published).

[6] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller and E. Teller, J. Chem. Phys. 21, 1087(1953).

[7] Forrest Brown, private communication.

ADAPTING ITERATIVE ALGORITHMS FOR SOLVING LARGE SPARSE LINEAR SYSTEMS FOR EFFICIENT USE ON THE CDC CYBER 205

DAVID R. KINCAID
AND
DAVID M. YOUNG

CENTER FOR NUMERICAL ANALYSIS
UNIVERSITY OF TEXAS AT AUSTIN

AUSTIN, TEXAS

ADAPTING ITERATIVE ALGORITHMS FOR SOLVING LARGE SPARSE LINEAR SYSTEMS FOR EFFICIENT USE ON THE CDC CYBER 205

David R. Kincaid
David M. Young

Center for Numerical Analysis
University of Texas at Austin
Austin, TX 78712

* Adapting and designing mathematical software to achieve optimum performance on the CYBER 205 will be discussed

* Comments and observations are made in light of recent work done at the Center for Numerical Analysis on

  - modifying the ITPACK software package

  - writing new software for vector supercomputers

Research goal - develop very efficient vector algorithms and software for solving large sparse linear systems using iterative methods

(older) SCALAR APPROACH - develop algorithms that minimize either number of iterations or arithmetic operations

* Not necessarily the correct approach for vector computers *

(newer) VECTOR APPROACH - avoid operations such as table lookups, indirect addressing, etc. that are inefficient on a vector computer, i.e., non-vectorizable

* Fully vectorizable code may involve more arithmetic operations but can be executed at a very high rate of speed *

* Advances in high performance computers and in computer architecture necessitates additional research in mathematical software to find suitable algorithms for the supercomputers of today and of the future *

THE VECTORIZATION OF THE ITPACK SOFTWARE PACKAGE

Scalar ITPACK:

package for solving large sparse linear systems
7 iterative algorithms available
sparse storage format used
Kincaid, Respess, Young, & Grimes [1982]
ITPACK 2C (ALGORITHM 586) in T.O.M.S.
"Transactions on Mathematical Software"

VECTORIZATION:

- First step: look for obvious vectorization changes since this was a large package of over 11,000 lines of code and we did not want to completely rewrite it

- Vector ITPACK (standard Fortran version): used a minimum of vector syntax available in CYBER 200 Fortran for a portable version of Vector ITPACK 2C

- Vector ITPACK (CYBER 205 version): a modified version of Vector ITPACK written using CYBER 200 Fortran vector syntax where possible

- DO loops which had been unrolled for scalar optimization were not recognized as vectorizable by optimizing vector compilers

- These loops were rewritten as simple tight DO loops so that they would be executed in vector mode

- The sparse storage scheme used for the matrix in Scalar ITPACK was row-oriented and inhibited vectorization (The IA-JA-A data structure as in Yale software YSMP used.)

- A column-oriented data structure was used in Vector ITPACK to increase vectorization (The COEF-JCOEF data structure as in Purdue software ELLPACK used.)

- The version of Vector ITPACK specifically for the CYBER 205 was tested on the CYBER 205 at Colorado State University (CSU) and has been added to their Program Library

- The improvements in time of the vector syntax version over the one in standard Fortran were not as significant as we had anticipated

- The automatic vectorization available in the CYBER 205 Fortran compiler did a very good job of optimization and vectorization

Moral: vector syntax best when used in designing and writing new code

PROBLEM:

$$\begin{cases} u_{xx} + 2\,u_{yy} = 0 & \text{on } S=(0,1)x(0,1) \\ u = 1 + xy & \text{on boundary of } S \end{cases}$$

Discretization: standard 5-point finite difference formula

Stopping Criterion: 5.0×10^{-6}

Mesh Sizes: 1/16; 1/32; 1/64; 1/128; 1/256

Number of Unknowns: 225; 961; 3969; 16,129; 65,025

Computer: CSU CYBER 205

CYBER 200 Fortran: Large pages, unsafe vectorization

Scalar ITPACK (unrolled DO-loops & YALE storage used; T.O.M.S. version)

Modified Scalar ITPACK (rolled DO-loops & minor changes: Q8SDOT used)

Vector ITPACK (rolled DO-loops, ELLPACK storage, & CYBER 200 Fortran vector syntax used)

TABLE I: CHANGING SPARSE STORAGE

(Iteration Times in Seconds with H = 1/64)

Method	Iterations	Scalar ITPACK	Modified Scalar ITPACK	Vector ITPACK
(Natural Ordering)				
JACOBI CG	178	2.509	2.184	.262
JACOBI SI	362	5.214	4.480	.580
SOR	216	4.700	4.597	2.453
SSOR CG	34	1.976	1.788	.831
SSOR SI	43	1.791	1.682	.970
(Red-Black Ordering)				
JACOBI CG	178	2.402	2.056	.268
JACOBI SI	362	4.987	4.209	.590
SOR	196	4.110	4.017	.523
SSOR CG	341	20.327	18.472	2.177
SSOR SI	196	7.734	6.690	.701
RS CG	90	1.445	1.358	.118
RS SI	182	2.980	2.779	.223

TABLE II: CHANGING PROBLEM SIZE

(Number of Iterations)

Method	H = 1/16	1/32	1/64	1/128	1/256
(Natural Ordering)					
JACOBI CG	49	94	178	330	629
JACOBI SI	56	179	362	772	1372
SOR	50	104	216	422	872
SSOR CG	16	22	34	51	73
SSOR SI	19	29	43	61	88
(Red-Black Ordering)					
JACOBI CG	49	94	178	330	629
JACOBI SI	56	179	362	772	1372
SOR	52	101	196	396	839
SSOR CG	34	62	341	1058	3061
SSOR SI	51	107	196	373	752
RS CG	25	48	90	167	321
RS SI	42	88	182	375	704

TABLE III: CHANGING PROBLEM SIZE

(Iteration Time in Seconds)

Method	H = 1/16	1/32	1/64	1/128	1/256
(Natural Ordering)					
JACOBI CG	.010	.040	.251	1.800	14.115
JACOBI SI	.014	.091	.560	4.196	28.741
SOR	.035	.292	2.446	19.828	164.940
SSOR CG	.027	.133	.828	4.953	28.157
SSOR SI	.029	.163	.967	5.583	32.249
(Red-Black Ordering)					
JACOBI CG	.010	.041	.257	1.847	14.511
JACOBI SI	.013	.091	.571	4.277	29.394
SOR	.011	.066	.475	4.028	34.939
SSOR CG	.018	.075	2.105	25.779	302.712
SSOR SI	.021	.113	.663	4.452	36.053
RS CG	.006	.019	.109	.757	5.981
RS SI	.008	.033	.207	1.557	11.881

COMMENTS ON TABLE I

- Two versions of Scalar ITPACK were compared with the CYBER 205 version of Vector ITPACK

- Mesh size H = 1/64 used for all runs

- Scalar ITPACK: unrolled DO-loops used in basic vector operations for increased optimization on scalar computers

- Modified Scalar ITPACK: standard tight DO-loops used

- Vector Fortran compiler recognizes tight loops as vectorizable but not unrolled loops

- A slight increase in speed from Scalar to Modified Scalar version

- Vector ITPACK uses tight loops, Fortran vector syntax, and a column-oriented sparse storage scheme

- This data structure allows the matrix-vector product operation to vectorize to a great extent

* Considerable improvement in performance from scalar to vector version of ITPACK *

COMMENTS ON TABLE II & III

- These tables are results of using Vector ITPACK on the same problem with varying mesh sizes

- The number of iterations increase as the problem size increase

- Comparisons based on number of iterations misleading as to the best method!

- On scalar computers, SOR with natural ordering is widely used while JACOBI is not but on vector computers ...

- Most efficient method on the CYBER 205:

JACOBI CG method when natural ordering is used

RS CG when red-black ordering is used

SCALAR ITPACK vs. VECTOR ITPACK

- Total time for each method is not significantly greater than the iteration time in the vector version (this was not the case in the scalar version)

- Only N additional workspace locations required for the vector version over the scalar version

- Faster scaling and permuting of the system with the column-oriented sparse storage scheme

- Improved performance of the SSOR methods with the red-black ordering in the vector version in spite of the greater number of iterations

A PRE-CONDITIONED CONJUGATE GRADIENT PACKAGE

Thomas C. Oppe, a graduate student at UT Austin, is working on a package which allows flexibility in the choice of basic methods and acceleration schemes.

The package has been designed to make the addition of further preconditionings and acceleration schemes easy.

Particular attention has been paid to the choice of matrix storage schemes with a view to maximizing vectorizability.

Features of Package:

- Conjugate Gradient Acceleration

- Pre-conditioning matrix Q (Jacobi, Symmetric Successive Overrelaxation, Reduced System, Incomplete Cholesky, Modified Incomplete Cholesky, Neumann Polynomial, Parameterized Polynomials, Other pre conditionings planned such as Incomplete Block Cyclic Reduction)

- Realistic Stopping Tests

- Automatic estimation of iteration parameters with adaptive procedures

- Two possible data structures allowed

DATA STRUCTURES

Data structures which allow vectorization to varying degree:

EXAMPLE:

$$A = \begin{vmatrix} 4 & -1 & -2 & 0 \\ -1 & 4 & 0 & -2 \\ -2 & 0 & 4 & -1 \\ 0 & -2 & -1 & 4 \end{vmatrix}$$

ELLPACK Data Structure:

$$COEF = \begin{vmatrix} 4 & -1 & -2 \\ 4 & -2 & -1 \\ 4 & -1 & -2 \\ 4 & -2 & -1 \end{vmatrix} \qquad JCOEF = \begin{vmatrix} 1 & 2 & 3 \\ 2 & 4 & 1 \\ 3 & 4 & 1 \\ 4 & 2 & 3 \end{vmatrix}$$

- matrix-vector product vectorizes with the use of gathering routines

- operations such as forward (back) substitutions using lower (upper) triangular matrices do not vectorize

DIAGONAL Data Structure:

$$COEF = \begin{vmatrix} 4 & -1 & -2 \\ 4 & 0 & -2 \\ 4 & -1 & * \\ 4 & * & * \end{vmatrix} \qquad JCOEF = (0,\ 1,\ 2)$$

- the matrix-vector product operation vectorizes without the use of gathering routines

- operations such as forward (back) substitution and factorizations vectorize to some extent

REFERENCES

David R. Kincaid, John R. Respess, David M. Young, and Roger G. Grimes, "ALGORITHM 586 ITPACK 2C: A FORTRAN Package for Solving Large Sparse Linear Systems by Adaptive Accelerated Iterative Methods", ACM Transactions on Mathematical Software, Vol. 8, No. 3, September 1982.

David R. Kincaid, Tom Oppe, and David M. Young, "Adapting ITPACK Routines for Use on Vector Computers," Report CNA-177, Center for Numerical Analysis, University of Texas at Austin, TX, August 1982. (In the Proceedings of the 1982 Symposium on CYBER 205 Applications, Institute for Computational Studies at Colorado State University, Fort Collins, CO.)

David R. Kincaid and Thomas C. Oppe, "ITPACK on Supercomputers", Report CNA-178, Center for Numerical Analysis, University of Texas at Austin, TX, September 1982. (To appear in the Proceedings of the InterAmerican Workshop on Numerical Methods, Springer-Verlag, NY.)

David R. Kincaid and David M. Young, Jr., "The ITPACK Project: Past, Present, and Future", Report CNA-180, Center for Numerical Analysis, University of Texas at Austin, TX, March 1983. (To appear in ELLIPTIC PROBLEM SOLVERS II, Academic Press, NY.)

FUNDAMENTAL ORGANOMETALLIC REACTIONS: APPLICATIONS ON THE CYBER 205

ANTHONY K. RAPPE'

COLORADO STATE UNIVERSITY

FORT COLLINS, COLORADO

Fundamental Organometallic Reactions: Applications on the Cyber 205

A. K. Rappe'
Colorado State University
Fort Collins, CO 80523

Abstract

Two of the most challenging problems of Organometallic chemistry (loosely defined) are pollution control with the large space velocities needed and nitrogen fixation, a process so capably done by nature and so relatively poorly done by man (industry). For a computational chemist these problems are on the fringe of what is possible with conventional computers (large models needed and accurate energetics required). A summary of the algorithmic modification needed to address these problems on a vector processor such as the Cyber 205 and a sketch of our findings to date on deNOx catalysis and nitrogen fixation are presented.

Introduction

Two of the most challenging problems in Organometallic chemistry (loosely defined) are pollution control with the large space velocities needed and nitrogen fixation, a process so capably done by nature and so relatively poorly done by man (industry). For a computational chemist these problems (and other similar problems) are on the fringe of what is possible with conventional computers (large models needed and accurate energetics required). The advent of vector processors such as the Cyber 205 is making such studies feasible. A summary of the algorithmic modification needed to address these problems on a vector processor is presented in section I, a sketch of the findings to date for deNOx catalysis is presented in section II, and finally a sketch of the nitrogen fixation results is presented in section III.

I. Algorithmic Modification.

The advent of vector processors is leading to a reexamination of fundamental computational algorithms of general use to computational chemists and the redesign of large scale codes. The present work illustrates both processes for the Cyber 205 computer. Reexamination of fundamental algorithms is illustrated with an examination of the similarity transform, a matrix operation of use to computational chemists. Large scale code redesign is examined through the implementation of a highly vectorized MC-SCF code.

A. Similarity Transform. A common sequence of matrix operations is the similarity transform

$$C = A^T B A \qquad (1).$$

For computational chemistry applications the matrices B and C are usually symmetric and generally stored in lower diagonal form. If the initial B matrix is expanded from upper diagonal form to full matrix representation vector operations are possible for both matrix multiplications. The linked triad instruction on the Cyber 205 is utilized for the first matrix multiplication and a vector dot product operation is used for the second matrix multiplication. In principle one could transpose matrix A and to use the linked triad instruction for both matrix multiplications; however, in this case since we only want slightly more than half of the final results the vector dot product is preferable as it permits selective manipulation of the column indices I and J. As is apparent from Table I the vectorized matrix transformation represents a substantial improvement over scalar mode with enhancements ranging from a factor of 10 to a factor of 40. Note for the 300x300 matrix case we are still approximately a factor of 2 off the maximum rate for the Cyber 205. The consideration of an algorithm where several matrices are transformed at once is in order. In addition it should be noted from Table I that the expansion from lower diagonal form does not add a significant cost (less than 10 percent). Finally, it should be apparent that the MFLOPS rate will be independent of the number of orbitals involved (indices I and J); the vectorized loops run over number of functions not orbitals (indices K and L).

B. SCF Coding Considerations. The fundamental kernel of self consistent field (SCF) codes in general[1,2] is the energy expression

$$E^{el} = \sum_{i,j}^{n} D_j^i \, h_{ij} + \sum_{\substack{i,j \\ k,l}}^{n} D_{kl}^{ij} (ik|jl) \qquad (5),$$

where

$$h_{ij} = \sum_{\mu,\nu}^{m} C_\mu^i C_\nu^j \langle X_\nu | h | X_\nu \rangle \qquad (6)$$

$$(ik|jl) = \sum_{\mu,\nu,\sigma,n}^{m} C_\mu^i C_\nu^k C_\sigma^j C_n^l \langle X_\mu(1) X\sigma(2) | r_{12}^{-1} | X_\nu(1) X_n(2) \rangle \qquad (7)$$

The integrals $\langle X_\mu | h | X_\nu \rangle$ and $\langle X_n X_\sigma | r_{12}^{-1} | X_\nu X_n \rangle$ need only be evaluated once (for a given geometric point), stored conveniently, and repeatively accessed during the orbital coefficient (C_μ^i) and density matrix element ($D_j^i; D_{kl}^{ij}$) optimization stages. For the Restricted Hartree Fock (RHF) wavefunction $D_i^i = 2$, $D_{ij}^{ij} = 2$, $D_{ji}^{ij} = -1$, and the remaining terms are zero.[1] For wavefunctions beyond RHF the wavefunction optimization step represents a vast majority of the time needed to variationally determine E, that is, the calculation of the X_μ integrals is usually relatively insignificant.[2] For this reason initial vectorization efforts have concentrated on enhancing the time intensive stages of an MCSCF (multiconfiguration SCF) program. It is generally accepted[2] that one of the most time intensive steps of a general MCSCF code is the 4 index transformation needed to convert the X_μ integrals to ϕ_i integrals where

$$\phi_i = \sum_\mu C_\mu^i X_\mu \qquad (5).$$

On scalar processors only the unique integrals are stored (the Canonical list) and the loops are structured so as to minimize the number of multiplications performed. On a vector processor

such as the Cyber 205 this step simply amounts to two sequential applications of the matrix transformation described in (1). This transformation will proceed at vector speed provided that for a given ij pair all kl integrals are available for k>l (this corresponds to an effective doubling of the integral file from its canonical length). This expansion of the canonical integral tape is accomplished through a straightforward two level bin sort written to take advantage of the 2 million 64 bit words available on the Cyber 205[3]. Since the vectorizable portions of this integral transform are contained in the matrix transform discussed above, the timing information in Table I applies here. Four index transformations for 50 basis functions will proceed at 28 MFLOPS and 300 basis function transformations in general will achieve 82 MFLOPS. Enhancements over scalar computation on the Cyber 205 will range from a factor of 9 to a factor of 34 for 50 to 300 basis functions. For example, a full integral transformation for 50 basis functions will maximumly take 28 seconds and for 100 basis functions 10 minutes on the Cyber 205.

For a wide class of useful wavefunctions (open-shell HF and perfect pairing-generalized valence bond [GVB-PP] are two such examples) the one- and two- electron density matrices D^i_j and D^{ij}_{kl} are expressible in diagonal form;[1] that is, the only nonzero elements are

$$D^i_i = 2f_i,\ D^{ij}_{ij} = a_{ij},\ \text{and}\ D^{ij}_{ji} = b_{ij} \qquad (6).$$

The energy expresion (2) simplifies to

$$E = 2\sum_i^n f_i h_{ii} + \sum_{i,j}^n (a_{ij}J_{ij} + b_{ij}K_{ij}) \qquad (7).$$

where

$$J_{ij} = (ii/jj) \text{ and } K_{ij} = (ij/ij) \tag{8}$$

are the usual Coulomb and exchange integrals. Restricting our attention to this class of wavefunction leads to particularly simple variational equations[1] partitionable into a step where occupied and virtual orbitals are mixed variationally (OCBSE)[4] and a step where independent occupied orbitals are mixed through pairwise rotations.[5] The OCBSE step utilizes terms representable as a vectorizable summation of J_i and K_i operators

$$\langle X_\mu | J_i | X_\nu \rangle \text{ and } \langle X_\mu | K_i | X_\nu \rangle \tag{9}$$

where

$$\langle X_\mu | J_i | X_\nu \rangle = \sum_{\sigma,n} C_\sigma^i, C_n^i \; (\mu\nu|\sigma n)$$

$$\langle X_\mu | K_i | X_\nu \rangle = \sum_{\sigma,n} C_\sigma^i, C_n^i \; (\mu\sigma|\nu n) \tag{10}.$$

That is

$$\langle X_\mu | H_i | X_\nu \rangle = \sum_j a_{ij} \langle X_\mu | J_j | X_\nu \rangle + b_{ij} \langle X_\mu | K_j | X_\nu \rangle \tag{11},$$

where a set of loops can be written (which are in linked triad form and will run at >170 MFLOPS for more than 50 basis functions) to evaluate the Ith hamiltonian (K runs from 1 to n(n+1)/2).

```
      DO 300 J=1,NHAM
      A = A(I,J)
      B = B(I,J)
      DO 100 K = 1, MXS
  100 H(K)=H(K)+A*AJ(K,J)
      DO 200 K=1,MXS
  200 H(K)=H(K)+B*AK(K,J)
```
(12)

As the rotations step utilizes a subset of the above integrals, the needed vectorization effort is narrowed down to rapidly

generating the terms in (10). If all σ,n terms σ>n are stored for a given μν the double sums in (10) can be reduced to a single dot product over a combined index γ of length n(n+1)/2

$$\langle X_\mu | J_i | X_\nu \rangle = \sum_\gamma D_\gamma^i J_\gamma^{\mu\nu}$$
$$\langle X_\mu | K_i | X_\nu \rangle = \sum_\gamma D_\gamma^i J_\gamma^{\mu\nu} \qquad (13)$$

where

$$D_\gamma^i = C_\sigma^i C_n^i$$
$$J_\gamma^{\mu\nu} = (\mu\nu/\sigma n) \qquad (14).$$
$$K_\gamma^{\mu\nu} = ((\mu\sigma/\nu n) + (\mu n/\nu\sigma))/2$$

Currently the D_γ^i are precalculated, stored, and used for an entire SCF iteration. Formulating the problem as in (13) permits vectors ranging from 1275 for 50 basis functions to 45150 for 300 basis functions. This step will function at between 80 and 100 MFLOPS representing enhancements of between 40 and 50 over scalar computation on the Cyber 205. Table II summarizes the timing for calculations ranging up to a 79 basis function calculation consisting of 4096 spatial configurations; that is, a GVB-PP(12/24) wavefunction.[1] If the calculation were stopped after the RHF step the SCF would represent less than 1% of the computational effort. Overall the GVB(12/24) wavefunction optimization represents 14% of the total effort. This is in sharp contrast to computations on scalar computers where this step would account for greater than 95% of the effort. The timing for an SCF iterative cycle for three cases is broken down in Table III. Note that the time needed to generate the terms in (13) is comparable to that needed to diagonalize the variational hamiltonians (OCBSE).

II. DeNOx Catalysis.

The catalytic reduction of nitrogen oxides has become increasingly important in recent years due to legislation aimed at reducing emission levels from non-biological sources[6]. As Nitric Oxide is the major NO_x component of exhaust streams[7] research has focused on the reduction of nitric oxide. Both homogeneous and heterogenous deNO_x studies have been performed[8-11]. The use of base-metal catalysts is of particular interest due to their ready availability and low cost. A transition metal ion of singular importance in pollution control is Fe(II) either as the bulk oxide or ion exchanged into zeolites. These Iron systems have been demonstrated to catalyze the conversion of nitric oxide to nitrogen with a co-reactant such as CO or H_2[8,9]. The mechanism originally proposed by Shelef and Kummer[12] consists of a two stage oxidation reduction sequence. The initial step involves the coupling of two nitric oxides to form nitrous oxide plus an Iron oxide.

$$2NO \longrightarrow N_2O + \text{'O'} \qquad (15)$$

The thus formed nitrous oxide is rapidly reduced by the catalyst[8b,8d,10].

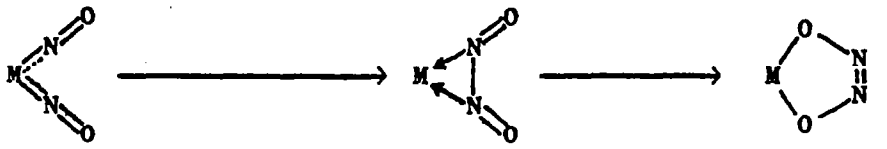
$$N_2O \longrightarrow N_2 + \text{'O'} \qquad (16)$$

Completing the cycle the Iron oxide is reduced by reaction with carbon monoxide forming carbon dioxide plus the regenerated catalytic site.

$$\text{'O'} + CO \longrightarrow CO_2 \qquad (17)$$

Efforts have primarily been directed at characterizing reaction (15) as this is likely to be the kinetically most difficult

step[8d]. For homogeneous systems (15) has been suggested to involve an intramolecular coupling of nitrosyls to form a dinitrogen dioxide ligand[11a] which rearranges to a bound cis hyponitrite.



(18)

1 2 3

Metal hyponitrites have been established to either decompose to nitrous oxide and the metal oxide[13a] or react with carbon monoxide to from carbon dioxide and nitrous oxide[13b-c].

It should be stressed that transition metal dinitrogen dioxide complexes have never been isolated nor unambiguously detected. Further, only a single mononuclear transition metal hyponitrite complex has been identified[13b].

In this section we report energetic support for the reaction sequence (18) for a model Fe(II) system: the dinitrosyl complex of Iron dichloride $FeCl_2(NO)_2$[14]. The relative energetics[15] and geometries[16] for the chosen complex 1, its coupled cognate dinitrogen dioxide complex 2, and the cis hyponitrite product 3, are discussed below. We find that the coupled products are potentially accessible; 2 is only 29 kcal/mol higher in energy than 1 and 3 only another 19 kcal/mol higher. These species, though unobserved, should be viable given an appropriate ligand backbone. Addition of waters of hydration profoundly affects the relative energies of the hdyrated forms of 1, 2, and 3 (4, 5, and 6 respectively). We find that intermediates 5 and 6 are thermally accessible. Intermediate 5 is 24 kcal/mol more stable

than 4 and 6 is only 4 kcal/mol above 4. This is not suprising as 1 is a 16 electron system, 2 is a 14 electron system, and 3 is a 12 electron system (unusual participation by the pi lone pairs was not observed in the wavefunction of 3 or 6).

A correlation of the bonding orbitals demonstrates that the coupling reaction 1 to 2 or 4 to 5 will be thermally allowed (occupied reactant orbitals correlate with occupied product orbitals[17]). Further, the LUMO is a non-bonding d orbital of B_2 symmetry indicating that this correlation diagram will be valid for systems with up to 2 more electrons. Finally, one of the high lying occupied orbitals is a non-bonding A_1 d orbital suggesting that the correlation diagram will be valid for systems with up to two fewer electrons. Thus group VI through group VIII metal dications are potential active catalysts.

Because Fe(II) dinitrosyls are structurally uncharacterized, because only a single transition metal hyponitrite complex has been structurally characterized, and because dinitrogen dioxide complexes are unprecedented a detailed discussion of the bond distances and bond angles that were optimized is in order. We find the N-Fe-N angle for the dinitrosyl is 94.9 degrees, as expected for a $\{M(NO)2\}^8$ system[16b]. The Fe-N distance of 1.69 A is in agreement with experimental structures for linear Iron dinitrosyls (1.66 A[18a] to 1.71 A[18b]). For the dinitrogen dioxide complex 2 we find a N-N distance of 1.53 A, longer than normal N-N single bonds (ranging from 1.402 A to 1.492 A[19]) but still significantly shorter than that for free dinitrogen dioxide (2.24 A[20]). This is consistent with substantial nitrogen-nitrogen sigma

bonding. The Fe-N distance found for the dinitrogen dioxide complex (2.23 A) is in accord with the Fe(II) nitrogen bond distance of 2.26 A[21] in $[Fe(C_4H_8NH)_6][Fe_4(CO)_{13}]$. Finally, for the cis hyponitrite complex 3 our Fe-O distance of 1.74 A compares favorably with 1.69 A (the sum of the ionic radii for OH^- (1.18 A) and an estimate for the ionic radius for four coordinate Fe(IV) (0.51 A)[22]). Our N-N distance of 1.21 A is the same as the N-N distance determined by X-ray crystallography for $[(Ph_3P)_2Pt(N_2O_2)]$[13b], the only structurally characterized hyponitrite.

Summarizing, we have demonstrated that (17) is a probable reaction sequence for group VI through group VIII transition metal $deNO_x$ catalysts. Specifically our energetics and correlation diagram suggest that dinitrogen dioxides are thermodynamically and kinetically accessible cognates of dinitrosyl complexes. We believe that these results can be extended to heterogeneous Fe(II) catalyzed $deNO_x$ processes as well. In fact we speculate that the stretching frequencies observed by Hall[8c] at 1917 cm^{-1} and 1815 cm^{-1} are due to bound dinitrogen dioxide which is blue shifted relative to the free compound (which has frequencies[23] at 1870 cm^{-1} and 1776 cm^{-1}. Because the coordination sphere of Fe(II) ion exchanged into zeolites is thought[24] to contain three oxygen ligands our energetics suggest the frequencies assigned to a dinitrosyl are instead due to the kinetically accessible and thermodynamically favored dinitrogen dioxide moiety. Further, it should be noted that dinitrosyl stretching frequencies as high as 1900 cm^{-1} are rare. In conclusion we suggest that the kinetically (and thermodynamical-

ly) most difficult step in (17) is the isomerization of the dinitrogen dioxide complex 2 (or 5) to the cis hyponitrite complex 3 (or 6).

III. Nitrogen fixation.

The fixation of dinitrogen is a reductive process of both biological and large scale industrial interest. Thermodynamically the conversion of dinitrogen to ammonia is straightforward and the conversion to hydrazine is feasible under high pressures (ΔG_{298} for these processes are -7.9 kcal/mol and +22.0 kcal/mol respectively; if the pressure is increased to 100 atm then the ΔG_{298} for hydrazine formation is +16.7 kcal/mol).

In the known nitrogen-fixing organisms the catalytic reduction of dinitrogen is carried out by molybdoenzymes known as nitrogenases[25]. These nitrogen-fixing enzymes consist of two protein components, a Fe-Mo protein and a Fe protein. Further, an iron-molybdenum cofactor has been isolated from the Fe-Mo component protein of nitrogenase. In fact extracts of the Mo-Fe component from inactive mutant strains of microorganisms are activated by addition of this cofactor. Two models of the active site have been proposed that are consistent with Mossbauer and EPR spectroscopic data[26] and EXAFS analysis[27] of the Fe-Mo cofactor. Unfortunately the models of such active sites synthesized to date do not reduce dinitrogen[28-30].

Industrially, dinitrogen reduction occurs over an Iron catalyst at high temperatures and pressures. The rate determining step is either the dissociative chemisorbtion of dinitrogen[31]

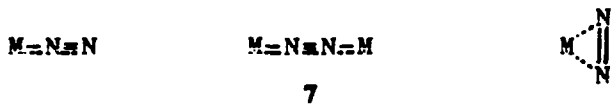
$$2* + N_2 \longrightarrow 2N-* \qquad (19)$$

or the simple chemisorbtion of an activated form of dinitrogen

$$* + N_2 \longrightarrow N_2\text{-}* \qquad (20)$$

Both of these processes are likely followed by rapid reaction with hydrogen (either molecular hydrogen of chemisorbed atomic hydrogen).

Thus, for both biological and industrial nitrification the activation of dinitrogen is a prerequisite for reaction with reductants such as hydrogen. Until very recently the observed forms of dinitrogen were bound to the metal with the nitrogen-nitrogen multiple bond largely intact (non-activated).



(21)

7

Thus these model compounds will only reduce dinitrogen under rather harsh conditions[32].

An understanding of a recently observed dinitrogen binding mode (analogous to organic azines)

(22)

8

will provide additional insight into biological and industrial nitrification. The reactivity and structural characteristics of a new class of Tantalum complexes[33] suggest the bonding pattern 8 in (22). The Ta-N bond distances of 1.796 A and 1.840 A are quite similar to those observed in normal Tantalum imido complexes[33] (1.765 A to 1.77 A). In addition, reactions (23) and (24) are both observed[33] (reactions characteristic of metal-ligand multiple bonding).

$$M{=}N{-}R \;+\; R'_2C{=}O \longrightarrow R'_2C{=}NR \;+\; 'M{=}O' \qquad (23)$$

$$M{=}N{-}N{=}M \;+\; 2R_2C{=}O \longrightarrow R_2C{=}N{-}N{=}C_2R \;+\; 2'M{=}O' \qquad (24)$$

Finally, there is an observable 'activation' of the nitrogen-nitrogen bond (N-N bond distances of 1.282 A and 1.298 A compared to free dinitrogen which has a N-N bond distance of 1.0976 A).

In this section we report energetic support for the kinetic and thermodynamic accessibility of **8** for molybdenum complexes. Our model consists of a bimetallic complex consisting of two Molybdenumtetrachloride units bridged by a dinitrogen molecule. For this complex we have characterized the 'reaction path' connecting the two likely resonance structures **7** and **8**

$$Cl_4Mo{-}N{=}N{-}MoCl_4 \longleftrightarrow Cl_4Mo{=}N{-}N{=}MoCl_4 \qquad (25)$$

9 **10**

We find local minima characteristic of each resonance structure indicating the 'resonance' interaction between these two forms is not enough to result in a single averaged structure[34]. However, the resonance interaction is sufficient to provide a very low barrier interconnecting them (less than 1 kcal/mol). Thermodynamically we find **9** to be 20 kcal/mol more stable than **10** for the tetrachloride ligand backbone. This thermodynamic difference could easily be overcome by an alteration of the ligand backbone and future studies will concentrate on this. Geometrically, for **9** the Mo-N distance is 2.28 A and the N-N distance is 1.10 A and for **10** the Mo-N distance is 1.82 A and the N-N distance 1.23 A. This is in accord with a suggestion that the tetrachloride backbone does not fully activate the dinitrogen (a fully activated N-N distance should be on the order of 1.30 A).

Table I. Comparison of Scalar and Vector Matrix Transformations.
(for various sized matrices, times in sec.)

Matrix size NxN)	Scalar (with Opt.) First Mult.	Second Mult.	Total Time	Vector (times x 100) Expand Array	First Mult.	Second Mult.	Total Time	Ratio (S/V)	MFLOPS (vec.)
50	0.041	0.083	0.124	0.063	0.78	0.51	1.36	9.1	27.8
100	0.32	0.65	0.96	0.23	3.65	2.59	6.48	14.8	46.5
150	1.07	2.58	3.64	0.51	9.34	6.91	16.76	21.7	60.5
200	2.52	6.74	9.25	1.01	19.34	14.32	34.67	26.7	69.3
250	5.39	14.35	19.74	1.83	33.43	25.64	60.90	32.4	77.1
300	9.90	27.14	37.03	2.92	53.22	42.23	109.84	33.7	82.4

Table II. Timing Breakdown for MC-SCF Energy Generation.
(times in seconds)

Step	Molecule/No. of basis functions			
	H_2O/7	$FeCl_2 \cdot (H_2O)_2$/43	$FeCl_2(NO)_2$/65	$FeCl_2(NO)_2(H_2O)_2$/79
Calculate One electron Integrals	0.13	36.4	48.5	81.0
Calculate Two electron Integrals	1.06	86.6	191.7	535.5
Sort Two Electron Integrals	0.05	14.7	94.3	247.7
Generate Extended Huckel Starting Guess	----	0.8	1.1	----
Obtain Hartree Fock Energy (10 it.)	0.11	1.8	3.1	----
Obtain MC-SCF Energy (10 it.)	----	----	72.5	137.5
	----	-----	-----	-----
Total Time	1.35	140.3	411.2	1001.7
% of Time				
HF	8.1	1.3	0.8	----
MC-SCF	----	----	17.5	13.7

Table III. SCF Timing Breakdown for an Individual Cycle.
(Times in seconds, rates in MFLOPS)

Wavefunction Description	Generate J_i and K_i Matrices		Transform J_i and K_i Matrices	OCBSE	Orbital Rotations	Optimize a_{ij} and b_{ij}	Total
	Time	Rate	Time	Time	Time	Time	Time
H_2O MBS HF	0.0001	4.6	0.006	0.004	----	----	0.011
$FeCl_2 \cdot (H_2O)_2$ HF	0.0082	49.0	0.017	0.078	----	----	0.177
$FeCl_2(NO)_2$ HF	0.0310	60.6	0.034	0.241	----	----	0.306
GVB(12/24)	2.012	81.4	2.832	1.990	0.328	0.091	7.253
$FeCl_2(NO)_2(H_2O)_2$ GVB(12/24)	4.302	88.2	5.322	3.515	0.516	0.090	13.745

Acknowlegement

We are grateful to the donors of the Petroleum Research Fund for support of the noncomputational aspects of this research. Further, the generous computational support of the Institute for Computational Studies at CSU is gratefully acknowledged.

References

1. Bobrowicz, F. W.; Goddard III, W. A. "Modern Theoretical Chemisty: Methods of Electronic Structure Theory", H. F. Schaefer III, Ed. Plenum Press, New York, NY, 1977 Vol. 3, p. 79.

2. Das, G.; Wahl, A. C. J. Chem. Phys., 1972, 56, 1769; Hinze, J. J. Chem. Phys., 1973, 59, 6424; Yaffe, L. G.; Goddard III, W. A. Phys. Rev. A, 1976, 13, 1682; Wahl, A. C.; Das, G. "Modern Theoretical Chemistry: Methods of Electronic Structure Theory", H. F. Schaefer III, Ed., Plenum Press, New York, NY, 1977 Vol. 3, p. 51; Dalgaard, E.; Jorgensen, P J. Chem. Phys., 1978, 69, 3833; Roothaan, C. C. J.; Detrich, J.; Hopper, D. G.; Int. J. Quantum Chem. Symp. 1979, 13, 93; Dalgaard, E. Chem. Phys. Lett., 1979, 65, 559; Jorgensen, P.; Yeager, D. J. Chem. Phys., 1979, 71, 757; Shepard, R.; Simons, J. Int. J. Quantum Chem. Symp. 1980, 14, 211; Lengsfield, B. H. J. Chem. Phys., 1980, 73, 382; Brooks, B. R.; Laidig, W. D.; Saxe, P.; Schaefer III, H. F. J. Chem. Phys., 1980, 72, 3837; Roos, B.; Taylor, P.; Siegbahn, P. Chem. Phys., 1980, 48, 157; Siegbahn, P.; Almlof, J.; Heiberg, A.; Roos, B. J. Chem. Phys., 1981, 74, 2384; Lengsfield III, B. H.; Liu, B. J. Chem. Phys., 1981, 75, 478; Shepard, R.; Shavitt, I.; Simons, J. J. Chem. Phys., 1982, 76, 543; Lengsfield III, B. H. J. Chem. Phys., 1982, 77, 4073.

3. Rappe', A. K. (1982) unpublished, tape input and driver routine from SORTIJK by Bair, R. A. and Goddard III, W. A. (1977) unpublished. This program does a bin sort where each bin is 1.5 milion words long, as many bins are used as needed to process the tape in a single pass. Significant vectorization has not yet been implemented.

4. Hunt, W. J.; Dunning Jr., T. H.; Goddard III, W. A. Chem. Phys. Lett., 1969, 3, 606.

5. Hunt, W. J.; Goddard III, W. A.; Dunning Jr., T. H. Chem. Phys. Lett., 1970, 6, 147.

6. a) Dwyer, F. G. Catal. Rev., 1972, 6, 261-291. b) Heywood, J. B. Prog. Energy Combust. Sci., 1976, 4, 135-164. c) Martin, G. B.; Heap, M. P. AIChE Symp. Series, 1977, 73, 349-365. d) Rosenberg, H. S.; Currau, L. M.; Slack, A. V.; Ando, J.; Oxley, J. H. Prog. Energy Combust. Sci., 1980, 6, 287-302. e) Kummer, J. T. Prog. Energy Combust. Sci., 1980, 6, 177-199.

7. Harrison, B.; Wyatt, M.; Gough, K. G. Catalysis, 1981, 5, 127-171.

8. a) Fu, C. M.; Deeba, M.; Hall, W. K. Ind. Eng. Chem. Prod. Res. Dev., 1980, 19, 299-304. b) Fu, C. M.; Korchak, V. N.; Hall, W. K.; J. Catal., 1981, 68, 166-171. c) Segawa, K. I.; Chen, Y.; Kubsh, J. E.; Delgass, W. N.; Dumesic, J. A.; Hall, W. K. J. Catal., 1982, 76, 112-132. d) Petunchi, J. O.; Hall, W. K. J. Catal., 1982, 78, 327-340.

9. a) McCandless, F. P.; Hodgson, K. M.; White, R. H.; Bowman, J. D.; Ind. Eng. Chem. Process Des. Dev., 1980, 19, 108-133. b) Courty, P.; Raynal, B.; Rebours, B.; Prigent, M.; Sugier, A. Ind. Eng. Chem. Prod. Res. Dev., 1980, 19, 226-231. c) Busca, G.; Lorenzeiil, V. J. Catal., 1981, 72, 303-313. d) Yang, R. T.; Li, K. T. Ind. Eng. Chem. Res. Dev., 1982, 21, 405-408. e) Yuen, S.; Chen, Y.; Kubsh, J. E.; Dumesic, J. A.; Topsoe, N.; Topsoe, H. J. Phys. Chem., 1982, 86, 3022-3032.

10. a) Lorenzelli, V.; Busca, G. B.; Al-Mashta, F.; Sheppard, N. J. Catal., 1981, 72, 389-391. b) Kappes, M. M.; Staley, R. H. J. Am. Chem. Soc., 1981, 103, 1286-1287.

11. a) Haymore, B. L.; Ibers, J. A. J. Am. Chem. Soc., 1974, 96, 3325-3327. b) Eisenberg, R.; Hendricksen, D. E. Adv. in Catalysis, 1979, 28, 79-172 and references within. c) Bottomley, F. ; Lin, I. J. B.; Mukaida, M. J. Am. Chem. Soc., 1980, 102, 5238-5242. d) Kaduk, J. A.; Tulip, T. H.; Budge, J. R.; Ibers, J. A. J. Mol. Catal., 1981, 12, 239-243.

12. Shelef, M.; Kummer, J. T. AIChE Symp. Ser., 1971, 67, 74-92.

13. a) Oza, T. M.; Oza, V. T. J. Chem. Soc., 1953, 909-913. b) Bhaduri, S.; Johnson, B. F. G.; Pickard, A.; Raithby, P. R.; Sheldrick, G. M.; Zuccaro, C. I. J. Chem. Soc. Chem. Comm. 1977, 354-355. c) Bhaduri, S. A.; Bratt, I.; Johnson, B. F. G.; Khair, A.; Segal, J. A.; Walters, R.; Zuccaro, C. J. Chem. Soc. Dalton, 1981, 234-239.

14. a) The reduced system $FeCl_2(NO)_2^-$ has been prepared and characterized (14b-e). In addition aqueous Fe(II) has been reported to reduce NO to N_2O. This reaction is thought to involve $Fe(NO)_2^{+2}$ (14f,g). b) Silverthorn, W.; Feltham, R. D. Inorg. Chem., 1967, 6, 1662-1666. c) Martini, G.; Tiezzi, E. Trans. Faraday Soc., 1971, 67, 2538-2547. d) Gwost, D.; Caulton, K. G. Inorg. Chem., 1973, 12, 2095-2099. e) Connelly, N. G.; Gardner, C. J. Chem. Soc. Dalton, 1976, 1525-1527. f) Bonner, F. T.; Pearsall, K. A. Inorg. Chem., 1982, 21, 1973-1978. g) Pearsall, K. A.; Bonner, F. T. Inorg. Chem., 1982, 21, 1978-1985.

15. a) The energetics reported are the differences between GVB-CI (15b) calculations for the three species, the differential effects due to waters of hydration were obtained with a GVB-PP (15c) wavefunction (the waters were treated at the Hartree Fock level). The pairs of electrons explicitly correlated were the ones shown in figure 3, the N-O sigma bonds and the nitrogen s lone pairs (a GVB(12/24) wavefunction). Within this 24 orbital space a R-CI(4) (15d) plus R-CI(1) times singles CI (15e) was performed with a maximum of eight open shell electrons (a total of 11499 spin eigenfunctions and 48921 determinants). Effective potentials were utilized on Fe (15f) and Cl (15g). The basis set on Cl was a minimum basis set optimized for $TiCl_4$ (15h). The s and p basis on Fe was the valence portion of Wachters basis (15i) augmented with core functions analogous to those used on Cl (15g) (s exponent=0.4907936, p exponent=0.1350391). The d basis was

the five gaussian basis described previously (15j). The N and O basis sets were valence double zeta as discussed previously (15h). The basis sets on the waters of hydration were minimum basis sets where the linear parameters were optimized for $FeCl_2(H_2O)_2$, the O exponents as above, and the H exponents were Huzinaga's four quassian set (15k) (scaled by 1.2). b) Davis, J. H.; Goddard III, W. A.; Harding, L. B. J. Am. Chem. Soc., 1977, 99, 2919-2925. c) Bobrowicz, F. W.; Goddard III, W. A. In "Modern Theoretical Chemistry: Methods of Electronic Structure Theory", H. F. Schaefer III, Ed., Plenum Press: New York, 1977 Vol. 3 Chapter 4, pp 79-127. d) Harding, L. B.; Goddard III, W. A. J. Am. Chem. Soc., 1976, 98, 6093-6099. e) Casewit, C. J.; Goddard III, W. A. ibid., 1982, 104, 3280-3287. f) Melius, C. F.; Olafson, B. D.; Goddard III, W. A. Chem. Phys. Lett., 1974, 28, 457-462. g) Rappe', A. K.; Smedley, T. A.; Goddard III, W. A. J. Phys. Chem., 1981, 85, 1662-1666. h) Rappe', A. K.; Goddard III, W. A. In "Potential Energy Surfaces and Dynamics Calculations", D. G. Truhlar, Ed., Plenum Press, New York, 1981 pp. 661-684. i) Wachters, A. J. H. J. Chem. Phys., 1970, 52, 1033-1036. j) Rappe', A. K.; Smedley, T. A.; Goddard III, W. A. J. Phys. Chem., 1981, 85, 2607-2611. k) Huzinaga, S. J. Chem. Phys., 1965, 42, 1293-1302.

16. a) For all the systems the Fe-Cl distance was fixed at 2.30 A (16b). For 1 the Fe-N distance and the N-Fe-N angle were optimized utilizing a GVB-CI (15). The N-O distance was held fixed at 1.15 A a value appropriate for linear nitrosyls(16c). For 1, 2, and 3 the Cl-Fe-Cl angle was fixed at 120.0 degrees (larger than a tetrahedral angle as would be expected for such systems). For complex 2 the Fe-N distance, the N-N distance, and the dependent N-Fe-N angle were optimized with an HF wavefunction. The N-O distance was fixed at 1.21 a value appropriate for a N-O double bond (15e). The N-N-O angle of 115 degrees was taken from the parent nitrosamine (15e). For complex 3 the Fe-O distance, the N-N distance and the dependent O-Fe-O angle were optimized with a HF wavefunction. The N-O distance was fixed at 1.41 A for cis hydroxydiimide (15e) (quite close to the 1.39 A found for the Platinum cis-hyponitrite (13b)). The N-N-O angle was taken as 118 degrees again from cis hydroxydiimide (15e) (also in agreement with the angle from the Platinum cis-hyponitrite (118.5 degrees) (13b)). For 4, 5, and 6 nitrogen and oxygen geometries were taken from 1, 2, and 3 respectively. The Cl-Fe-Cl angle was increased to 170.0 degrees. Finally, the geometries of the waters of hydration were taken directly from gas phase H_2O, the O-Fe distance of 2.15 was taken from $FeCl_2$ dihydrate (16d) and the O-Fe-O angle fixed at 90.0 degrees. For 4 the planes containing the hydrogens and oxygens of the waters were taken as perpendicular to the plane containing the Iron and the two nitrogens. For 5 and 6 the water planes were the same as the one containing the Iron and the two nitrogens. b) Extrapolated from analogous Mn and Ru dinitrosyl complexes; Laing, M.; Reimann, R. H.; Singleton, E. Inorg. Chem., 1979, 18, 1648-1653; Pierpont, C. G.; Eisenberg, R. Inorg. Chem., 1972, 11, 1088-1094. c). Feltham, R. E.; Enemark, J. H. In "Topics in Inorganic and Organometallic Stereochemistry", G. Geoffroy Ed., Wiley-Interscience, New York,

1981 pp 155-209. d) Morosin, B.; Graeber, E. J. J. Chem. Phys., 1965, 42, 898-901.

17. Woodward, R. B.; Hoffmann, R. J. Am. Chem. Soc., 1965, 87, 395-397; Woodward, R. B.; Hoffmann, R. "The Conservation of Orbital Symmetry", Verlag Chemie, Gmbh., Weinheim/Bergstrasse, 1970.

18. a) Clegg, W. Inorg. Chem., 1976, 15, 2928-2931. b) Albano, V. G.; Araneo, A.; Bellon, P. L.; Ciani, G.; Manasseri, M. J. Organomet. Chem., 1974, 67, 413-422.

19. a) Bartell, L. S.; Higginbotham, H. K. Inorg. Chem., 1965, 4, 1346-1351. b) Morino, Y.; Iijiama, T.; Murata, Y. Bull. Chem. Soc. Japan, 1960, 33, 46-48. c) Gilbert, M. M.; Gundersen, G.; Hedberg, K. J. Chem. Phys., 1972, 56, 1691-1697.

20. Kukolich, S. G. J. Am. Chem. Soc., 1982, 104, 4715-4716.

21. Doedens, R. J.; Dahl, L. F. J. Am. Chem. Soc., 1966, 88, 4847-4855.

22. Shannon, R. D.; Prewitt, C. T. Acta Cryst., 1969, B25, 925-946; Shannon, R. D. ibid., 1976, A32, 751-767.

23. Fately, W. G.; Bent, H. A.; Crawford, B. J. Chem. Phys., 1959, 31, 204-217.

24. Maxwell, I. E. Adv. in Catal., 1982, 31, 1-76.

25. For a recent review of molybdenum biochemistry, see: "Molybdenum and Molybdenum-Containing Enzymes", M. P. Coughlan, Ed., Pergamon Press: New York 1980.

26. Rawlings, J.; Shah, V. K.; Chisnell, J. R.; Brill, W. J.; Zimmermann, R.; Munck, E.; Orme-Johnson, W. H.; J. Biol. Chem., 1978, 253, 1001.

27. Cramer, S. P.; Hodgson, K. O.; Gillum, W. O.; Mortenson, L. E.; J. Am. Chem. Soc., 1978, 100, 3398; Cramer, S. P.; Gillum, W. O.; Hodgson, K. O.; Mortenson, L. E.; Stiefel, E. I.; Chisnell, J. R.; Brill, W. J.; Shah, V. K. ibid., 3814.

28. Coucouvanis, D. Acc. Chem. Res., 1981, 14, 201.

29. Wolff, T. E.; Berg, J. M.; Warrick, C.; Hodgson, K. O.; Holm, R. H.; Frankel, R. B.; J. Am. Chem. Soc., 1978, 100, 4630; Wolff, T. E.; Berg, J. M.; Warrick, C.; Hodgson, K. O.; Frankel, R. B.; Holm, R. H. ibid., 1979, 101, 4140; Wolff, T. E.; Berg, J. M.; Power, P. P.; Hodgson, K. O.; Holm, R. H.; Frankel, R. B. ibid., 5454; Wolff, T. E.; Power, P. P.; Frankel, R. B.; Holm, R. H. ibid., 1980, 102, 4694; Armstrong, W. H.; Holm, R. H. ibid., 1981, 103, 6246.

30. Christou, G.; Garner, C. D.; Mabbs, F. E. Inorg. Chim. Acta,

1978, 28, L189; Christou, G.; Garner, C. D.; Mabbs, F. E.; King, T. J. J. Chem. Soc., Chem. Commun., 1978, 740; Christou, G.; Garner, C. D.; Mabbs, F. E.; Drew, M. G. B. ibid., 1979, 91; Acott, S. R.; Chritou, G.; Garner, C. D.; King, T. J.; Mabbs, F. E.; Miller, R. M. Inorg. Chim. Acta, 1979, 35, L337; Christou, G.; Garner, C. D.; Miller, R. M. J. Inorg. Biochem., 1979, 11, 349.

31. For recent reviews of the industrial nitrogen fixation process, see: Ertl, G. Catal. Rev.-Sci. Eng., 1980, 21, 201-223; Boudart, M. Catal. Rev.-Sci. Eng., 1981, 23, 1-15.

32. Chatt, J.; Dilworth, J. R.; Richards, R. L. Chem. Rev., 1978, 589-625 and references within.

33. Turner, H. W.; Fellmann, J. D.; Rocklage, S. M.; Schrock, R. R. J. Am. Chem. Soc., 1980, 7811:Rocklage, S. M.; Schrock, R. R. ibid., 7808; Churchill, M. R.; Wasserman, H. J. Inorg. Chem., 1981, 20, 2899; Churchill, M. R.; Wasserman, H. J. ibid., 1982, 21, 223; Churchill, M. R.; Wasserman, H. J. ibid., 278.

34. Pauling, L. ''The Nature of the Chemical Bond'', Cornell University Press, Ithaca, New York 1960.

THREE-DIMENSIONAL FLOW OVER A CONICAL AFTERBODY CONTAINING A CENTERED PROPULSIVE JET: A NUMERICAL SIMULATION

GEORGE S. DEIWERT
NASA AMES RESEARCH CENTER
MOFFET FIELD, CALIFORNIA

HERBERT ROTHMUND
CONTROL DATA CORPORATION
SUNNYVALE, CALIFORNIA

THREE-DIMENSIONAL FLOW OVER A CONICAL AFTERBODY CONTAINING A CENTERED PROPULSIVE JET: A NUMERICAL SIMULATION

George S. Deiwert*
NASA Ames Research Center, Moffett Field, California

Herbert Rothmund†
Control Data Corporation, Sunnyvale, California

Abstract

The supersonic flow field over a body of revolution incident to the free stream is simulated numerically on a large, array processor (the CDC Cyber 205). The configuration is composed of a cone-cylinder forebody followed by a conical afterbody from which emanates a centered, supersonic propulsive jet. The free-stream Mach number is 2, the jet-exit Mach number is 2.5, and the jet-to-free-stream static pressure ratio is 3. Both the external flow and the exhaust are ideal air at a common total temperature. The thin-layer approximation to the time-dependent, compressible, Reynolds-averaged Navier-Stokes equations are solved using an implicit finite-difference algorithm. The data base, of 5 million words, is structured in a "pencil" format so that efficient use of the array processor can be realized. The computer code is completely vectorized to take advantage of the data structure. Turbulence closure is acheived using an empirical algebraic eddy-viscosity model. The configuration and flow conditions correspond to published experimental tests and the computed solutions are consistent with the experimental data.

Introduction

In 1980, a computational study was described in which the three-dimensional flow field over axisymmetric boattailed bodies at moderate angles of attack was simulated.[1] The exhaust plumes were modeled by solid plume simulators, and a second-order-accurate, implicit finite-difference algorithm was used to solve the governing partial differential equations on the ILLIAC IV array processor. Several flow fields were computed and the results compared with published experimental data. The promising results of that first study provided the incentive to extend the work to include propulsive exhaust jets emanating from the afterbody base. The ILLIAC IV was subsequently removed from service, however, and it became necessary to scale down the size and scope of the study to the capacity of existing computer resources.

In January 1983, the results of a study of supersonic axisymmetric flow over boattails containing a centered propulsive jet were presented.[2] Those results, obtained using a Cray 1S computer with 10^6 words of main memory, were compared with existing experimental data. Jet-to-free-stream static pressure ratio and nozzle exit angle were varied parametrically; and the predicted trends agreed well with experiment.

The purpose of this paper is to describe the vectorized implementation of the three-dimensional Navier-Stokes code on a Cyber 205 computer for boattailed afterbodies at moderate angles of attack that contain a centered propulsive jet. Some computed results, which correspond in part to a published experimental study for a like configuration and flow conditions, are included for illustration.

Afterbody Configuration

The geometric configuration is a 9 caliber body of revolution composed of a 14° half-angle conical nose, a cylindrical forebody, and an 8° half-angle conical afterbody of 1 caliber length. Centered inside the afterbody is a conical nozzle with exit diameter of 0.6 caliber that is flush with the afterbody base. The nozzle exit half-angle is 20°.

Experimental studies for the same configuration were performed by White and Agrell[3] for the model immersed in an air stream flowing at $M_\infty = 2.0$ and a jet-exit Mach number of 2.5. White and Agrell considered angles of incidence to the free stream up to 8° and jet-to-free-stream static-pressure ratios up to 15. Because of limited acces to the Cyber 205 computer, computed results are included in this paper only for the case in which the angle of incidence is 6° and the jet-to-free-stream pressure ratio is 3.0.

Governing Equations

The equations describing the flow are the Reynolds-averaged Navier-Stokes equations. These are written below in strong conservative form in generalized coordinates as

$$\partial_t Q + \partial_\xi(F \cdot \vec{g}^\xi) + \partial_\eta(F \cdot \vec{g}^\eta) + \partial_\zeta(F \cdot \vec{g}^\zeta) = 0 \quad (1)$$

*Research Scientist, Member AIAA.
†Computer Analyst.

where

$$Q = J^{-1}\begin{pmatrix}\rho\\ \rho u\\ \rho v\\ \rho w\\ e\end{pmatrix}, \quad F = J^{-1}\begin{pmatrix}\rho\vec{q}\\ \rho u\vec{q} + \tau\cdot\vec{e}_x\\ \rho v\vec{q} + \tau\cdot\vec{e}_y\\ \rho w\vec{q} + \tau\cdot\vec{e}_z\\ e\vec{q} + \tau\cdot\vec{q} - K_e\nabla T\end{pmatrix}$$

and $\vec{e}_x, \vec{e}_y$, and $\vec{e}_z$ are the Cartesian unit vectors and $\vec{g}^\xi, \vec{g}^\eta$, and $\vec{g}^\varsigma$ are the contravariant base vectors, which can be written as

$$\vec{g}^\xi = \xi_x\vec{e}_x + \xi_y\vec{e}_y + \xi_z\vec{e}_z$$
$$\vec{g}^\eta = \eta_x\vec{e}_x + \eta_y\vec{e}_y + \eta_z\vec{e}_z$$
$$\vec{g}^\varsigma = \varsigma_x\vec{e}_x + \varsigma_y\vec{e}_y + \varsigma_z\vec{e}_z$$

The components of momentum, ρu, ρv, and ρw, are in Cartesian space and the velocity vector $\vec{q}$ is generally expressed in terms of the contravariant velocity components, U, V, and W as

$$\vec{q} = u\vec{e}_x + v\vec{e}_y + w\vec{e}_z$$
$$= U\vec{g}_\xi + V\vec{g}_\eta + W\vec{g}_\varsigma$$

where $\vec{g}_\xi, \vec{g}_\eta$, and $\vec{g}_\varsigma$ are the covariant base vectors written as

$$\vec{g}_\xi = x_\xi\vec{e}_x + y_\xi\vec{e}_y + z_\xi\vec{e}_z$$
$$\vec{g}_\eta = x_\eta\vec{e}_x + y_\eta\vec{e}_y + z_\eta\vec{e}_z$$
$$\vec{g}_\varsigma = x_\varsigma\vec{e}_x + y_\varsigma\vec{e}_y + z_\varsigma\vec{e}_z$$

The Jacobian J of the transformation is given by

$$J^{-1} = x_\xi y_\eta z_\varsigma + x_\varsigma y_\xi z_\eta + x_\eta y_\varsigma z_\xi - x_\xi y_\varsigma z_\eta - x_\eta y_\xi z_\varsigma - x_\varsigma y_\eta z_\xi$$

The flux vector F can be decomposed into a parabolic part, F_P, which contains only gradient diffusive terms, and a hyperbolic part, F_H, which contains only convective-like terms, as

$$F_H = \begin{pmatrix}\rho\vec{q}\\ \rho u\vec{q} + p\vec{e}_x\\ \rho v\vec{q} + p\vec{e}_y\\ \rho w\vec{q} + p\vec{e}_z\\ (e+p)\vec{q}\end{pmatrix}, \quad F_P = F - F_H \quad (2)$$

For flows in which the shear layers are thin (when $Re >> 1$) and aligned with one principal plane (say the plane normal to the η coordinate), the parabolic part of F can be neglected in the other two coordinates (ξ and ς), without any real loss in accuracy. This is consistent with boundary-layer theory and yet maintains the coupling between the viscous and inviscid regions that is critical in simulating interactive flows. With this thin-layer approximation, Eq. (1) is rewritten as:

$$\partial_t Q + \partial_\xi(F_H\cdot\vec{g}^\xi) + \partial_\eta(F\cdot\vec{g}^\eta) + \partial_\varsigma(F_H\cdot\vec{g}^\varsigma) = 0 \quad (3)$$

Computational Grid

A body-oriented computational grid is constructed in a manner compatible with the thin-layer approximation. Shown in Fig. 1 is the grid used in the present computations. Figure 1a shows the complete configiration and Fig. 1b the detail in the base region of the afterbody. Radial grid lines on the forebody join the surface orthogonally. On the afterbody and in the exhaust plume, the radial lines are normal to the body axis. There are 81 points distributed along the body,

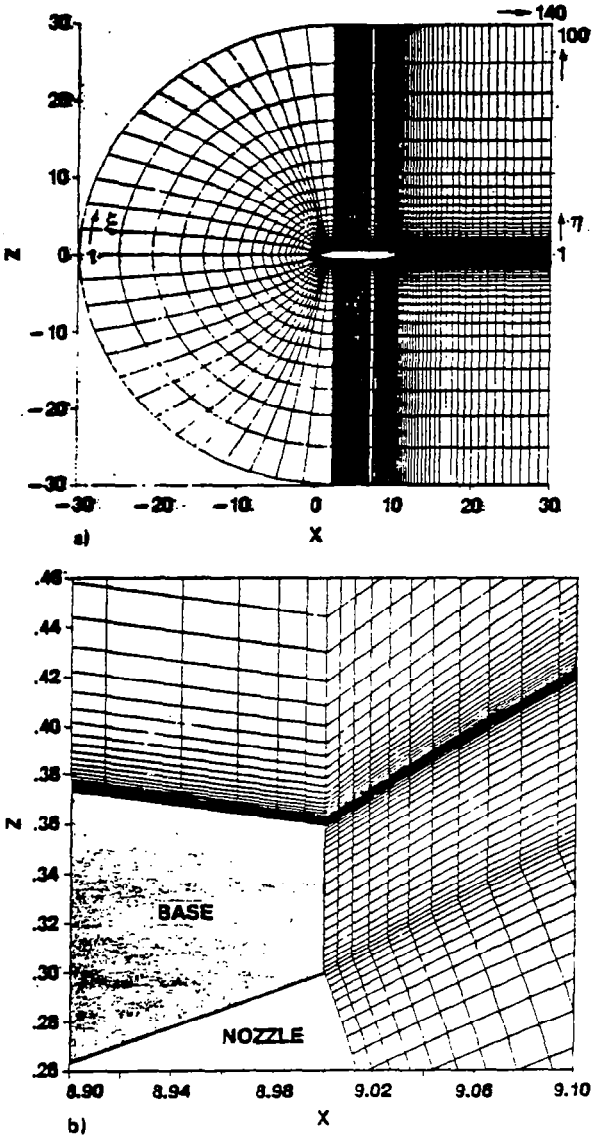


Fig. 1 Computational grid: bilateral plane of symmetry. a) Complete configuration (140 x 100 x 20); b) Base-region detail.

with clustering near the nose and near the base. Of the 81 points, 21 are used to define the afterbody shape; the afterbody is 1 caliber long. An additional 59 points are distributed downstream of the afterbody to a distance equal to 21 forebody diameters from the nozzle base. These 140 total points define the ξ coordinate distribution. The radial distribution, corresponding to the η coordinate, extends from the body surface to a distance equal to 30 forebody diameters both ahead of the nose and normal to the body axis. A total of 60 points is used in this region, with a high degree of streching used in order to resolve the sublayer of the turbulent boundary layer. (Here the first grid point off the body surface corresponds approximately to a value of η^+ of 8 where $\eta^+ = (\rho_w \tau_w)^{1/2}(\eta - \eta_w)/\mu_w$.) An additional 40 points are distributed across the nozzle and its blunt base, extending from the centerline to the body surface. Of these, 20 are in the jet exit plane and 20 are on the blunt base itself.

One- and two-parameter hyperbolic-tangent streching functions[4] are used in the base region to focus resolution near the corners and to achieve a smooth, piecewise continuous distribution of points across the exhaust plume and base. At the nozzle exit, points are distributed along an arc describing the conical flow exit plane (that is, the arc radius is equal to the nozzle exit radius of 0.3 caliber divided by the sine of the nozzle-exit half-angle of 20°). Downstream of the nozzle, the grid lines are aligned so as to closely approximate the exhaust plume shape for an experimentally observed axisymmetric flow by Agrell and White,[5] which is for the same geometric configuration and free-stream conditions, but for a jet-to-free-stream pressure ratio of 9. The third dimension, ς, is generated by rotating the two-dimensional (ξ, η) grid about the cylindrical axis while maintaining a uniform angular distribution between the rotated planes. Here, 20 radial planes are used with planes 2 and 19 coinciding with the bilateral plane of symmetry, where plane 2 corresponds to the lee and plane 19 to the windward. Planes 1 and 20 are image planes used to enforce a symmetry boundary condition. Thus, there are (ξ, η) planes distributed every 10.588° around the half-body.

The total grid dimensions are (140 x 100 x 20), corresponding to the ξ, η, and ς directions, respectively, for a total of 280,000 points. Of these, (80 x 40 x 20), or 64,000, lie inside the body and are not used in the computation, leaving an actual total of 216,000 points used in the computation.

Data Structure

There are 23 variables required at each grid point corresponding to the 5 conserved quantities in the Q vector, 5 residuals for the solution vector, 9 metric coefficients, the Jacobian of the transformation, and 3 components of vorticity used in the turbulence transport model. This results, for a computational grid of 216,000 points, in a data base of 5×10^6 words.

To accommodate this large data base on a vector processor with a limited main memory, the computational grid is divided into subsets called "blocks." This data structure was originally devised for implementation on the ILLIAC IV array processor by Lomax and Pulliam and is described in detail in Ref. 6. In the present case, each block is a 20 x 20 x 20 cube for a total of 8,000 points and a data base subset of 184,000 words for the 23 variables. The blocks are stacked together in each coordinate direction to form a sequence of blocks called "pencils."

For a given coordinate direction, one complete pencil of data is loaded into the central memory, and computations are performed on that data corresponding to the coordinate direction. At any point in the computation, only 17 variables are required to be in the main memory at one time (6 of the 9 metric coefficients are not used in any given direction). This results in a data-base subset of 136,000 words. For a processor with 10^6 words of main memory then, as many as seven blocks of data can be held in storage for immediate processing. The block dimension is an adjustable parameter and is limited only by the maximum pencil length and the main memory of the vector processor.

Shown in Fig. 2, in physical coordinates, are the block boundaries for the present configuration. Figure 2a shows the complete configuration and Fig. 2b the detail in the afterbody region. Figure 3 shows the corresponding block structure in computational space. The mesh nodes of the computational domain are arranged in a rectangular latice with positive integer coordinates (ξ, η, ς). Each node belongs to three pencils, a ξ-pencil, an η-pencil, and a ς-pencil. The pencils of each sweep direction are given a definite order. For the ξ-pencils, the η-coordinate varies most rapidly as the pencil index increases; for both the η-pencils and ς-pencils the coordinate ξ varies most rapidly. Figure 4 illustrates this sequencing for the present data structure.

Within a pencil, the planes are naturally ordered by the sweep coordinate. The pencils of data can be stored in the correct pencil ordering for just one sweep direction only. When sweeping in the other directions, pencils of data are gathered and fetched for computation and scattered back when writing the updated values. Additionally, the ordering of nodes within a plane can be correct for just one sweep direction, and it is necessary to transpose the the data in memory so that each plane of nodes normal to the sweep direction forms a contiguous set of memory locations. In

the present code, the ordering of nodes is correct for the ξ-direction and transpose routines are used for the other sweep directions.

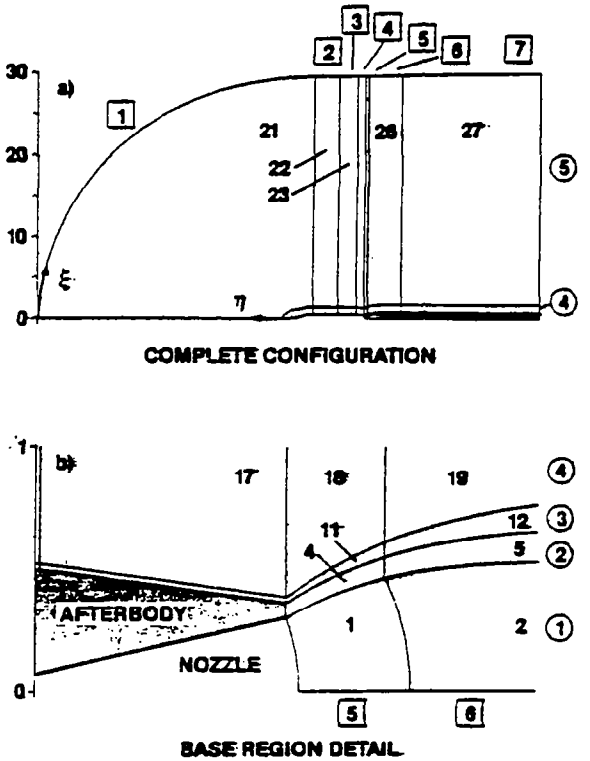


Fig. 2 Block boundaries: physical space. a) Complete configuration; b) Base-region detail.

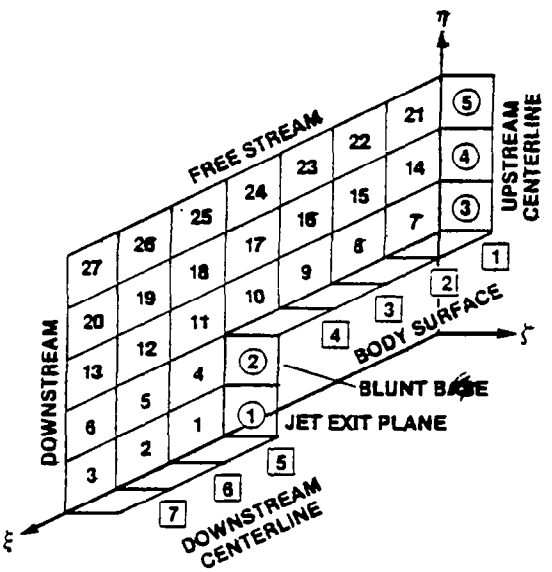


Fig. 3 Block boundaries: computational space, complete configuration.

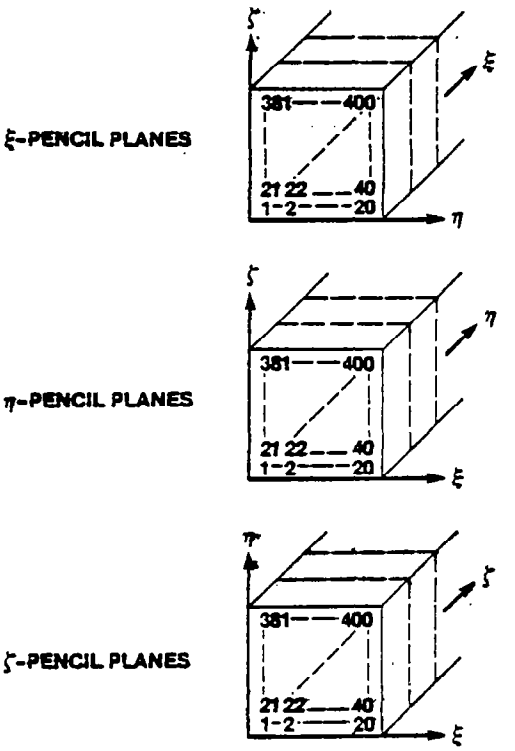


Fig. 4 Data structure within pencil data base.

Numerical Algorithm

The numerical algorithm used to solve Eq. (3) is the approximate factored scheme of Beam and Warming.[7] Rewriting Eq. (3) as

$$\partial_t Q = -\partial_\xi(F_H \cdot \vec{g}^\xi) - \partial_\eta(F \cdot \vec{g}^\eta) - \partial_\zeta(F_H \cdot \vec{g}^\zeta) = \mathcal{R} \quad (4)$$

the corresponding difference equation is then

$$L_\eta L_\zeta L_\xi \Delta_t Q = \mathcal{R}_\xi + \mathcal{R}_\eta + \mathcal{R}_\zeta \quad (5)$$

where the operators are defined by

$$L_\xi = (I + \Delta t\, \delta_\xi A^n - \epsilon_I J^{-1} \nabla_\xi \Delta_\xi J)$$
$$L_\eta = (I + \Delta t\, \delta_\eta C^n - \epsilon_I J^{-1} \nabla_\eta \Delta_\eta J - \Delta t \delta_\eta J^{-1} M^n J)$$
$$L_\zeta = (I + \Delta t\, \delta_\zeta B^n - \epsilon_I J^{-1} \nabla_\zeta \Delta_\zeta J)$$
$$\mathcal{R}_\xi = -\Delta t\, \delta_\xi (J F_H \cdot \vec{g}^\xi)^n - \epsilon_E J^{-1} (\nabla_\xi \Delta_\xi)^2 J Q^n$$
$$\mathcal{R}_\eta = -\Delta t\, \delta_\eta (J F \cdot \vec{g}^\eta)^n - \epsilon_E J^{-1} (\nabla_\eta \Delta_\eta)^2 J Q^n$$
$$\mathcal{R}_\zeta = -\Delta t\, \delta_\zeta (J F_H \cdot \vec{g}^\zeta)^n - \epsilon_E J^{-1} (\nabla_\zeta \Delta_\zeta)^2 J Q^n$$

where the δ_ξ, δ_η, and δ_ζ are central-difference operators; ∇_ξ, ∇_η, and ∇_ζ are backward-difference operators; and Δ_ξ, Δ_η, and Δ_ζ are forward-difference operators in the ξ-, η-, and ζ-directions, respectively. The Δ_t term is a forward-difference operator in time. For example,

$$\Delta_t Q = Q^{n+1} - Q^n$$

$$\Delta_\xi Q = Q(\xi + \Delta\xi, \eta, \zeta) - Q(\xi, \eta, \zeta)$$

and

$$\nabla_\xi Q = Q(\xi, \eta, \zeta) - Q(\xi - \Delta\xi, \eta, \zeta)$$

The Jacobian matrices

$$A = \partial_Q(F_H \cdot \vec{g}^\xi)$$

$$B = \partial_Q(F_H \cdot \vec{g}^\eta)$$

$$C = \partial_Q(F_H \cdot \vec{g}^\zeta)$$

$$M = \partial_Q(F_P \cdot \vec{g}^\eta)$$

are described in detail by Pulliam and Steger.[8] Fourth-order explicit terms (preceded by the coefficient ϵ_E) and second-order implicit terms (preceded by the coefficient ϵ_I) have been added to control nonlinear instabilities.

Equation (5) is solved in three successive sweeps of the data base, each sweep inverting one of the operators on the left-hand side:

$$\mathcal{L}_\zeta \mathcal{L}_\xi \Delta_t Q = \mathcal{L}_\eta^{-1}(\mathcal{R}_\xi + \mathcal{R}_\eta + \mathcal{R}_\zeta)$$

$$\mathcal{L}_\xi \Delta_t Q = \mathcal{L}_\zeta^{-1}\mathcal{L}_\eta^{-1}(\mathcal{R}_\xi + \mathcal{R}_\eta + \mathcal{R}_\zeta)$$

$$\Delta_t Q = \mathcal{L}_\xi^{-1}\mathcal{L}_\zeta^{-1}\mathcal{L}_\eta^{-1}(\mathcal{R}_\xi + \mathcal{R}_\eta + \mathcal{R}_\zeta)$$

The solution is advanced in time by adding $\Delta_t Q$ to Q after the ξ sweep.

In the general case, pencils of data are loaded into central memory four times and operated on for each time-step advance: once each for the ξ and η directions and twice for the ζ direction. First the right-hand side of Eq. (5) is formed and then the left-hand-side operators are inverted one by one. A flow schematic showing the ordering of operations, including data reads, transposes, computations, and data writes is shown below where the symbols R and ω represent variables used to accumulate the right-hand-side elements and vorticity elements, respectively, for each coordinate direction.

ξ-pencils: (initial step only)

Read:	Q, J, ξ-metrics
Compute:	$\mathrm{R} = \mathcal{R}_\xi$, $\omega = \omega(\xi)$
Write:	R, ω

Begin Loop

ζ-pencils:

Read:	Q, J, R, ω, ζ-metrics
Transpose:	Q, J, R, ω
Compute:	$\mathrm{R} = \mathcal{R}_\xi + \mathcal{R}_\zeta$, $\omega = \omega(\xi) + \omega(\zeta)$
Transpose:	R, ω
Write:	R, ω

η-pencils:

Read:	Q, J, R, ω, η-metrics
Transpose:	Q, J, R, ω
Compute:	$\omega = \omega(\xi) + \omega(\zeta) + \omega(\eta)$ $\mu_T(\omega)$ $\mathrm{R} = \mathcal{R}_\xi + \mathcal{R}_\zeta + \mathcal{R}_\eta$ $\mathcal{L}_\eta^{-1}(\mathrm{R})$
Transpose:	$\mathcal{L}_\eta^{-1}(\mathrm{R})$
Write:	$\mathcal{L}_\eta^{-1}(\mathrm{R})$

ζ-pencils:

Read:	Q, J, $\mathcal{L}_\eta^{-1}(\mathrm{R})$, ζ-metrics
Transpose:	Q, J, $\mathcal{L}_\eta^{-1}(\mathrm{R})$
Compute:	$\mathcal{L}_\zeta^{-1}\mathcal{L}_\eta^{-1}(\mathrm{R})$
Transpose:	$\mathcal{L}_\zeta^{-1}\mathcal{L}_\eta^{-1}(\mathrm{R})$
Write:	$\mathcal{L}_\zeta^{-1}\mathcal{L}_\eta^{-1}(\mathrm{R})$

ξ-pencils:

Read:	Q, J, $\mathcal{L}_\zeta^{-1}\mathcal{L}_\eta^{-1}(\mathrm{R})$, ξ-metrics
Compute:	$\Delta_t Q$, Q, $\mathrm{R} = \mathcal{R}_\xi$, $\omega = \omega(\xi)$
Write:	Q, R, ω

End Loop

In this flow sequence, 62 variables are read, 57 variables are transposed, and 31 variables are written. For the special case in the present study in which the ζ-pencils are just one block long, a more efficient operation sequence can be used that substantially reduces the number of reads and writes required. This is shown below.

ξ-pencils: (initial step only)

Read:	Q, J, ξ-metrics
Compute:	$\mathrm{R} = \mathcal{R}_\xi$, $\omega = \omega(\xi)$

Begin Loop

ζ-pencils:

Read:	ζ-metrics
Transpose:	Q, J, R, ω
Compute:	$\mathrm{R} = \mathcal{R}_\xi + \mathcal{R}_\zeta$
	$\omega = \omega(\xi) + \omega(\zeta)$
Transpose:	R, ω
Write:	R, ω

η-pencils:

Read:	Q, J, R, ω, η-metrics
Transpose:	Q, J, R, ω
Compute:	$\omega = \omega(\xi) + \omega(\zeta) + \omega(\eta)$
	$\mu_T(\omega)$
	$\mathrm{R} = \mathcal{R}_\xi + \mathcal{R}_\zeta + \mathcal{R}_\eta$
	$\mathcal{L}_\eta^{-1}(\mathrm{R})$

ζ-pencils:

Read:	ζ-metrics
Transpose:	Q, J, $\mathcal{L}_\eta^{-1}(\mathrm{R})$
Compute:	$\mathcal{L}_\zeta^{-1}\mathcal{L}_\eta^{-1}(\mathrm{R})$
Transpose:	$\mathcal{L}_\zeta^{-1}\mathcal{L}_\eta^{-1}(\mathrm{R})$
Write:	$\mathcal{L}_\zeta^{-1}\mathcal{L}_\eta^{-1}(\mathrm{R})$

ξ-pencils:

Read:	Q, J, $\mathcal{L}_\zeta^{-1}\mathcal{L}_\eta^{-1}(\mathrm{R})$, ξ-metrics
Compute:	$\Delta_t Q$, Q
	$\mathrm{R} = \mathcal{R}_\xi$, $\omega = \omega(\xi)$
Write:	Q,

End Loop

In this flow sequence, 32 variables are read, 52 are transposed, and 18 variables are written, a savings of nearly 50% in the I/O. In both the general case and the special case, the data read-transpose sequence and the transpose-write sequence can be replaced by the more efficient "gather" and "scatter" commands available for the Cyber 205 (Ref. 9). Further improvements in efficiency can be obtained by using asynchronous I/O in conjunction with a rotating memory backing store. The most efficient code, however, will be realized by using a solid-state backing store in conjunction with gather and scatter commands or with a code that is fully core contained.

The numerical algorithm conforms well to large vectorization. For block sizes of 20 x 20 x 20, the vector length is 400. Timing studies with the present code indicate an MFLOP rate (million of floating- point operations per second) of 115 when computing in half precision (32-bit word lengths) on a 2-pipe configuration. On a 4-pipe configuration the MFLOP rate increased to 207. There are approximately 3,800 floating point operations executed for every grid node per time step resulting in a CPU time of 33 x 10^{-6} sec per point per time-step on a 2-pipe machine and 18 x 10^{-6} sec per point per time-step on a 4-pipe machine. The transpose times (transposes do not contain any floating-point operations) are 5.6 x 10^{-6} sec per point. Equivalent transposes performed by gather and scatter instructions require just 1.8 x 10^{-6} sec per point. When synchronized I/O to and from rotating backing store was used, the average I/O time was 25 msec per variable per block. This translates directly into 172 x 10^{-6} sec per point, but overlapping the I/O reduces this to 94 x 10^{-6} sec per point. (The Cyber 205 used for these timing studies was configured with four I/O channels to accommodate overlapping.) This time, a result in large part of the latency time in accessing disk files, can be reduced to nearly zero by using I/O buffers in conjunction with asynchronous I/O or with solid-state backing storage. The use of I/O buffers, however, implies the availability of additional main memory and imposes an additional constraint on the pencil size. To avoid this constraint, the data flow should be modified such that a subset of contiguous blocks of data in a pencil are operated on while blocks at each end of the subset are being buffered in and out.

Boundary Conditions

Boundary conditions are imposed at the ends of each data pencil; the data pencils are identified by number in Fig. 3. For the ξ-direction, pencil No. 1 starts at the jet-exit plane. Supersonic conical flow conditions corresponding to a jet-exit Mach number of 2.5 and a static pressure of $3p_\infty$ are imposed at the first data plane. At the last plane of each of the five ξ-pencils, which correspond to the outflow boundary, first-order extrapolation is used so that $\partial_\xi Q = 0$. Pencil No. 2 in the ξ-direction begins at the blunt base. Here slip conditions and an impermeable adiabatic wall are imposed so that

$$\partial_\xi(\rho) = \partial_\xi(\rho v) = \partial_\xi(\rho w) = 0$$

$$\rho u = 0$$

$$\partial_\xi[e - 0.5(\rho u^2 + \rho v^2 + \rho w^2)] = 0$$

Pencils 3, 4, and 5 in the ξ-direction begin on the grid centerline of revolution (at $\xi = 0$) ahead of the forebody nose. Here a second order extrapolation to the centerline is used such that

$$\partial_\xi(\rho) = \partial_\xi(\rho u) = \partial_\xi(\rho w) = \partial_\xi(\rho e) = 0$$

while the lateral momentum is set to zero

$$\rho v = 0$$

In addition, at each η, the Q values are averaged over ζ on the centerline and used as boundary values for all ζ at each η. Special treatment of the base corner at the afterbody–blunt-base junction is used to account for the singular nature of that line. For the ξ-sweeps, the ζ-line of data in pencil No. 3 that corresponds to this corner is treated in the same manner as the first plane of data in pencil No. 2 that corresponds to the blunt base. This line of data is treated differently in the η-sweep and is described in the second paragraph following.

After the forebody flow field is fully developed during the course of the solution, the first two η-pencils can be dropped from the computation and boundary conditions imposed on the ξ-pencils that correspond to the fully developed flow at the plane that is the upstream boundary of η-pencil No. 3. This reduces the total data base by six blocks without altering the validity of the solution. This simplification is strictly valid only for supersonic external flows. The solution downstream can be further developed to steady state, and jet parameters can even be varied to generate additional solutions.

Boundary conditions for the η-direction consist of the imposition of free stream conditions at the last plane of each of the seven η-pencils; no-slip, adiabatic wall condition for the first plane of η-pencils 1 through 4, which correspond to the body surface; and first-order extrapolation to the centerline for pencils 5, 6, and 7 such that $\partial_\eta Q = 0$. Centerline averaging, as described for the ξ-pencil boundary ahead of the body, is also used for the η-pencil boundary in the jet. The line of data in η-pencil No. 5, which corresponds to the corner between the afterbody and the blunt base, is treated in the same manner as the first plane of η-pencils 1 through 4. As a result, this line of data is double valued: one value for the ξ sweep described previously and the no-slip, adiabatic value for the η-sweep.

For the ζ-direction, bilateral symmetry is imposed by setting the data at the first and last ζ-planes equal to the values in the third plane and in the second from last plane, respectively, with a sign change included in the lateral momentum component (ρv).

Turbulence Closure

The Reynolds stresses and turbulent heat-flux terms have been included in the stress tensor and heat-flux vector by using the eddy-viscosity and eddy-conductivity concept, whereby the coefficients of viscosity and thermal conductivity are the sum of the molecular (laminar) part and an eddy (turbulent) part. Eddy-viscosity models incorporate turbulent transport into the molecular-transport stress tensor by adding the scalar eddy-viscosity transport coefficient μ_T to the coefficient of molecular viscosity, ($\mu_e = \mu + \mu_T$), thereby relating turbulent transport directly to gradients of the mean-flow variables. In a Cartesian coordinate system, the three-dimensional molecular stress tensor can be written as

$$\begin{aligned}\tau_\ell = {} & (p+\sigma_x)\bar{\ell}_x\bar{\ell}_x + \tau_{xy}\bar{\ell}_x\bar{\ell}_y + \tau_{xz}\bar{\ell}_x\bar{\ell}_z \\ & \tau_{yx}\bar{\ell}_y\bar{\ell}_x + (p+\sigma_y)\bar{\ell}_y\bar{\ell}_y + \tau_{yz}\bar{\ell}_y\bar{\ell}_z \\ & \tau_{zx}\bar{\ell}_z\bar{\ell}_x + \tau_{zy}\bar{\ell}_z\bar{\ell}_y + (p+\sigma_z)\bar{\ell}_z\bar{\ell}_z\end{aligned}$$

In the thin-shear-layer approximation, the only components of the stress tensor that are retained are those having gradients with respect to η only.

Turbulent heat transport is defined in terms of mean-energy gradients and an eddy-conductivity coefficient K_e such that $K_e = K + K_T$. Typically, the eddy-conductivity coefficient is related to the eddy-viscosity coefficient via a turbulent Prandtl number Pr_T where

$$Pr_T = C_p \mu_T / K_T$$

The turbulent Prandtl number is assumed constant at a value of 0.9.

The algebraic eddy-viscosity model used here is that proposed by Baldwin and Lomax.[10] This model is particularly well suited to complex flows that contain regions in which the length scales are not clearly defined. It is described briefly as follows: For wall-bounded shear layers, a two-layer formulation is used such that

$$\mu_T = (\mu_T)_{inner} \quad \text{for} \quad \eta < \eta_{crossover}$$

$$\mu_T = (\mu_T)_{outer} \quad \text{for} \quad \eta > \eta_{crossover}$$

where η is the normal distance from the wall and $\eta_{crossover}$ is the smallest value of η at which values from the inner and outer formulas are equal. The Prandtl-Van Driest formulation is used in the inner (or wall) region.

$$(\mu_T)_{inner} = \rho \ell^2 |\omega|$$

$$\ell = 0.4\eta\,[1 - exp(-\eta/A)]$$

$$A = 26\mu_w / \sqrt{\rho_w \tau_w}$$

The formulation for the outer region is given by

$$(\mu_T)_{outer} = 0.0168\, C_{cp}\, F_{wake}\, F_{Kleb}(\eta)$$

$$F_{wake} = \min\begin{pmatrix} \eta_{max} F_{max}, \\ C_{wk}\, \eta_{max}\, q^2_{dif} / F_{max} \end{pmatrix}$$

The quantities η_{max} and F_{max} are determined from the function

$$F(\eta) = \eta\,|\omega|\,[1 - exp(-\eta/A)]$$

where F_{max} is the maximum value of $F(\eta)$, and η_{max} is the value of η at which it occurs. The function $F_{Kleb}(\eta)$

is the Klebanoff intermittency function given by

$$F_{Klob}(\eta) = [1 + 5.5(C_{Klob}\eta/\eta_{max})^6]^{-1}$$

The quantity q^2_{dif} is the difference between the maximum and minimum total squared velocity in the profile (along an η-coordinate line),

$$q^2_{dif} = q^2_{max} - q^2_{min}$$

and for boundary layers, the *minimum* is defined as zero. The other constants are given by

$$C_{ep} = 1.6 \quad , \quad C_{wk} = 0.25 \quad , \quad C_{Klob} = 0.3$$

The advantage of this model for boundary-layer flows are as follows: 1) for the inner region, the velocity and length scales are always well defined, and the model is consistent with the "law of the wall"; 2) in the outer region for well-behaved (simple) boundary layers, where there is a well-defined length scale (η_{max}), the velocity scale is determined by F_{max}, which is a length scale times a vorticity scale; 3) in the outer region of complex boundary layers where the length from a wall becomes meaningless, a new length scale is determined from a velocity (q_{dif}) divided by a velocity gradient ($|\omega|$), and the velocity scale is q_{dif}.

The outer formulation, which is independent of η, is also used in the free-shear flow regions of separated flow and in regions of strong viscous/inviscid interaction. In these regions the van Driest damping term, $[exp(-\eta/A)]$, is neglected. For jets and wakes, the Klebanoff intermittency factor is determined by measuring from the grid centerline, and the minimum term in q_{dif} is evaluated from the profile instead of being defined as zero.

The validity of the eddy-viscosity model constants for high-pressure, compressible exhaust jets has not been established, and compressibility effects are not accounted for.

At the exhaust-jet exit plane and in the near-base region, the eddy viscosity is assumed to be negligibly small and to increase spatially to the value given by the outer model over a short distance downstream of the base.

Computed Results

As mentioned in a preceding section (Afterbody Configuration), a flow field has been computed for the body placed at an angle of incidence of 6° to a free stream at Mach 2. The jet-exit Mach number is 2.5 with a static pressure 3 times that of the free stream. Beginning with an impulsive start in a uniformly flowing stream at Mach 2, the solution was advanced timewise to a dimensionless time ($t\,d/U_\infty$) of 5.1, where d is the forebody diameter and U_∞ is the undisturbed free-stream speed. Although a solution at a time of 5.1 is probably not sufficiently converged to permit valid quantitative comparisons with experiment, it is sufficient to establish the basic flow-field character and to illustrate the features of the solution and the computer code.

The initial time-step size of $\Delta t = 0.0001$ was increased to $\Delta t = 0.001$ as the solution passed through its initial rapid transient. A variable time-step was used in the subsonic flow regime downstream of the base in order to minimize the growth of nonlinear instabilities aggravated by changes in sign of the eigen-values in this region. The time-steps in this subsonic region were scaled down by a factor equal to the local streamwise Mach number with a cutoff minimum factor of 0.001 imposed to prevent the time-step from going to zero.

Occurring physically in this region is a rapid expansion of the jet around the nozzle lip followed immediately by a strong recompression in the form of a barrel shock; in addition there is a slip surface defining the boundary between the exhaust plume and the external flow. Each of these three high-gradient features is focused at the nozzle lip and demands a high degree of resolution that has not been provided for in the computational grid used here.

Shown in Fig. 5 are computed density contours in the bilateral plane of symmetry in the vicinity of the body. The lower surface is the wind side. Clearly defined downstream of the afterbody is the slip surface demarcating the boundary between the exhaust plume and the external flow. The propulsive jet expands rapidly around the nozzle lip and can induce flow separation on the afterbody surface. For low-pressure jets, or no jet at all, there will be a region of recirculating flow on the blunt base. The afterbody drag is strongly influenced by the detail of the separated flow.

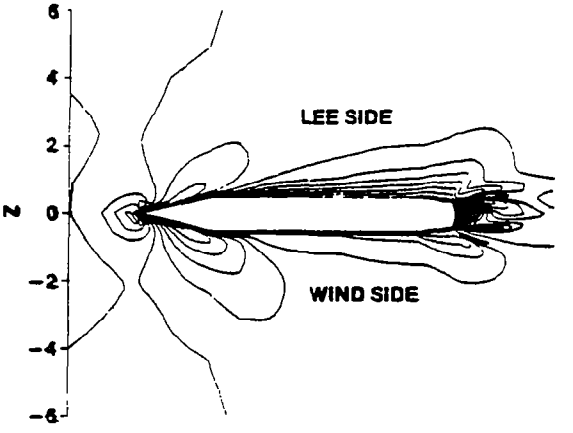


Fig. 5 Computed density contours, plane of symmetry: $M_\infty = 2$, $M_J = 2.5$, $P_J/P_\infty = 3$, $\alpha = 6°$, $Re_d = 1.5\text{x}10^6$.

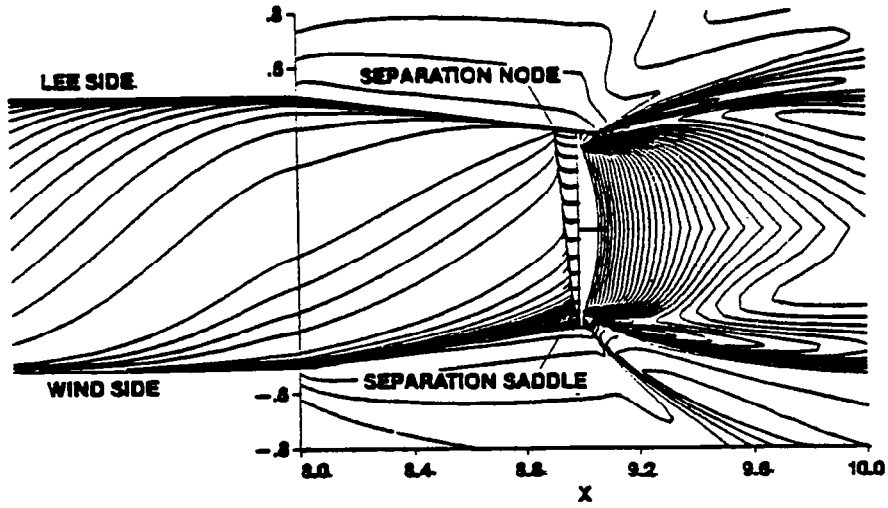


Fig. 6 Afterbody flow detail: surface streamlines and density contours on bilateral plane of symmetry.

The detail of the separation pattern is shown in Fig. 6 in which computed surface streamlines have been mapped on the afterbody and projected on the bilateral plane-of-symmetry view of the density contour plot over the aft portion of the body only. There is a separation node on the lee generator of the conical afterbody at $x = 8.92$. All surface streamlines on the lee side of the body flow into this node. A line of separation extends from this node, downward on the afterbody surface, to a separation saddle at $x = 8.98$, 33° from the wind generator. The flow direction along this line of separation is upward from the saddle to the node. There is also flow outward from the separation saddle downward to the end of the base, around to the wind generator.

Shown in Fig. 7 is a perspective view of the surface streamlines on the afterbody and the blunt base. The outer edge of the base is a dividing surface

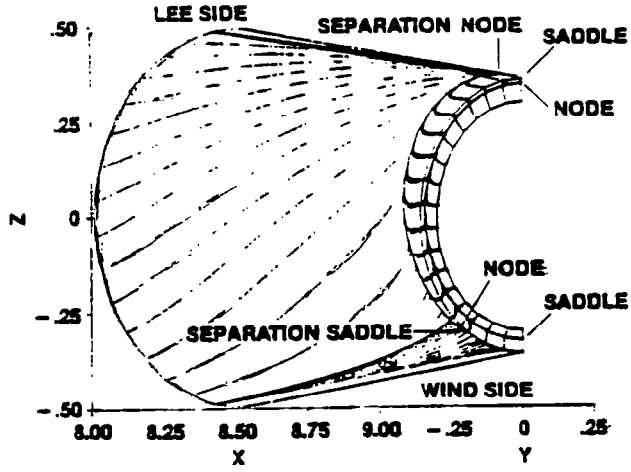


Fig. 7 Perspective view of surface streamlines over conical afterbody and annular base.

streamline extending from a saddle point on the lee generator to a node point approximately 33° from the wind generator. A dividing streamline can be seen circumscribing the annular base connecting a saddle point on the windward and a nodal point on the lee. This line separates the external flow from the flow from the jet. Flow is upward from the windward saddle to the lee-side node.

Shown in Fig. 8 is a sketch of an end-view projection of the full view of the afterbody (not to scale) showing all the dividing streamlines and their corresponding singular points and flow directions.

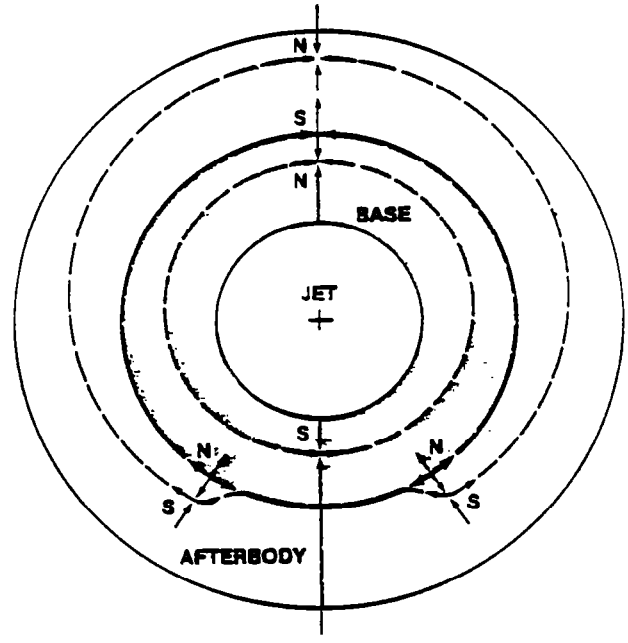


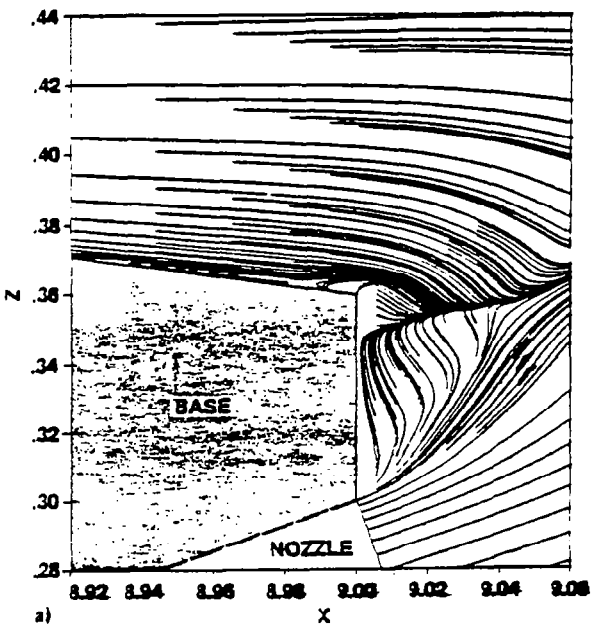
Fig. 8 End-view schematic of dividing surface and singular points streamlines.

The trajectories of the fluid particles in the plane of symmetry in the base region are shown in Fig. 9. On the lee, seen in Fig. 9a, the fluid from the jet expands around the nozzle lip and moves outward toward the edge of the base. Upon meeting the external flow, it turns downstream and defines the exhaust plume boundary. A region of reverse flow can be clearly seen above the afterbody lee generator. The path of the fluid in the external flow is over this separation region and around the afterbody base to the slip surface defining the boundary between the exhaust plume and external flow. The point defined by the outer edge of the base and the afterbody lee generator is a singular point that from the fluid streamlines, appears as a saddle point in both the circumferential plane and in the radial plane, and as a nodal point in the streamwise bilateral plane of symmetry (the plane of the base).

On the windward, shown in Fig. 9b, the streamlines just off the wind generator of the afterbody turn the corner and move toward the slip surface between the jet and the external flow. All external flow streamlines (excluding the surface streamline) approach the slip surface downstream of a saddle point in the bilateral plane of symmetry located at $x = 9.016$ on the plume-external flow boundary. The surface streamline turns the corner and approaches the windward saddle point on the base itself. Fluid from the jet expands around the nozzle lip and moves outward. The fluid just off the lip moves to the saddle point on the base and the fluid farther inside the lip expands toward the plume boundary downstream of the saddle point on the slip surface.

Surface-pressure distributions over the afterbody surface and over the base are shown in Figs. 10a and 10b, respectively. An expansion at the forebody-afterbody junction over the afterbody surface can be seen. This expansion is greatest on the windward, where the pressure level is highest, and decreases toward the lee. The circumferential variation of pressure near the lee side is quite small for the entire length of the afterbody. Toward the end of the afterbody there is a slight recompression on the lee side which is not observed on the windward. Just at the end of the afterbody there is an expansion as the flow turns around the afterbody toward the base.

Figure 10b shows a projected view of the base and jet-exit pressure distribution. The left side of the "top hat" pressure distribution corresponds to the lee, and the far side corresponds to the windward. The large uniform pressure distribution of the "top hat" configuration corresponds to the high-pressure jet, and the undulating "brim" of the hat is the distribution on the annular base. On the windward there is a rapid expansion at the nozzle lip followed by a fairly large recompression toward the outer edge of the base. The same trend is observed at other radial positions around the base but to a lesser degree. The circumferential variation of base pressure is consistent with the experimentally observed variation of White and Agrell for the same jet-to-free-stream pressure ratio. It is interesting to note, however, that in most experimental studies the radial variation of pressure is assumed negligible and is not measured. The distribution in Fig. 10b clearly indicates a substantial variation across the annular base.



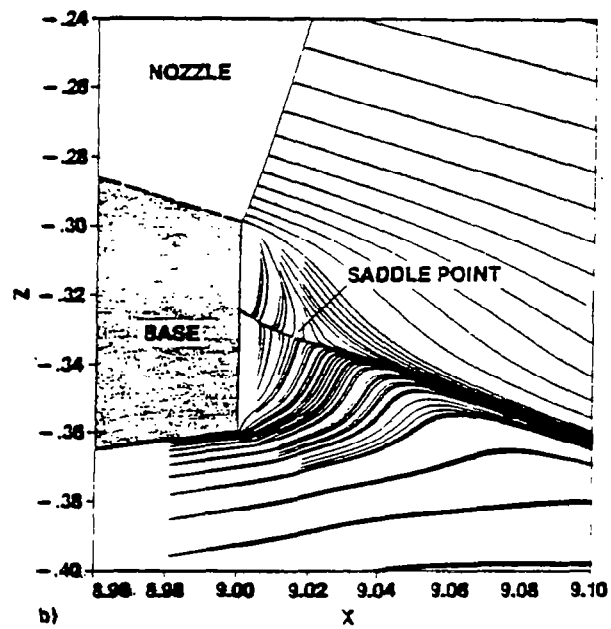


Fig. 9 Base-region path lines: plane of symmetry. a) Lee; b) Windward

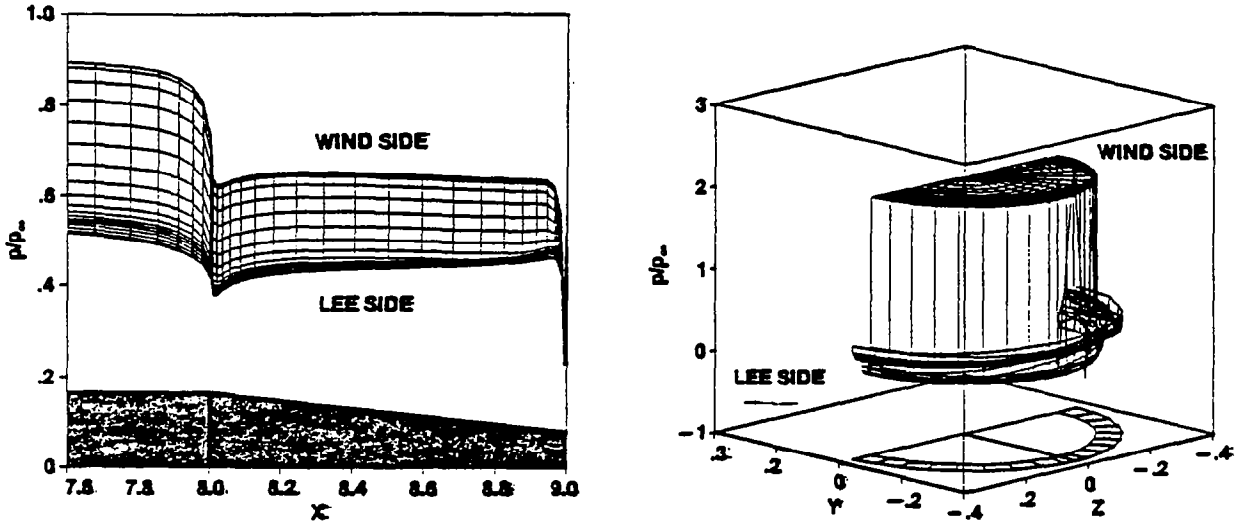


Fig. 10 Surface pressure distribution: perspective. a) Conical afterbody; b) Annular base and jet exit plane.

Concluding Remarks

An implicit solution procedure for the thin-layer approximation to the three-dimensional, time-dependent, compressible, Reynolds-averaged Navier-Stokes equations on a large array processor has been described. An example problem was simulated on the Cyber 205 computer that required a data base of 5×10^6 words. The efficient treatment of this large data base has been described in some detail.

The flow-field simulated was the supersonic flow over a body of revolution at incidence to the free stream. A propulsive jet emanated from the boattailed afterbody, inducing a complex, three-dimensional separated-flow pattern. This separated flow-field, which contributes substantially to the afterbody drag, has been described in detail for the particular geometry and flow conditions considered. The computed solution is consistent with experimental data observed for the same configuration and flow conditions.

References

[1]Deiwert, G. S., "Numerical Simulation of Three-Dimensional Boattail Afterbody Flowfields," AIAA Journal, Vol. 19, No 2, May, 1981, pp. 582-588.

[2]Deiwert, G. S., "A Computational Investigation of Supersonic Axisymmetric Flow over Boattails Containing a Centered Propulsive Jet," AIAA Paper 83-0462, 1983.

[3]White, R. A. and Agrell, J., "Boattail and Base Pressure Prediction Including Flow Separation for Afterbodies with a Centered Propulsive Jet and Supersonic External Flow at Small Angles of Attack," AIAA Paper 77-958, 1977.

[4]Vinokur, M., "On One-Dimensional Stretching Functions for Finite Difference Calculations," NASA CR-3313, 1980.

[5]Agrell, J. and White, R. A., "An Experimental Investigation of Supersonic Axisymmetric Flow over Boattails Containing a Centered Propulsive Jet," FFA Tech. Note AU-913, 1974.

[6]Lomax, H. and Pulliam, T. H., "A Fully Implicit Factored Code for Computing Three-Dimensional Flows on the ILLIAC IV," Parallel Computations, G. Rodrigue, Ed., Academic Press, New York, 1982, pp. 217-250.

[7]Beam, R. and Warming, R. F., "An Implicit Finite-Difference Algorithm for Hyperbolic Systems in Conservation-Law-Form," Journal of Computational Physics, Vol. 22, Sept. 1976, pp. 87-110.

[8]Pulliam, T. H. and Steger, J. L., "Implicit Finite-Difference Simulations of Three-Dimensional Compressible Flow," AIAA Journal, Vol. 18, No. 2, Feb. 1980, pp. 159-169.

[9]CDC Cyber 200 FORTRAN Version 2 Reference Manual, Control Data Corporation, Sunnyvale, Calif., 1981.

[10]Baldwin, B. S. and Lomax, H., "Thin Layer Approximation and Algebraic Model for Separated Turbulent Flows," AIAA Paper 78-257, 1978.

STEADY VISCOUS FLOW PAST A CIRCULAR CYLINDER

BENGT FORNBERG

DEPARTMENT OF APPLIED MATHEMATICS
CALIFORNIA INSTITUTE OF TECHNOLOGY

PASADENA, CALIFORNIA

STEADY VISCOUS FLOW PAST A CIRCULAR CYLINDER

Bengt Fornberg
Department of Applied Mathematics 217-50
California Institute of Technology
Pasadena, Ca 91125.

ABSTRACT

Viscous flow past a circular cylinder becomes unstable around Reynolds number Re = 40. With a numerical technique based on Newton's method and made possible by the use of a supercomputer, steady (but unstable) solutions have been calculated up to Re = 400. It is found that the wake continues to grow in length approximately linearly with Re. However, in conflict with available asymptotic predictions, the width starts to increase very rapidly around Re = 300. All numerical calculations have been performed on the CDC Cyber 205 at the CDC Service Center in Arden Hills, Minnesota.

INTRODUCTION

The structure of viscous steady flow past a circular cylinder at high Reynolds numbers forms one of the classical problems in fluid mechanics. In spite of much attention, several fundamental questions remain open. Apart from a previous calculation by the present author [6], complete, steady flow fields have been obtained numerically only up to around Re = 100. This is also close to the upper limit for experiments (due to temporal instabilities). Both the early numerics and the experiments point to a recirculation region growing linearly in length with Re. Figure 1 shows the length of the wake bubble against Reynolds number according to some different calculations. Persistence of this growth for Re -> ∞ has been assumed in most

recent asymptotic studies of steady high Reynolds number flows past a body (e.g. F.T. Smith [13]). A possible Euler flow, consistent with this idea, was analyzed by Brodetsky [3] in 1923. It is known as the Helmholtz- Kirchhoff free streamline model. This suggested limit is characterized by two vortex sheets leaving the body tangentially approximately 55° from the upwind center line and extending to downstream infinity, enclosing a region of stagnant flow. Although this undoubtedly is a solution for Re = ∞, G.K. Batchelor [2] gave in 1956 several arguments against this being a possible limit for Re -> ∞. He proposed an alternative in which a finite wake with piecewise constant vorticity was bounded by vortex sheets. Some suggestions about how such a flow might be reached as a limit for increasing Reynolds number have been given by Peregrine [10]. However, only very few Euler solutions of this so called Prandtl-Batchelor type have been calculated (e.g.[12] contains one example and some further references). None of these are for flow past a cylinder. Figure 2 gives an 'artists impression' of what the two models for infinite Re might look like. The calculation [6] hinted at a process leading to a shortening of the wake. The present work suggests (in agreement with F.T. Smith [14]) this shortening at Re = 300 was erroneous and caused by insufficient numerical resolution. However, our best current evidence is that the qualitative result was correct. We beleive that a reversal of trends towards a shorter wake can be expected around Re = 500. This contrasts with the conclusions in [14]. Our main evidence is that the wake increases in width far more rapidly after Re = 300 than the asymtotic analysis allows for. Independently of the position of artificial boundaries and of numerical resolution, we find that the flow is of different character past Re = 300. Significant amounts of vorticity are then re-circulated back into the wake bubble from its end. We hope to soon carry this study past Re = 400.

All the numerical calculations in this present work were performed on the Control Data Corporation Cyber 205 computer located at the CDC Service Center in Arden Hills, Minnesota. We wish to express our gratitude to Control Data Corporation for making this system available for this work.

MATHEMATICAL FORMULATION.

With a cylinder of radius 1 and a Reynolds number based on the diameter, the governing time independent Navier-Stokes equations, expressed in streamfunction Ψ and vorticity ω, take the form:

$$(1) \quad \Delta\Psi + \omega = 0$$

$$(2) \quad \Delta\omega + \frac{Re}{2}\left\{\frac{\partial\Psi}{\partial x}\cdot\frac{\partial\omega}{\partial y} - \frac{\partial\Psi}{\partial y}\cdot\frac{\partial\omega}{\partial x}\right\} = 0$$

Accurate numerical approximation and economical computational solution of these equations in the given geometry poses a series of difficulties which previous investigators have dealt with in a variety of ways. The most serious of the difficulties seem to be:

1. Boundary comditions for Ψ at large distances.
2. Boundary condition for ω at the body surface.
3. Avoiding the loss of accuracy that comes with upwind differencing.
4. Economical choice of computational grid.
5. Reliable and fast rate of convergence of numerical iterations.

The point 5 above has been the limiting factor in virtually all previous attempts to reach high Reynolds numbers. No reliable technique has emerged to prevent slowly converging iteration schemes from picking up physical instabilities in the artificial time of the iterations.

NUMERICAL METHOD

All vorticity is concentrated on the body surface and in a quite thin wake downstream of the body. Outside this region we can use the much simpler equations:

$$\Delta\Psi = 0 \tag{3}$$

$$\omega = 0 \tag{4}$$

The top part of Figure 3 shows the upper half plane minus a unit circle and, dotted, a region which contains all the vorticity (apart from the far wake). The bottom part of the figure shows how the mapping $z = \sqrt{x} + 1/\sqrt{x}$ maps these to the first quadrant and a rectangle respectively. Figure 4 shows what a rectangular grid in the z-plane (with non-uniform stretching in the vertical direction) can look like in the x-plane. The Navier-Stokes equations, transformed to the z-plane take a form almost identical to (1) and (2):

$$\Delta\Psi + \omega/J = 0 \tag{5}$$

$$\Delta\omega + \frac{Re}{2}\left\{\frac{\partial\Psi}{\partial\xi}\cdot\frac{\partial\omega}{\partial\eta} - \frac{\partial\Psi}{\partial\eta}\cdot\frac{\partial\omega}{\partial\xi}\right\} = 0 \tag{6}$$

where $J = \left|\frac{dz}{dx}\right|^2$ is the Jacobian of the mapping. These equations were modified further by subtracting out potential flow. The stream function for the difference is $\Psi = \Psi - 2\xi\eta$. On a grid in the (stretched) z-plane, equations (5) and (6) were approximated at all interior points with centered second order finite differences. To close the system, boundary conditions have to be implemented for Ψ and ω at all boundaries.

The extreme sensitivity of the final solution to small errors in these conditions has only recently been fully recognized [6]. For example already at Re = 2 it was found that use of the free stram value for Ψ along circular outer boundaries at distances 23.1 and 91.5 caused 18 % and 4.4 % errors in the level of vorticity on the body surface.

The 'Oseen' approximation is the leading term in an asymptotic expansion for the flow far out in a wake (e.g. Imai [8]). In polar coordinates, it takes the form

$$(7) \qquad \Psi = \frac{C_D}{2} \left(\frac{\theta}{\pi} - \operatorname{erf} Q \right)$$

$$(8) \qquad \omega = - \frac{C_D \mathrm{Re}\, Q}{4 \sqrt{\pi}\, r} e^{-Q^2}$$

where $Q = (\frac{1}{2} \mathrm{Re}\, r)^{1/2} \sin \frac{1}{2}\theta$, $\operatorname{erf} Q = 2\pi^{-1/2} \int_0^Q e^{-s^2} ds$ and C_D the drag coefficient. C_D can be evaluated as a line integral around the body.

The performance of this Oseen condition as an outer boundary condition is disappointing. The percentage errors mentioned above improve, but only to to 3.4 % and 1.2 % respectively. For increasing Re, direct use of (7) becomes meaningless. Figure 5 illustrates this by comparing the true Ψ (here the difference between streamfunction and free stream, not potential flow) with the values from (7) at Re = 200. The two fields bear no resemblance to each other at the distances from the body we are interested in.

Comparison with numerics suggest that (8) is far more accurate than (7). Furthermore

1. Any errors in (8) are present only in a very narrow region along the outflow axis, not along the whole upper boundary as with (7).

2. The governing equation for ω is of a type which cannot transport incorrect information for ω back up towards the cylinder.

With this background, let us briefly outline how the boundary conditions of high accuracy can be implemented on the edges of the present computational region. Figure 6 shows this region in the z-plane with a typical vorticity field together with its reflections in the coordinate axis.

BOUNDARY CONDITIONS FOR ψ.

Left boundary: $\xi = 0$, $0 \le \eta \le \eta_N$. $\psi = 0$.

Bottom boundary: $\eta = 0$, $0 \le \xi \le \xi_M$. $\psi = 0$.

Right boundary: $\xi = \xi_M$, $0 \le \eta \le \eta_N$. $\frac{\partial^2 \psi}{\partial \eta^2} = \omega$ (noting that $\frac{\partial^2 \psi}{\partial \xi^2} << << \frac{\partial^2 \psi}{\partial \eta^2}$ along this boundary).

Top boundary: $\eta = \eta_N$, $0 \le \xi \le \xi_M$. $\psi_{\xi,\eta_N} = \int_{-\xi_M}^{\xi_M} \int_{-\eta_N}^{\eta_N} \omega \ln\left((\xi_i - \xi)^2 + (\eta_N - \eta)^2\right)/J \, d\xi d\eta$

A correction to the integral above for vorticity reaching outside the downstream boundary can easily be incorporated. For a fixed grid, the dependence of ψ at each boundary point on ω at each internal point is independent of Re and can be calculated as a large matrix once and for all. A boundary condition of this kind was used in all the calculations presented below. However, we currently use a different condition. A wide two level difference formula can be found which is consistent only with the decaying modes of the equation $\Delta \psi = 0$ (as opposed to the usual 5-point 3-level formula used inside the region to approximate both growing and decaying modes).

BOUNDARY CONDITIONS FOR ω.

Left boundary: $\xi = 0$, $0 \le \eta \le \eta_N$. $\omega = 0$.

Bottom boundary: $\eta = 0$, $0 \le \xi < 2$. A relation based on $\Delta \psi + \omega/\eta = 0$ and ψ an even function of η.

$2 \le \xi \le \xi_M$. $\omega = 0$.

Right boundary: $\xi = \xi_M$, $0 \le \eta \le \eta_N$. $\omega_{\xi_M} = \omega_{\xi_{M-1}} \left(\xi_M / \xi_{M-1}\right)^2$

Top boundary: $\eta = \eta_N$, $0 \le \xi \le \xi_M$. $\omega = 0$.

The condition at the right boundary comes from the observation that the leading term of (8), transformed to ξ, η -coordinates simplifies to

$$\omega = \frac{c_1}{\xi^2} \eta \, e^{-c_2 \eta} \tag{9}$$

where c_1 and c_2 are constants. The mapping has achieved a separation of variables.

The discrete approximations at the interior points together with the boundary conditions form, after minor simplifications (explicitly eliminating all boundary unknowns apart from Ψ at the top boundary), a non-linear algebraic system of (M-2)(2N-3) equations with equally many unknowns. In most earlier works, great care has been taken to ensure that, at this stage, this (or some equivalent) non-linear system has a diagonally dominant form for low Re. This would allow direct functional iteration to convergence. Techniques like upwind differencing [1],[4],[11] help in this respect at the cost of lowered accuracy. Newton's method, described below, offers an outstanding alternative.

NEWTON'S METHOD.

Newton's method is a very well known procedure for finding zeros of scalar functions. If a function f(x) is given, we can find an x such that f(x)=0 by the procedure:

(10) x_0 'close' guess of root

(11) $$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \qquad n = 0, 1, 2, \ldots$$

The iteration step can be written

(12) $$f'(x_n)\ \Delta x_n = -f(x_n)$$

Known, f' evaluated at the latest available approximation x_n.	Unknown, the correction we should apply to x_n, i.e. $x_{n+1} = x_n + \Delta x_n$.	Known, residual. Should be zero if x_n had been exact.

Written in this form, the generalization to systems is straightforward. For example the system with three equations in three unknowns:

$$
\begin{aligned}
f(x, y, z) &= 0 \\
g(x, y, z) &= 0 \\
h(x, y, z) &= 0
\end{aligned}
\tag{13}
$$

can be iterated

$$
\begin{bmatrix}
\frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} & \frac{\partial f}{\partial z} \\
\frac{\partial g}{\partial x} & \frac{\partial g}{\partial y} & \frac{\partial g}{\partial z} \\
\frac{\partial h}{\partial x} & \frac{\partial h}{\partial y} & \frac{\partial h}{\partial z}
\end{bmatrix}
\begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{bmatrix}
= -
\begin{bmatrix} f(x, y, z) \\ g(x, y, z) \\ h(x, y, z) \end{bmatrix}
$$

Known, "Jacobian" of system. Unknown, corrections. Known, residual.

Each iteration involves the solution of a linear system. Like in the scalar case, convergence is quadratic and guaranteed to occur for approximations sufficiently close to any 'simple' solution. The realization that this procedure is practical for extremely large systems (several thousands of equations) is rather recent and linked to the emergence of powerful computers.

For our present problem, use of Newton's method offers several major advantages:

1. The quadratic convergence allows no possibility of 'inheriting' temporal instabilities to the artificial time of the iterations. Convergence is guaranteed if an isolated solution exists in the neighborhood of a guess.

2. If turning points or bifurcation points are found, they will cause no difficulties.

3. No upwind differencing is needed. This procedure is typically employed for two reasons:

1. To ensure convergence of an iterative method.

2. To avoid mesh size oscillations.

The first reason no longer applies. The second one alone can then be addressed in more refined ways.

4. Boundary conditions at the body surface become easier to implement. The fact that we have two conditions on ψ and none on ω can cause a problem if (5) and (6) are treated separately. With Newton's method, all we need is that the number of conditions is right.

The only disadvantage with Newton's method is the computational cost. This is where supercomputers enters our picture.

SOLUTION OF LINEAR SYSTEM

Let $[\psi]_j$, j=2,3,...,N be vectors with ψ -values from grid lines 2,3,...,N and similarly for ω_j (j=2,...,N-1). For example ψ_2 would contain the ψ -values along the grid row nearest to the ξ -axis and ψ_N the values along the top boundary. The structure of the entries in the Jacobian matrix reflects directly on the difference stencils and the boundary conditions. Figure 7 shows a suitable ordering of equations and unknowns and the corresponding structure of the Jacobian. Since the top right corner contains a single diagonal, explicit multiples of the top (N-2)(M-2) equations can be superposed on the equations below to modify the structure to the one in Figure 8. The bottom left corner form a separated system of size (N-1)(M-2). This system was solved by a border algorithm similar to the one described in [9]. The major cost comes from the LU-factorization of A. However, one more rearrangement can be done to achieve a significant speedup. The A-matrix has a block 5-diagonal form with the structure shown in Figure 8. A similarity transform with a permutation matrix can rearrange this into another matrix of identical structure. Instead of N-2 rows of blocks, each of size M-2, we get M-2 rows of blocks of size N-2. With M typically around 6*N and cost proportional to the square of the bandwidth, this reduces the memory needed for the LU-decomposition about 6 and the operation count by 36.

The complete linear solver lends itself ideally to vectorization. Every part of significant cost turns out take a form of a 'linked triad' with vectors never shorter than 4(N-2)+1 or M. The linked triad on the Cyber 205 is the fastest floating point

operation the machine offers. Expressions of the form *vector-op-vector-op-scalar where one 'op'* is + or -, the other * can execute with both operations running simultaneously. On the 2-pipe 205, the algorithm has a potential for 200 mflops (million floating point operations per second, 64-bit accuracy). Including a startup cost of 83 macihne cycles per linked triad operation, average vector length of around 166 (which we will exceed in later test cases) could give a full 100 mflop overall computational performance. In the calculations presented below, the grid had 131 by 21 points. Building up the Jacobian (in scalar mode) takes 2.3 seconds and the solution of the linear system 3.7 seconds (for an average of 55 mflops during this part). Recently implemented vectorization of the Jacobian and the new boundary condition brings these numbers to 0.026 seconds, 1.75 seconds and 60 mflops respectively.

PHYSICAL CONCLUSIONS

This report is a preliminary one of work in progress. Only a few initial test runs have been performed so far. However, we can already conclude that the wake appears to continue a linear growth in length with increasing Reynolds numbers up to Re = 400. Figure 9 shows wake length versus Re for some previous calculations compared with current results. Figure 10 shows streamlines and Figure 11 vorticity fields for different values of Re up to 400. The vorticity field at Re = 400 shows a recirculation back into the wake from the end of the bubble as well as a quite sudden increase in width. Our most recent tests with a computational grid of 196 by 31 points (density increased by 3/2 in each direction) leaves these features completely unchanged. The onset occurs near Re = 300 and the widening progresses at a rate which can be determined accurately and which far exceeds the one predicted by available asymptotic models.

The flow fields in figures 10 and 11 were obtained from a 131*21 grid in the z-plane with

(14) $\xi_i = i/12$, $i=0,1, \ldots ,130$

(15) $\eta_j = \alpha\zeta_j^3 + (1-4\alpha)\zeta_j$, $\zeta_j = j/18$, $j=0,1, \ldots ,20$, $\alpha=0.15$

This places the right boundary at a distance 115.4 from the center of the cylinder. Preliminary tests involving moving this and the top boundaries in and out suggest that they are sufficiently far out with the present choice of grid. Figure 4 showed part of this grid.

The major open questions at the moment are:

Physically:

1. Will the wake keep on growing?

2. Are there any other branches of solutions (bifurcations etc.)?

Numerically:

1. Is there any alternative to Newton's method which still possesses a reliable rate of convergence?

2. Is there any faster way than Gaussian elimination to solve the linear system in Newton's method?

At present, the numerical questions are wide open and of fundamental importance to many other applications as well. Current numerical methods together with vector computers like the Cyber 205 probably form sufficiently powerful tools to settle conclusively the physical questions raised here.

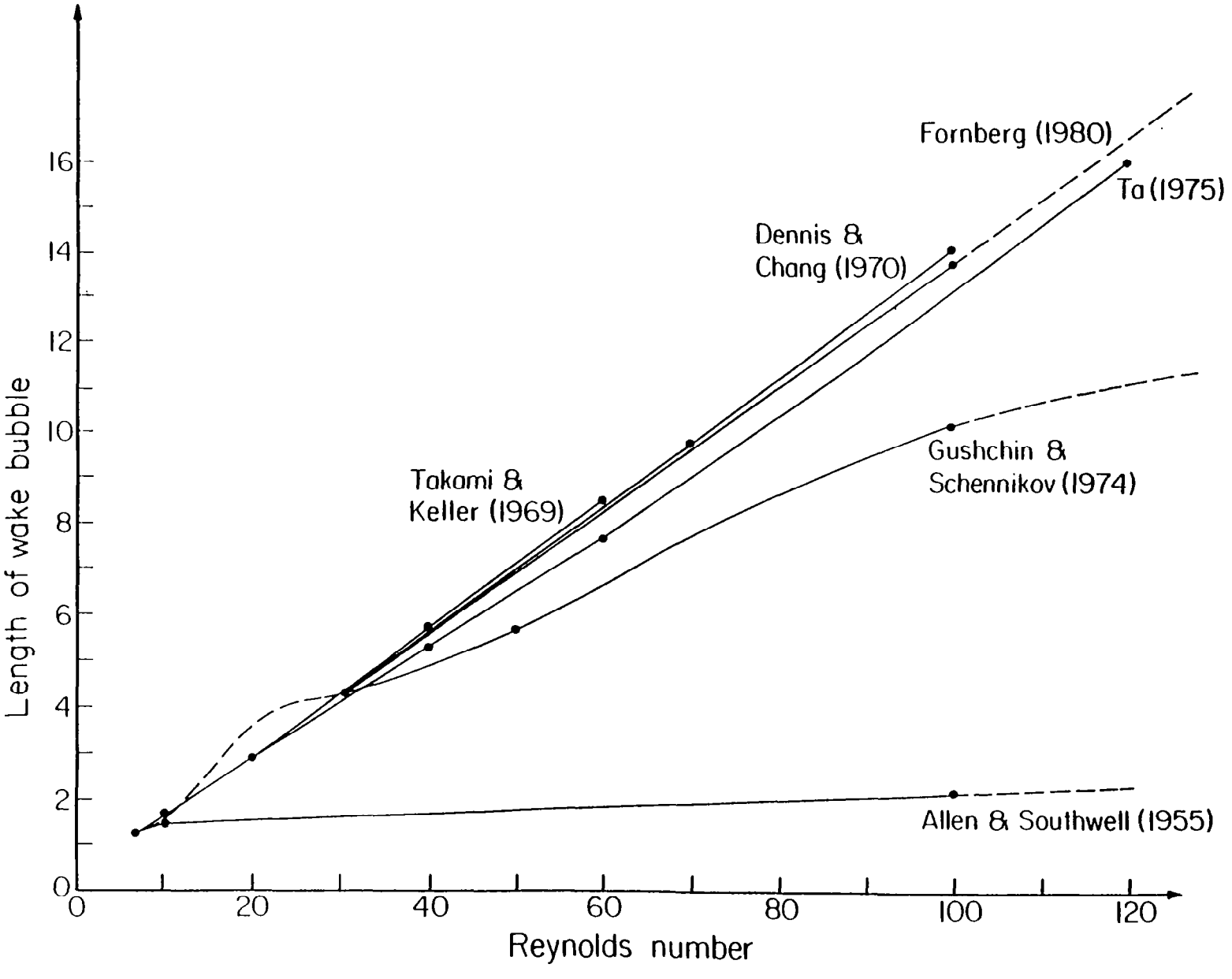


Figure 1. Length of wake bubble for low Reynolds numbers according to some different calculations.

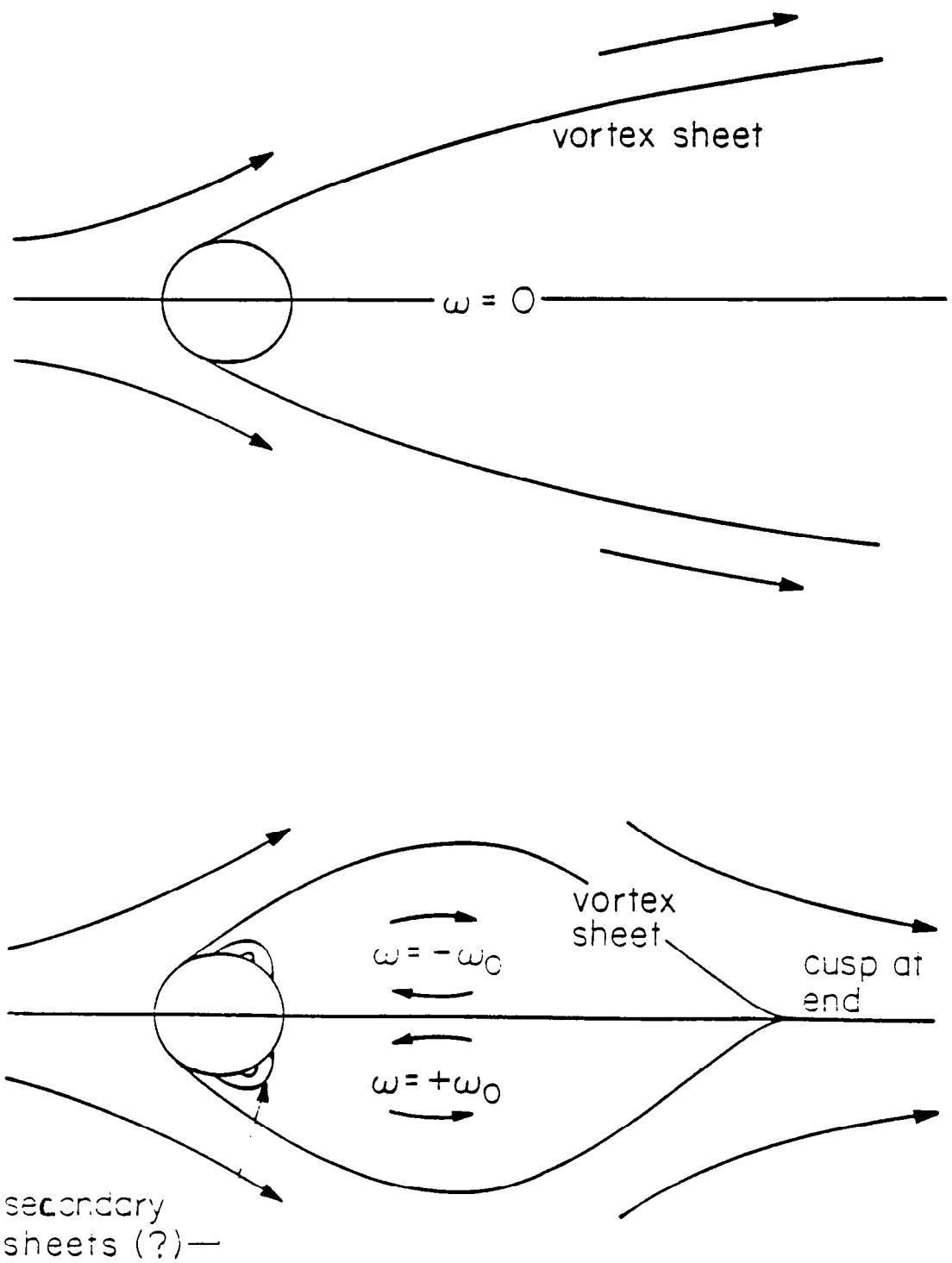


Figure 2. Schematic illustration of free streamline and the Prandtl-Batchelor models.

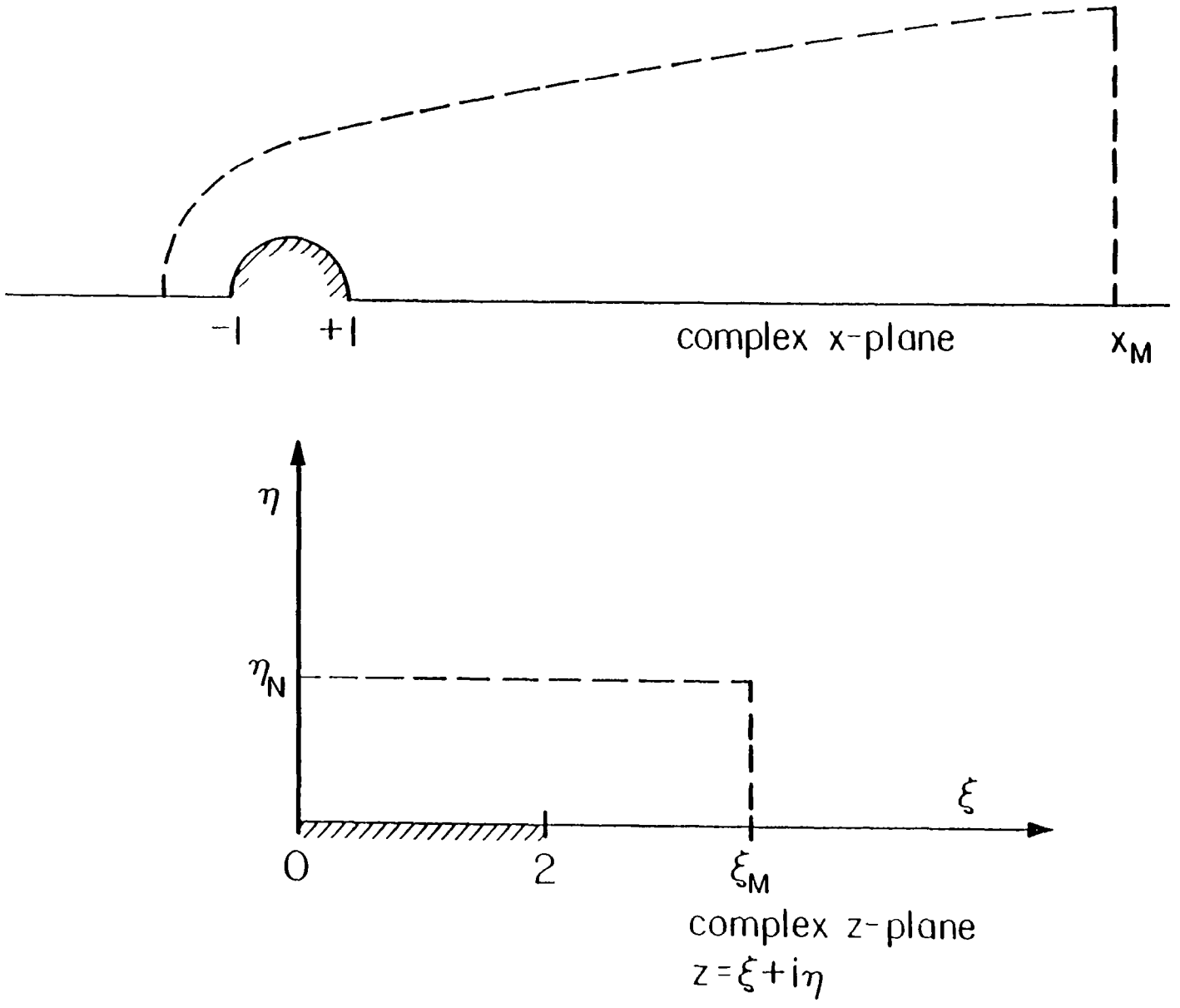


Figure 3. Conformal mapping of computational region.

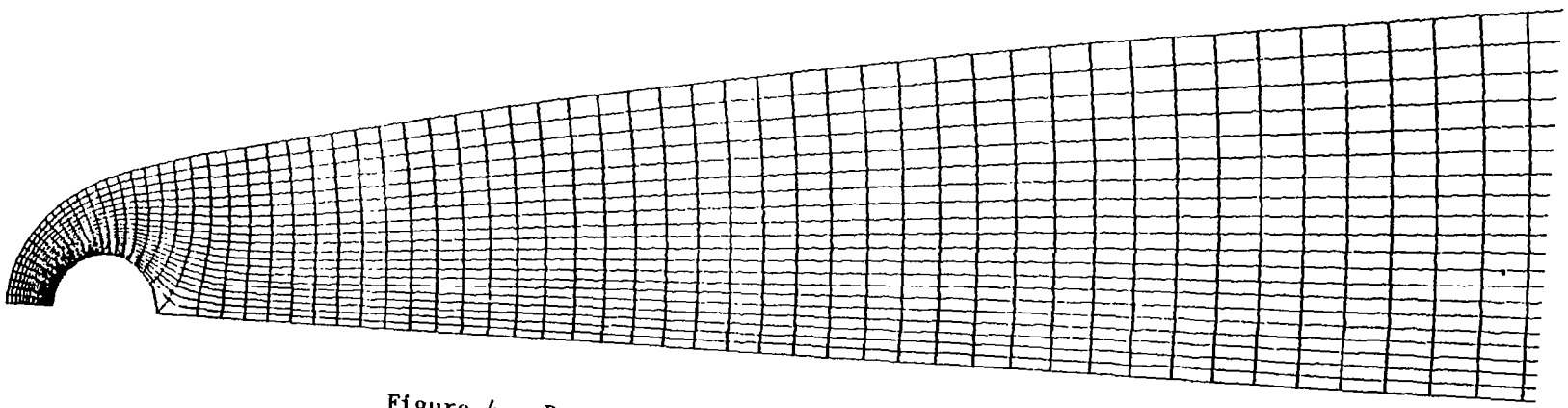


Figure 4. Part near cylinder of computational grid.

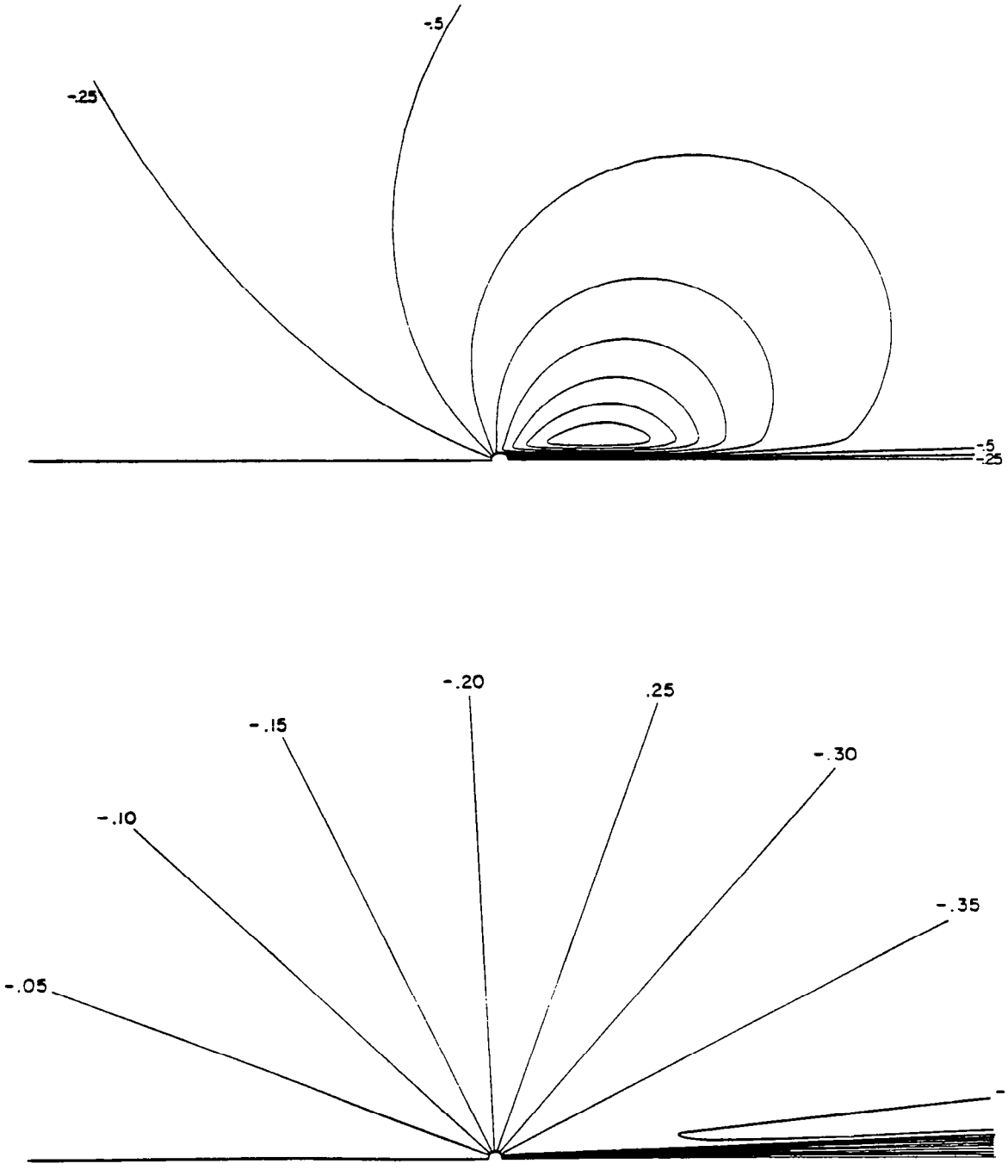


Figure 5. True difference between streamfunction and free stream compared with Oseen approximation for Re = 200.

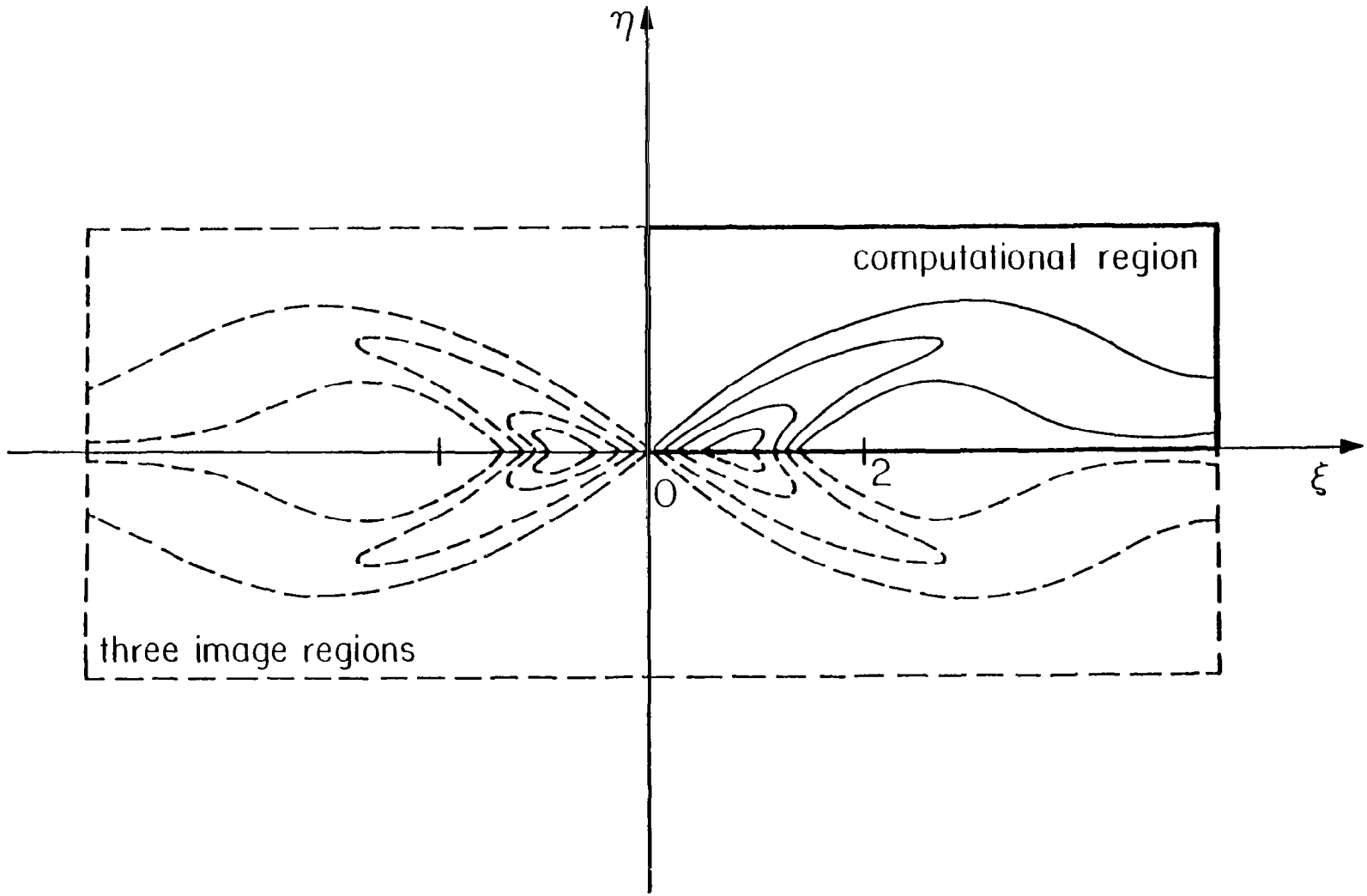


Figure 6. Typical vorticity field in complex z-plane.

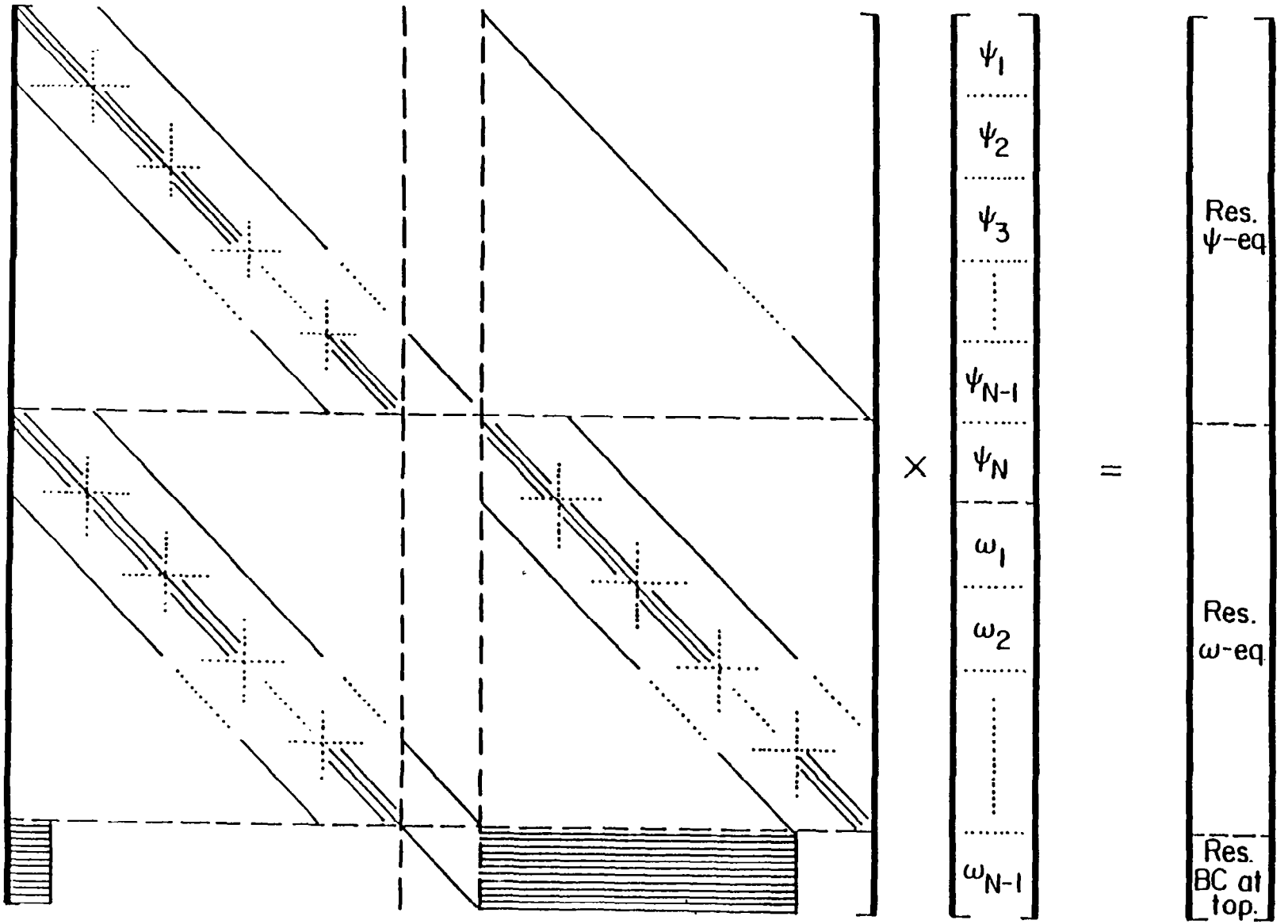


Figure 7. Linear system in Newton's method.

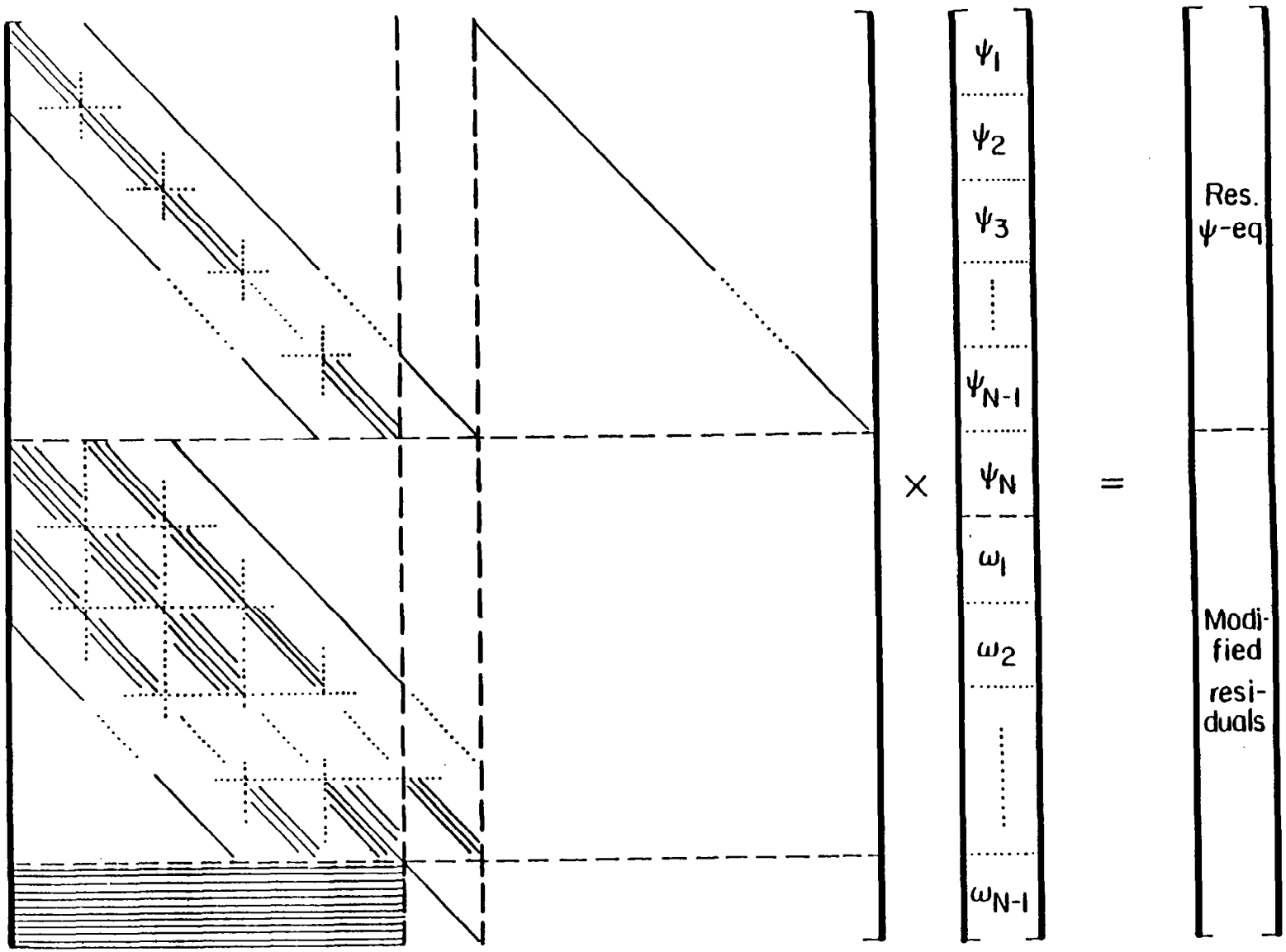


Figure 8. Modified linear system.

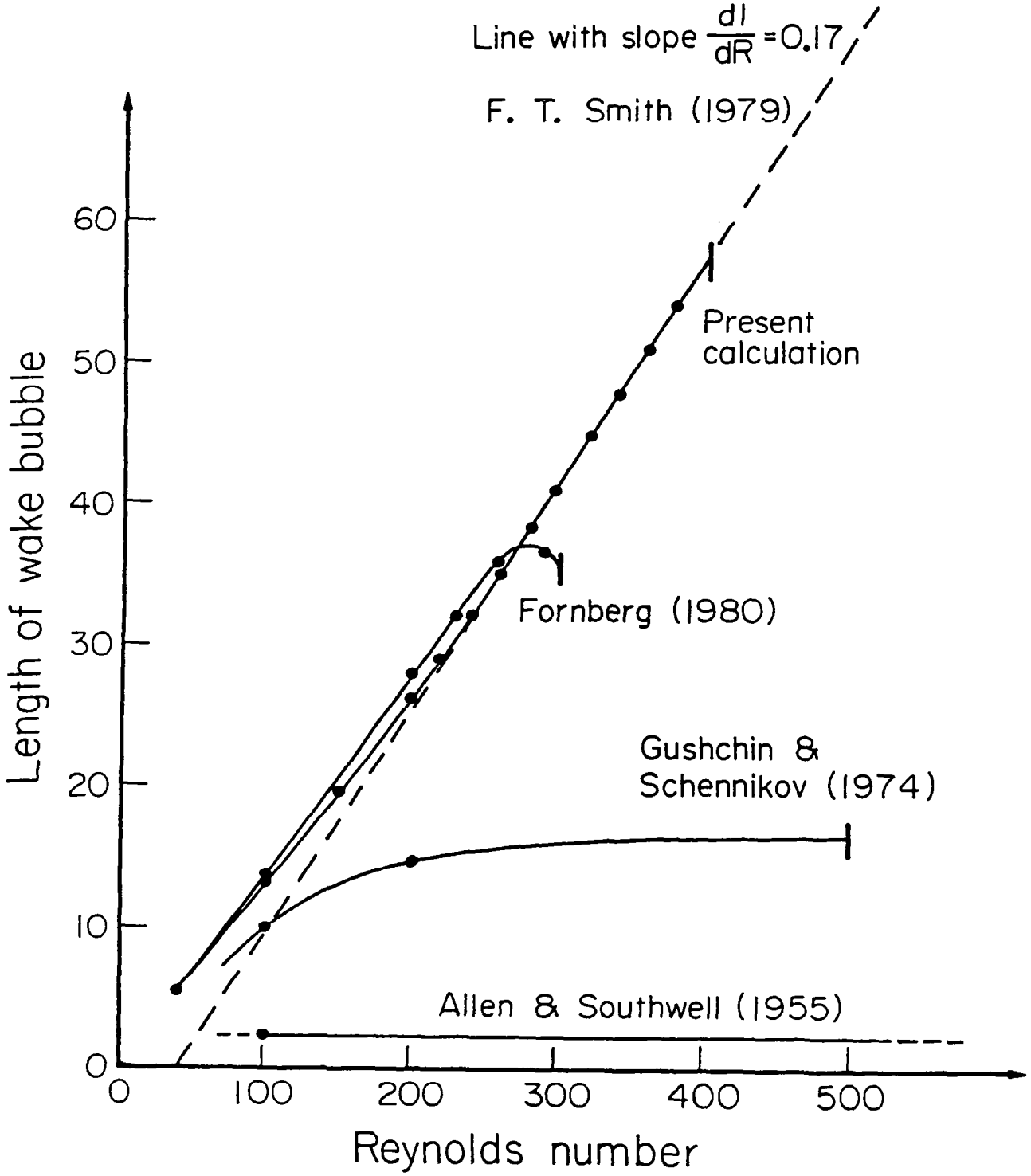


Figure 9: Length of wake bubble for different Reynolds numbers.

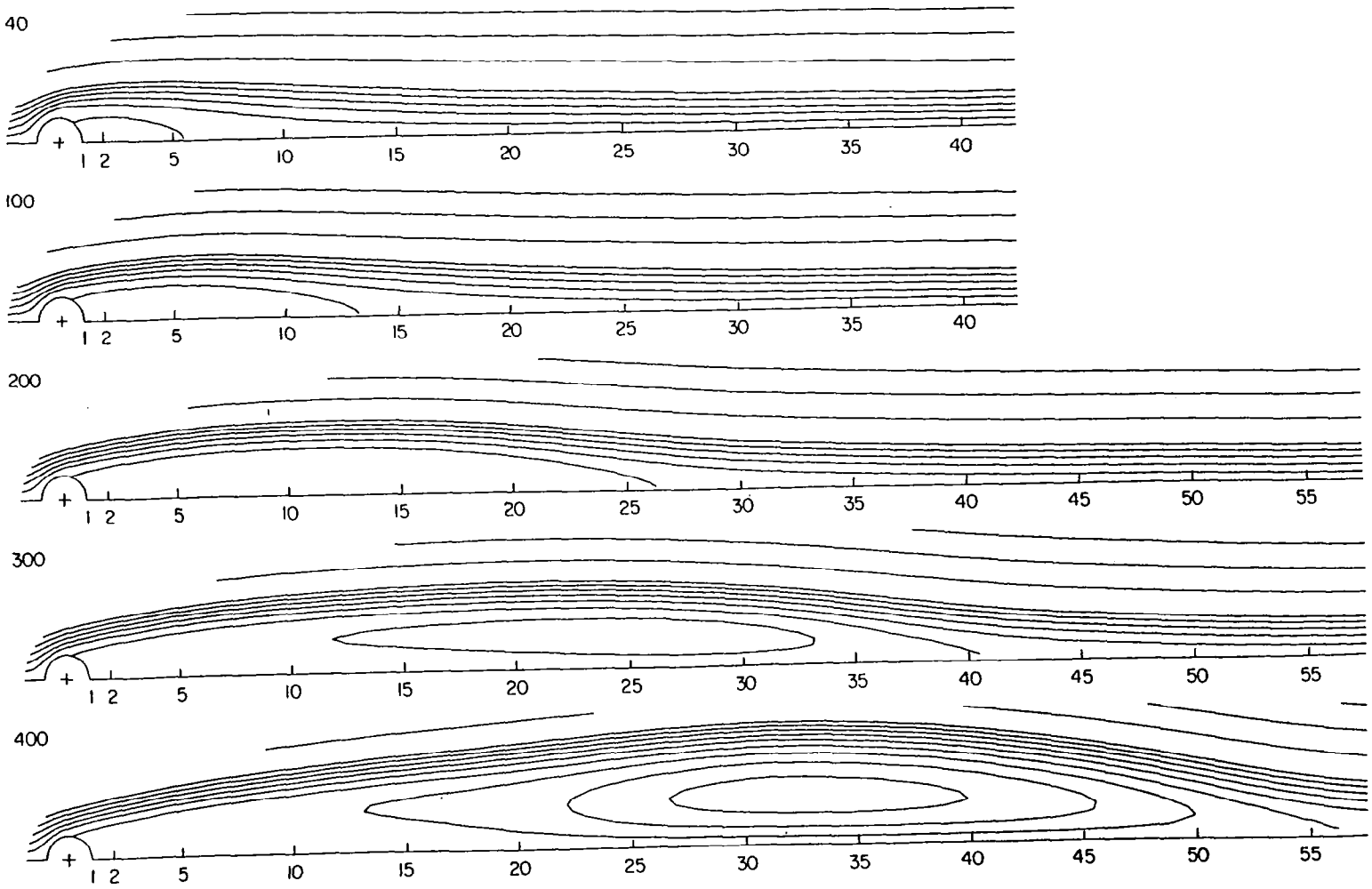


Figure 10: Streamlines (values ψ= -1.4, -1.2,..., .8, 1., 2.,..8., 9.) for different Reynolds numbers.

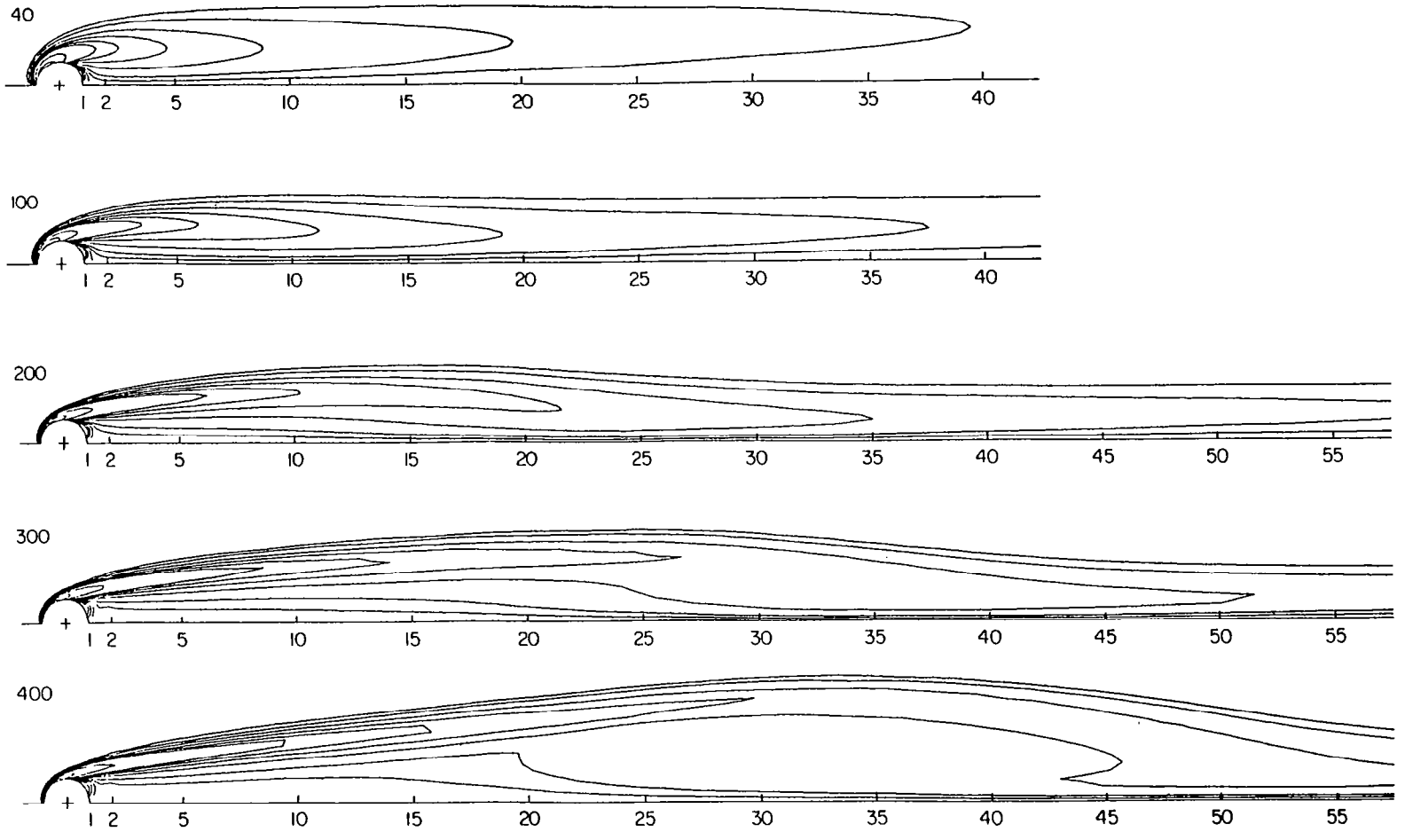


Figure 11: Contours of constant vorticity (values $\omega = -12., -8., -4., -2., -1., -.8, -.6, -.4, -.2, -1., 0., .1, .2, .5$) for different Reynolds numbers.

REFERENCES

1. Allen, D.N. de G. & Southwell, R.V.: Relaxation methods applied to determine the motion, in two dimensions, of a viscous fluid past a fixed cylinder. Quart. J. Mech. Appl. Math. 8 (1955) pp 129-145.

2. Batchelor, G.K.: A proposal concerning laminar wakes behind bluff bodies at large Reynolds number. J. Fluid Mech. 1 (1956) pp 388-398.

3. Brodetsky, S.: Discontinuous fluid motion past circular and elliptic cylinders. Proc. Roy. Soc. A 102 (1923) pp 542-553.

4. Dennis, S.C.R.: A numerical method for calculating steady flow past a cylinder. Proc. 5th Int. Conf. on Numerical Methods in Fluid Dynamics (Ed. A.I. van de Vooren & P.J. Zandbergen), Lecture notes in Physics, Springer, vol 59 (1976) pp 165-172.

5. Dennis, S.C.R. & Chang, G.Z.: Numerical solutions for steady flow past a circular cylinder at Reynolds numbers up to 100. J. Fluid Mech. 42 (1970) pp 471-489.

6. Fornberg, B.: A numerical study of steady viscous flow past a circular cylinder, J. Fluid Mech. 98 (1980) pp 819-855.

7. Gushchin, V.A. & Schennikov, V.V.: A numerical method of solving the Navier-Stokes equations. Zh. vychist. Mat. mat. Fiz. (1974), pp 512-520.

8. Imai, I.: On the asymptotic behaviour of viscous fluid flow at a great distance from a cylindrical body, with special reference to Filon's paradox. Proc. Roy. Soc. A 208 pp 487-516.

9. Keller, H.B.: The bordering algorithm and path following near singular points of higher nullity. Submitted to SIAM J. Sci. Stat. Computing.

10. Peregrine, D.H.: Note on the steady high-Reynolds-number flow about a circular cylinder. School of Mathematics, University of Bristol, Report No. AM-81-04 (1981).

11. Roache, P.J.: Computational fluid dynamics. Hermosa Publishers, Albuquerque (1976)

12. Saffman, P.G. & Tanveer, S.: Prandtl-Batchelor flow past a flat plate with a forward facing flap. Manuscript in preparation (1983).

13. Smith, F.T.:Laminar flow of an incompressible fluid past a bluff body: the separation, reattachment, eddy properties and drag. J. Fluid Mech. 92 (1979) pp 171-205.

14. Smith, F.T.: Comparisons and comments concerning recent calculations for flow past a circular cylinder. J. Fluid Mech. 113, pp 407-410.

15. Ta, P.L.: Étude numérique de l'écoulement d'un fluide visqueux incompressible autour d'un cylindre fixe ou en rotation. Effet Magnus. J. Méc. 14 (1975) pp 109-134.

16. Takami,H. & Keller, H.B.: Steady two-dimensional viscous flow of an incompressible fluid past a circular cylinder. Phys. Fluids Suppl. II pp 51-55.

NAVIER-STOKES SIMULATION OF HOMOGENEOUS TURBULENCE ON THE CYBER 205

C. T. WU,
J. H. FERZIGER,
AND
D. R. CHAPMAN
STANFORD UNIVERSITY
STANFORD, CALIFORNIA

AND

R. S. ROGALLO
NASA AMES RESEARCH CENTER
MOFFETT FIELD, CALIFORNIA

NAVIER-STOKES SIMULATION OF HOMOGENEOUS TURBULENCE ON THE CYBER 205

C. T. WU, J. H. FERZIGER, AND D. R. CHAPMAN

STANFORD UNIVERSITY, STANFORD, CALIFORNIA

AND

R. S. ROGALLO

NASA AMES RESEARCH CENTER, MOFFETT FIELD, CALIFORNIA

ABSTRACT

A computer code which solves the Navier-Stokes equations for three-dimensional, time-dependent, homogeneous turbulence has been written for the Cyber 205. The code has options for both 64-bit and 32-bit arithmetic. With 32-bit computation, mesh sizes up to 64^3 are contained within core of a 2 million 64-bit word memory. Computer speed timing runs were made for various vector lengths up to 6144. With this code, speeds a little over 100 Mflops have been achieved on a 2-pipe Cyber 205. Several problems encountered in the coding are discussed.

1. INTRODUCTION

Turbulent fluid motion is common to many branches of engineering and science. Since turbulence phenomena are highly nonlinear, they are not amenable to classical analytical approaches. Consequently, turbulence predictions are generally based on semi-empirical models. Experiments which generate model information are expensive, but are needed because current models are not generally accurate enough for engineering purposes. Detailed simulations of turbulent flows can help complement laboratory data. Direct numerical simulations of turbulent flows are more accurate than current semi-empirical computational methods and can be used to both generate physical understanding and to improve the models. In these simulations, turbulent flows are directly computed from the Navier-Stokes equations. Computations of this type are necessarily three-dimensional and time-dependent; they require a large number of grid points, and thus, long computation time. The Cyber 205 computer appears ideally suited for efficient numerical simulations of this type. Exploration of the use of the Cyber 205 for direct numerical simulation of turbulence is a principal objective of this work.

The basic code was written by one of the authors (RSR). It was modified to take advantage of the 205 compiler's automatic vectorizing capability. Vector syntax and special functions were applied to the code segments which could not be automatically vectorized. Finally, machine language instructions were used for the parts of the code that existing compiler could not handle.

In the next section, a description of the particular problem to be solved is given. In Section 3, the numerical methods used are discussed. This is followed by a brief description of the Cyber 205 at Colorado State University. The construction of long vectors is discussed in Section 5. In Section 6, performance data obtained to date are presented, and in Section 7, problems encountered are described. A typical simulation of homogeneous isotropic turbulence is presented in Section 8. In the final section, a brief statement of conclusions is presented.

2. PROBLEM STATEMENT

Homogeneous turbulent flows, of which there is a considerable variety, can be simulated numerically at low Reynolds number without using any turbulence model. In the flows we will consider, the computational domain contains a fixed mass of fluid within a rectangular parallelepiped, the opposing sides of which can move inward or outward with time. Thus, the cases which can be computed are quite varied: decaying homogeneous isotropic turbulence is generated if all six sides are stationary; turbulence undergoing uniform compression (or expansion) if all three pairs of sides move inward (outward) at same rate; turbulence undergoing one-dimensional compression, if one pair of sides moves inward; or turbulence undergoing plane strain if one pair of sides moves inward at the same rate a second pair moves outward, while the third pair remains stationary. Isotropic turbulence has been computed before, but turbulence undergoing compression or expansion has not. The compression cases are of interest, for example, in internal combustion engine modeling and in the interaction of turbulence with a shock wave.

It will be assumed that the Mach number is sufficiently small that the fluid is compressed uniformly in space, so that the fluid density depends only on time.

The governing Navier-Stokes equations for a fluid of uniform viscosity and uniform density in space are:

$$\frac{\partial u_i'}{\partial t} + u_j' \mathcal{U}_{i,j} + B_{kj}(u_i' u_j')_{,k} + B_{ki} P_{,k}' = \nu B_{kj} B_{\ell j} u_{i,k\ell}' \qquad i=1,3$$

$$B_{ji} u_{i,j}' = 0$$

where u^1_i p^1, v^2, and t are fluctuating velocity components, fluctuating pressure, kinematic viscosity and time respectively. The summation convention is implied. This set of governing Navier-Stokes equations allow us to simulate homogeneous turbulent flows in Lagrangian coordinate system that moves with the mean flow. Coordinate transformation tensor Bij is determined by:

$$\frac{dB_{ij}}{dt} + B_{ik} \mathcal{U}_{k,j} = 0$$

Note that mean strain rate tensor, $\mathcal{U}_{i,j}$ is zero and $B_{ij}=\delta_{ij}$ for isotropic homogeneous turbulence.

Periodic boundary conditions are applied in all three space directions. The velocity field is initialized to an isotropic state that satisfies continuity and has a given energy spectrum which approximates that of experimental isotropic turbulence.

3. NUMERICAL METHOD

The spectral method is used to compute all spatial derivatives. This method, which uses FFT's, is good for problems with periodic boundary conditions and has very high accuracy. To avoid aliasing in the nonlinear terms, both the truncation and phase shifting techniques are used.[1]

A second order Runge-Kutta method is used to advance the solution in time. Thus, all spatial derivatives need to be computed twice each time step. The time step was chosen small enough that no significant error is produced. It was determined by increasing the step size until the error was approximately 1 percent over the full integration period.

4. THE CYBER 205

The Cyber 205 we are using is the Colorado State machine with 2-pipes and a 2 million 64-bit word fast memory.[2] QTE Telenet has been used for data transfer between Stanford and CSU. We have found that both are reliable, convenient to use, and have provided satisfactory service so far.

Figure 1 shows the performance for add/multiply as function of vector length. The asymptotic performance which requires maximum vector length (65535) is 100 Mflops for 64-bit arithmetic and 200 Mflops for 32-bit arithmetic.[3]

It is obvious that the performance improves with vector length. Vector length 1000 (64-bit case) or 2000 (32-bit case) is required to reach 90 percent of the asymptotic performance. Constructing a code which uses long vectors is therefore important if maximum performance from the machine is to be obtained.

5. DATA MANAGEMENT

Based on the "longer vector gives better performance" philosophy, we chose to do the Fourier transforms in parallel. This will be explained in detail later.

In Figure 2, NX, NY, and NZ are the number of mesh points in the x, y, and z directions respectively; MY and MZ are called "pencil sizes".

On the first sweep, MZ x-y planes of data are Fourier transformed in the y direction in parallel. The transform length is NY, but by doing them in parallel, a vector length of NX/2*MZ*3 is achieved; the factor 3 is due to the simultaneous processing of three velocity components, and the factor 1/2 is due to only half of the modes are needed in wave space to represent a real function in physical space. To accomplish this, it is useful to lump every dependent variable into a single big array. The main array in our code is DATA(NX/2,NY,NZ,4,2); the dimensions represent x, y, z, a dependent variable index, and real and imaginary parts of a complex number.

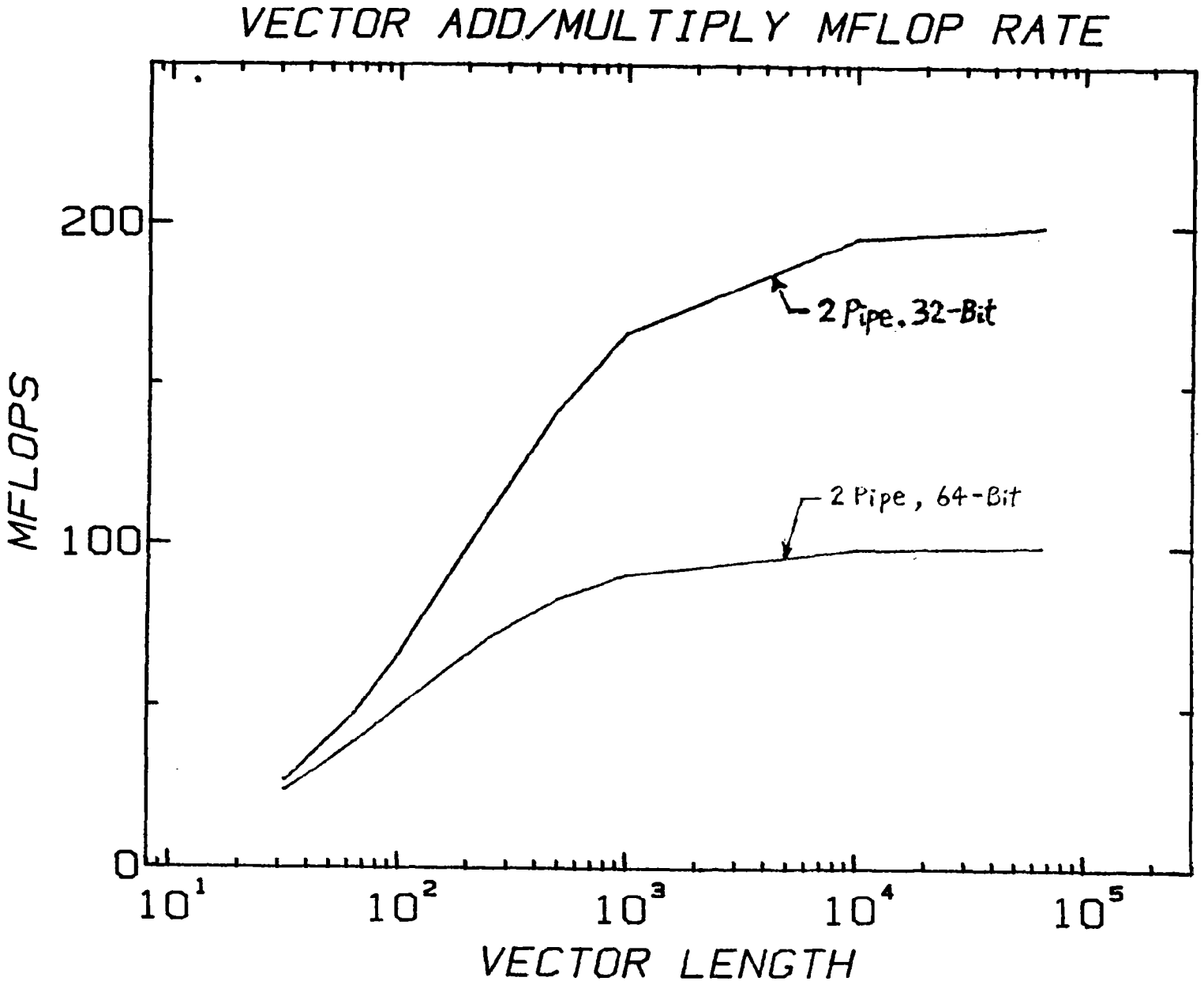

VECTOR ADD/MULTIPLY MFLOP RATE
200
100
0
MFLOPS
2 Pipe, 32-Bit
2 Pipe, 64-Bit
10^1
10^2
10^3
10^4
10^5
VECTOR LENGTH

Figure 1

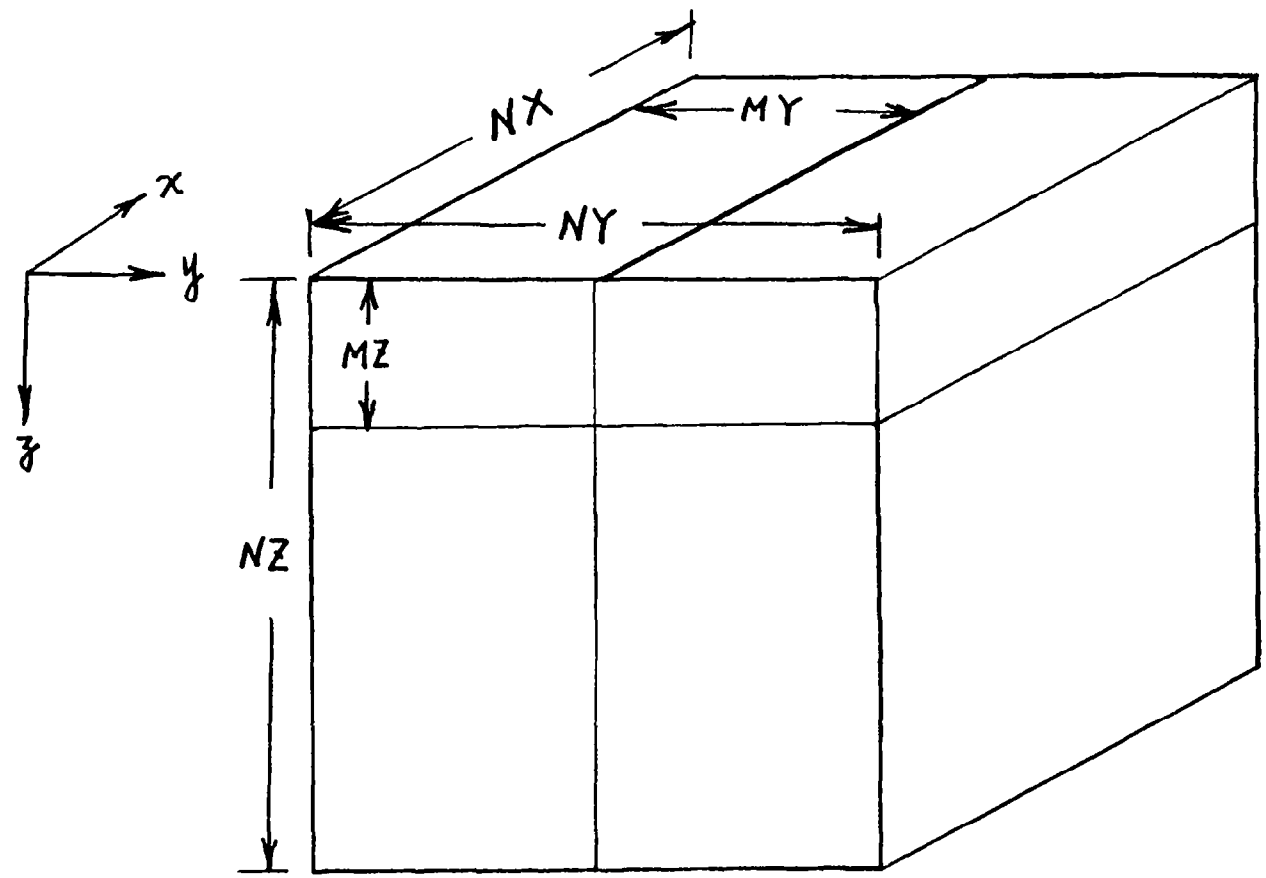


Figure 2

On the second sweep, MY x-z planes are processed. Fourier transforms in z and x directions are done on this sweep. The vector lengths are NX/2*MY*3 and NZ*MY*3 respectively.

A Cyber 205 vector is defined as a contiguous set of memory locations. Since the two sweeps are in different directions, an array transpose has to be done between sweeps and within the second sweep in order to keep processed data in a contiguous set of memory locations. The transpose is done by using gather instructions. The gather instruction puts array elements which are at various locations into a contiguous set of memory locations. An index vector is needed to pick up desired elements. Q8VGATHR function (64-bit) or Q8VXTOV subroutine (32-bit) is used to do the transposing. As the array gets bigger, so does the index vector length, and an appreciable amount of overhead working space is needed. In the 64^3 (32x16) run, the index vector has 17,408 elements.

6. COMPUTER PERFORMANCE

The performance data obtained to date, based on a hand count of the number of operations per time step, are presented in Table 1. The mesh size is given in column 2 (each node requires 7 words of data storage). The pencil size is given in column 3; this, together with mesh size, determines the vector length shown in column 4. The computational precision is given in column 5, the CPU time in column 6, and the CPU computation rate in column 7. The I/O time per step in seconds is meaningful only for runs with virtual memory paging. Explicit I/O would reduce I/O time considerably, but we have not yet attempted to use explicit I/O.

Figure 3 shows computation rate as function of vector length for our code on the 2-pipe CSU Cyber 205. It approaches an asymptote as vector length increases.

Comparing Runs 3 and 4, and Runs 5 and 6 in Table 1, it is found that the CPU time for a 32-bit (half) precision run is 60 percent of that for the corresponding 64-bit (full) precision run. We kept track of the timing in the transpose part of the code and found an interesting fact. In full precision runs, the transpose takes 15 percent of the CPU time; 85 percent of the CPU time is spent in floating point operations. In half precision runs, due to the lack of a half precision gather utility, the transpose takes the same amount of time as in full precision runs, while the floating point operations require only half of the full precision CPU time. Consequently, for half precision run, the transpose takes 25 percent of the total time.

Detailed timing from Run 8 shows that 51 percent of the CPU time is spent in the FFT subroutine, which contains 78 percent of the floating point operations. In other words, the FFT operates at 157.6 Mflops. The remaining 22 percent of the floating point operations are executed at 95 Mflops due mainly to shorter vector lengths and IF statements.

7. PROBLEMS ENCOUNTERED

Runs 7 and 8 of Table 1 require 3.5M words storage, and hence, do not fit within the 2M core memory at CSU with full precision. Thus, we must use 32-bit computation for efficient use of the CSU Cyber 205. Half-precision computation is sufficiently accurate for this code, and twice the operating speed is achieved.

TABLE 1.--PERFORMANCE OF CYBER 205 AT CSU
(2 PIPES WITH 2M 64-BIT WORD)

RUN	MESH SIZE (NODES)	PENCIL SIZE (NODES)	VECTOR LENGTH (IN FFT)	PREC. (BITS)	CPU TIME PER STEP (SEC.)	MFLOPS	I/O TIME PER STEP (SEC.)	MEMORY (M WORDS)	COMMENTS
1	8x8x8	8x8	192	64	0.014	23.5	-	0.02	in core
2	32x32x32	4x4	384	64	0.690	31.0	-	0.30	in core
3	32x32x32	32x32	3,072	64	0.399	53.6	-	0.69	in core
4	32x32x32	32x32	3,072	32	0.240	89.2	-	0.69	in core
5	64x64x64	16x16	3,072	64	3.378	59.6	56.6	2.70	paging
6	64x64x64	16x16	3,072	32	2.022	99.6	-	2.70	in core
7	64x64x64	32x16	4,608	32	1.980	101.7	-	3.47	in core
8	64x64x64	32x32	6,144	32	1.914	105.2	8.7	3.52	paging

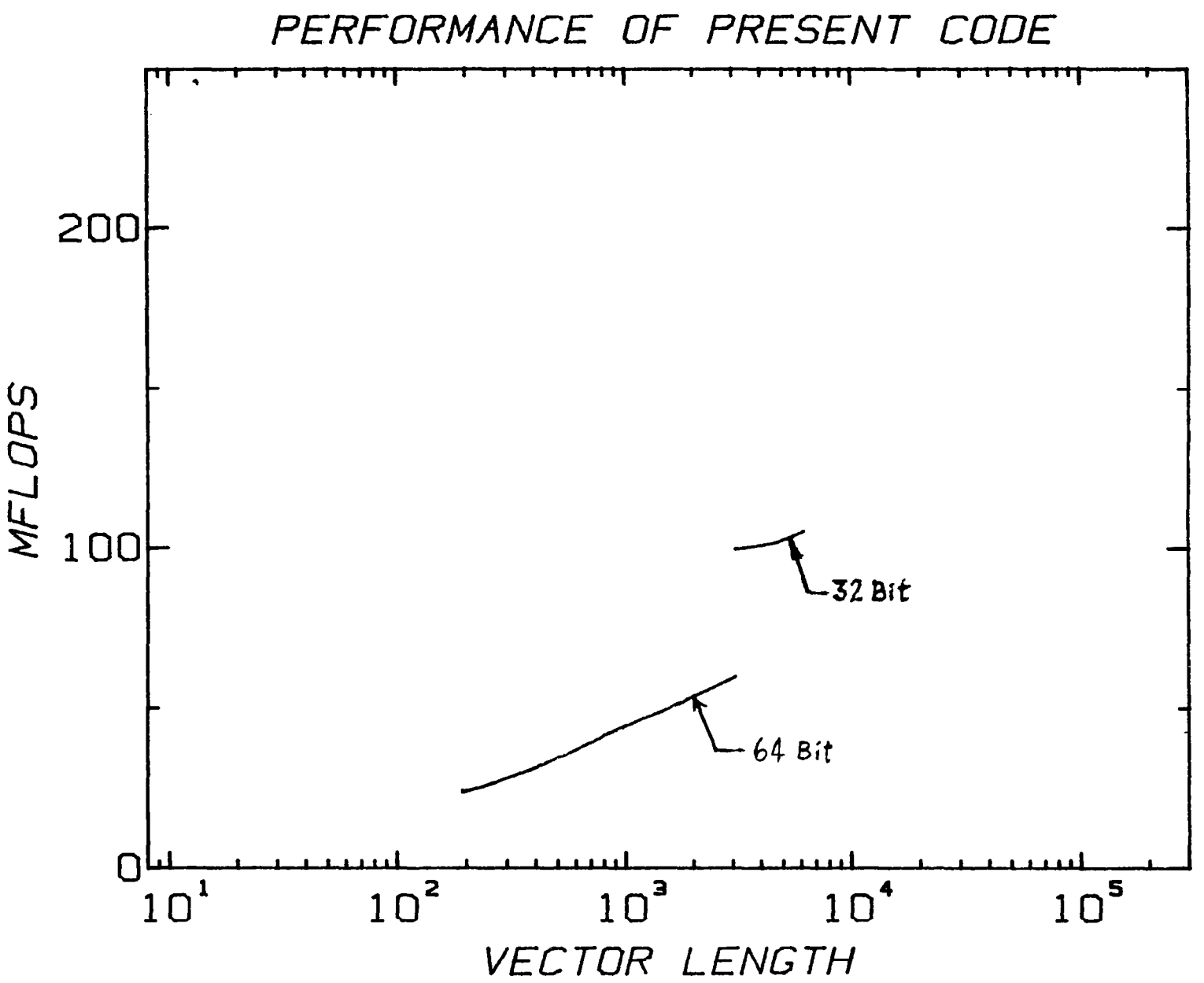

PERFORMANCE OF PRESENT CODE
200
100
0
MFLOPS
32 Bit
64 Bit
10^1
10^2
10^3
10^4
10^5
VECTOR LENGTH

Figure 3

Since there is no compiler available yet for half precision gather/scatter[4] calls, we have to use special Q8 calls[5] (machine instructions) to get the half precision code to compile properly on the CSU Cyber 205; the special Q8 instructions execute at full precision speed. Mr. Herbert Rothmund of CDC Sunnyvale was most helpful to us in providing these utilities.

It is apparent that the I/O rate is not balanced with the CPU time. The reason is that the CSU Cyber 205 has only two channels to transfer data between fast memory and disk and they are inherently slow. Solid-state backing memory (or equivalent) would speed up the data transfer rate. For our problem, faster I/O would allow us to go to 128^3 mesh size.

Since December 1982, three different compilers have been used: cycles 201109, L575, and 575B. Cycle 201109 did not have the half precision feature. Cycle L575 had half precision but lacked some automatic vectorization features. Cycle 575B, the most recent version, does not have gather/scatter in half precision. Further improvements are needed if users are to get optimum performance from this machine.

8. SIMULATION OF ISOTROPIC HOMOGENEOUS TURBULENCE

A typical simulation of homogeneous isotropic turbulent flow is presented in this section. Figure 4 shows the time history of the three-dimensional energy spectrum from initial time step to 300 time steps. Figure 5 shows the 3-D spectra of the components of the turbulent kinetic energy at time step 300. The flow is slightly anisotropic at low wavenumbers. This is due to the small number of modes at low wavenumbers.

All of these results are in excellent agreement with both experiments[6] and previous simulations.[7] Thus, we are confident that the code is performing satisfactorily and we will proceed to the simulation of compressed flows. The code presently runs at 1.9 second per time step for a 64^3 mesh on the 2-pipe Cyber 205; this compares with 5 seconds for the same type of code on the CRAY-1S in VECTORAL language.

9. CONCLUSION

In summary, we have written, debugged, and tested a code for solving the Navier-Stokes equations and for computing various turbulence statistical quantities. Most of the operations are readily vectorized, and 100 Mflops has been obtained for 64^3 mesh size in-core runs on a 2-pipe Cyber 205. The major problems encountered so far are concerned with the lack of compiler utilities, such as half-precision compiling capability for transpose operations.

The program works well and has been validated for homogeneous isotropic turbulence. The code will next be used to help develop turbulence models for compressed flow in engines.

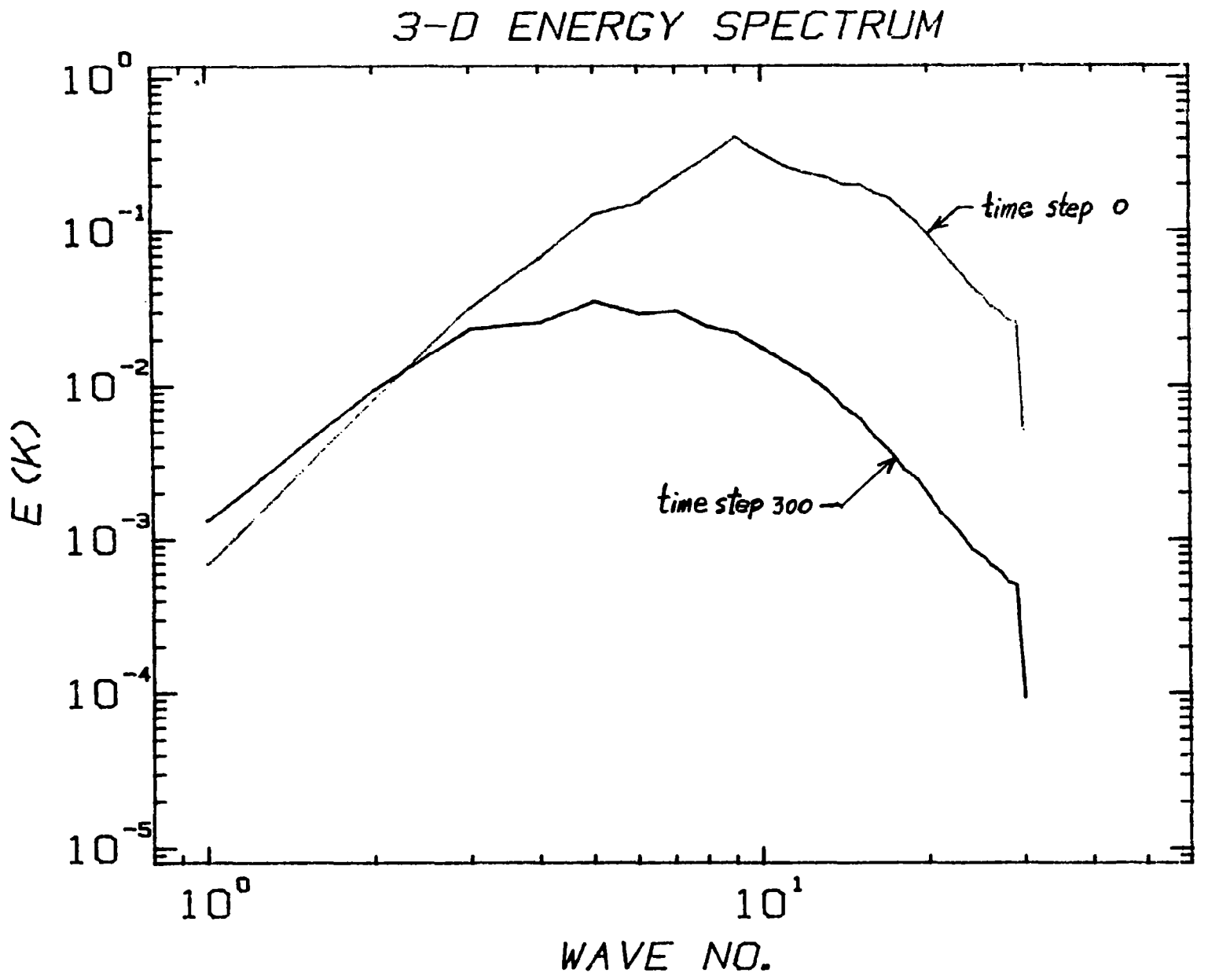

3-D ENERGY SPECTRUM
10^0
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
E (K)
time step 0
time step 300
10^0
10^1
WAVE NO.

Figure 4

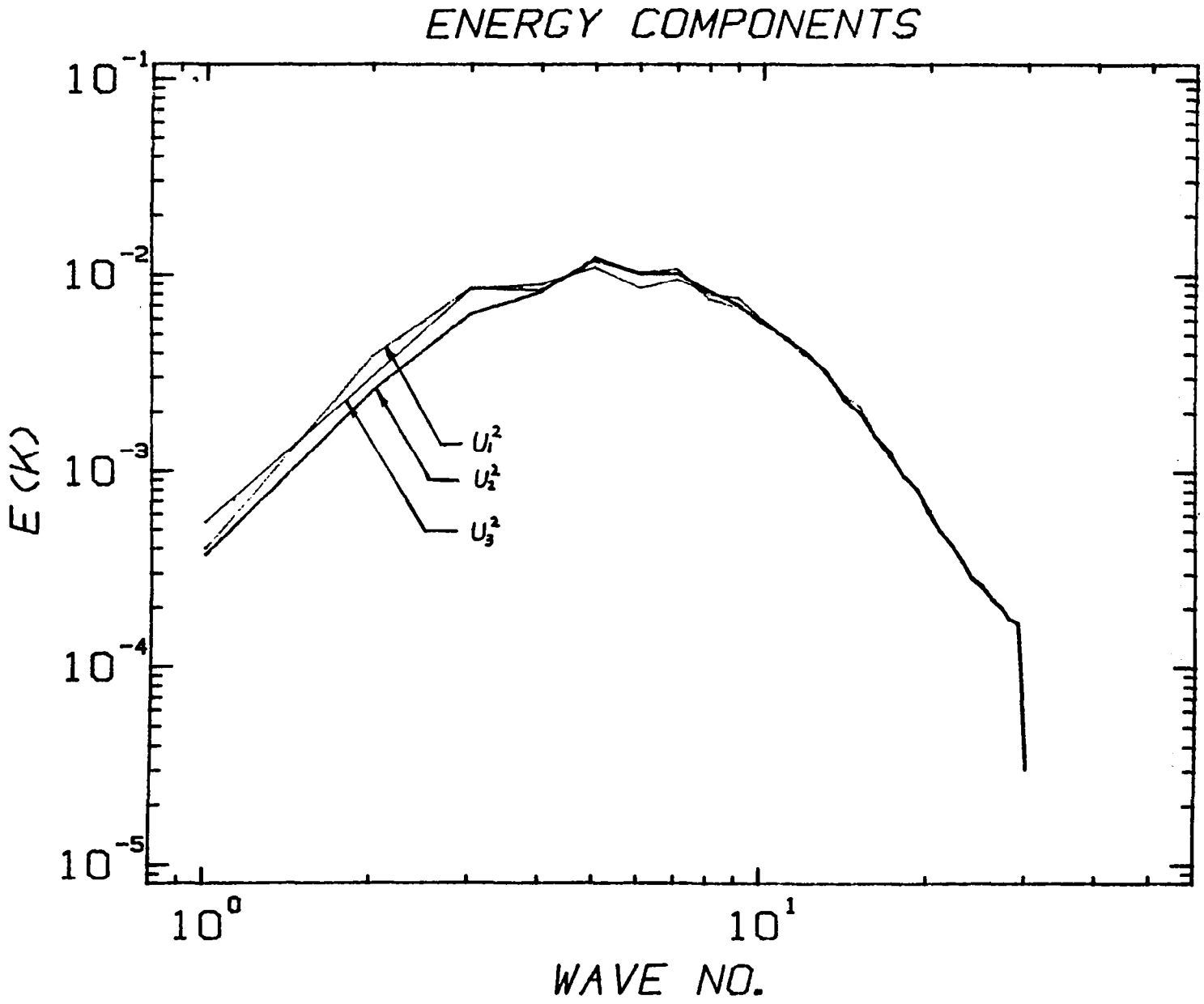

ENERGY COMPONENTS
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
E(K)
U_1^2
U_2^2
U_3^2
10^0
10^1
WAVE NO.

Figure 5

ACKNOWLEDGMENTS

This work was supported by Control Data Corp. under project No. 82S302. The authors would like to thank Mr. Herb Rothmund of CDC Sunnyvale, Mr. Art Lazanoff, Ms. Jeanne Adams and liaison staff of Colorado State University, and Dr. Nagi Mansour for their important contributions to this work.

NOMENCLATURE

B_{ij}	Coordinate transformation tensor
MY	Pencil size in Y-direction
MZ	Pencil size in Z-direction
NX	Number of mesh points in X-direction
NY	Number of mesh points in Y-direction
NZ	Number of mesh points in Z-direction
p'	Pressure fluctuations
t	Time
$\overline{U}_{i,j}$	Mean strain rate tensor
u_i	Velocity fluctuations in i-direction
x	Space coordinate
y	Space coordinate
z	Space coordinate
δ_{ij}	Kronecker delta
ν	Kinematic viscosity

REFERENCE

1 Rogallo, R. S., "Numerical Experiments in Homogeneous Turbulence," NASA TM81315, September 1981.

2 "Guide to Vector Processing Services," CSU Computer Center and the Institute for Computational Studies, October 1982.

3 Kascic, M. J., Jr., "Vector Processing on the Cyber 200," Workshop I, CSU, December 1982.

4 "CDC Cyber 200 Fortran Version 2 Reference Manual," Control Data Corporation, November 1981.

5 "CDC Cyber 200 Model 205 Computer System Hardware Reference Manual," Control Data Corporation, March 1981.

6 Comte-Bellot, G., and Corrsin, S., "Simple Eulerian Time Correlation of Full- and Narrow-Band Velocity Signals in Grid-Generated 'Isotropic' Turbulence," J. Fluid Mech. (1971), Vol. 48, part 2, pp. 273-337.

7 Shirani, E., Ferziger, J. H., and Reynolds, W. C., "Mixing of a Passive Scalar in Isotropic and Sheared Homogeneous Turbulence," Rept. No. TF-15, Thermosciences Division, Department of Mechanical Engineering, Stanford University, Stanford, Calif. May 1981.

EFFICIENT SPARSE MATRIX MULTIPLICATION SCHEME FOR THE CYBER 203

JULES J. LAMBIOTTE, JR.

NASA/LANGLEY RESEARCH CENTER

HAMPTON, VIRGINIA

Efficient Sparse Matrix Multiplication Scheme for the CYBER-203

Jules J. Lambiotte, Jr.
NASA/Langley Research Center
Hampton, Virginia

Abstract

Many important algorithms for solving problems in linear algebra require the repeated computation of the matrix-vector product $b = Ax$ where A is symmetric and sparse. Examples are the conjugate gradient and Lanczos methods.

This work has been directed toward the development of an efficient algorithm for performing this computation on the CYBER-203. The desire to provide software which gives the user the choice between the often conflicting goals of minimizing central processing (CPU) time or storage requirements has led to a diagonal-based algorithm in which one of three types of storage is selected for each diagonal. For each storage type, an initialization subroutine estimates the CPU and storage requirements based upon results from previously performed numerical experimentation. These requirements are adjusted by weights provided by the user which reflect the relative importance the user places on the two resources.

The three storage types employed were chosen to be efficient on the CYBER-203 for diagonals which are sparse, moderately sparse, or dense; however, for many densities, no diagonal type is most efficient with respect to both resource requirements. The user-supplied weights dictate the choice.

Introduction

Many of the important numerical techniques used today to solve linear equations require repeated computation of a symmetric matrix times a vector. Examples are the conjugate gradient method, with all its variants, for solving

simultaneous linear equations (refs. 1 and 2) and the Lanczos algorithm for eigenvalue and eigenvector extraction (ref. 3). These methods are particularly attractive when the matrix is sparse since, unlike direct methods, they do not require storage of the entire matrix. The matrix is only used to multiply a vector and to do this one only needs to know the nonzero elements and their position within the matrix.

The primary objective of this work has been to develop software for the CYBER-203 that provides an efficient means for computing $b = Ax$ when A is an $n \times n$, symmetric, sparse matrix.

Because use of vector hardware instructions on a vector processor has very definite implications about the storage, a user's desire to minimize both the required central processing unit (CPU) time and the total storage needed to represent A are often conflicting goals. Thus, a more specific objective of the work has been to design the software so that it provides alternative storage/computational procedures for the matrix A and automatically selects the procedure which best reflects the users relative concerns about minimizing the two resources.

These objectives have led to the development of a diagonal-based storage and computation scheme in which a preprocessing subroutine, CMPACT, chooses one of three storage methods for each diagonal using CPU and storage estimates and user-provided resource weighting information. The subroutine, CMXV, can be called repeatedly to compute Ax using the compact form of matrix A.

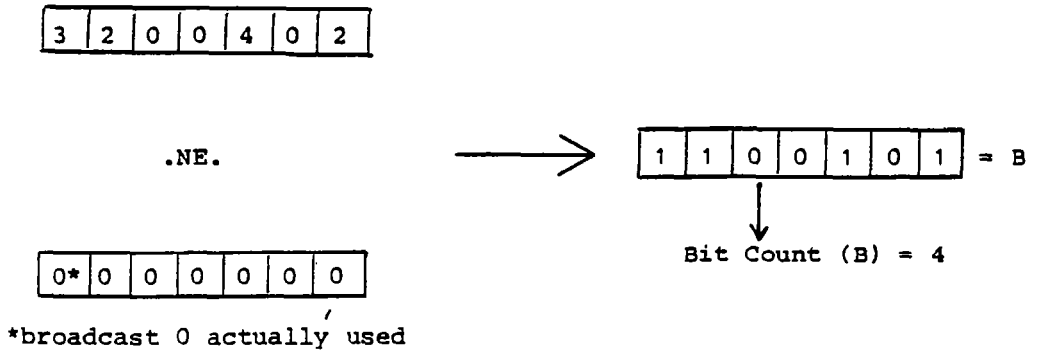
Subsequent sections of the paper will describe the relevant CYBER-203 instructions used, the diagonal-based algorithm with the tradeoffs between the methods, a description of the implementation used, and results for several sparse matrices.

CYBER-203 Characteristics

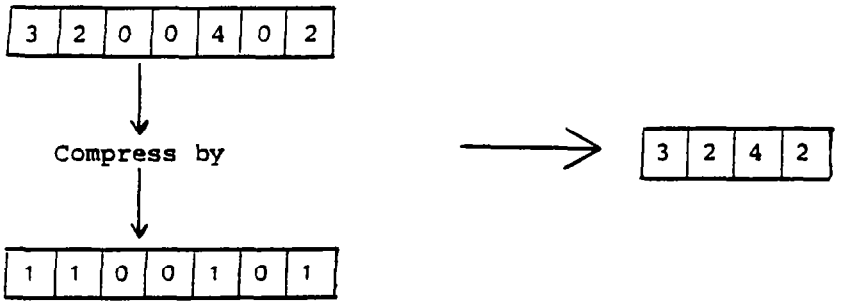
The CYBER-203 at Langley Research Center is a vector processing computer capable of producing 50 million floating point results (64 bit) for a vector addition and 25 million for a vector multiplication. It has one million words of bit addressable central memory in a virtual memory architecture.

The high CPU rates are achieved by operations on long vectors whose components, by definition, are consecutively stored in memory. However, if vector lengths are short (say, 50 or less), the fast scalar capability makes serial computation superior.

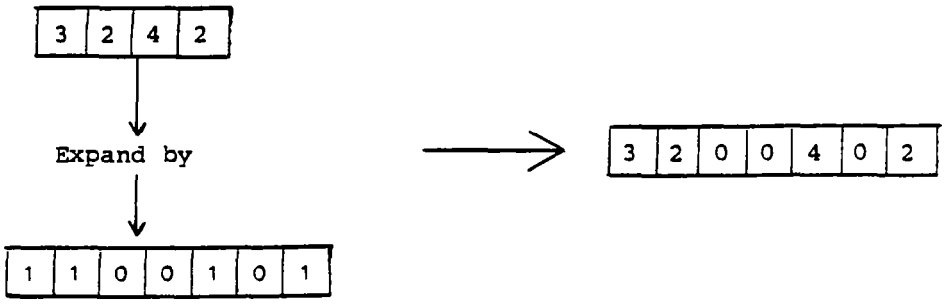
In addition to the usual arithmetic operations (+, -, *, and ÷), several nontypical hardware instructions exist which proved useful in this work. These were the vector compare, compress, expand, and bit count. Figure 1 demonstrates their use.



(a) Compare vector not equal to 0; result to bit vector, B; count "on" bits in B.



(b) Compress vector by bit vector.



(c) Expand compressed vector by bit vector.

Figure 1. CYBER-203 nontypical vector instructions.

Diagonal-Based Matrix Multiplication

It is possible to describe the multiplication process $b = Ax$ for a matrix A in terms of elements of each diagonal. Let $A(\ell)$ denote the ℓ^{th} superdiagonal (also the ℓ^{th} subdiagonal since A is symmetric) and let $A_k(\ell)$ be the k^{th} component. That is, $A_k(\ell) = a_{k,k+\ell} = a_{k+\ell,k}$. The procedure for computing $b = Ax$ for the $n \times n$ matrix A is

$b_k \leftarrow A_k(o)\ x_k \qquad k = 1,2,\ldots,n.$

For $\ell = 1,2,\ldots,n-1.$

$$b_k \leftarrow b_k + A_k(\ell)\ x_{k+\ell} \qquad \text{for } k = 1,2,\ldots,n-\ell \tag{1}$$

$$b_{k+\ell} \leftarrow b_{k+\ell} + A_k(\ell)\ x_k \qquad \text{for } k = 1,2,\ldots,n-\ell \tag{2}$$

End F

Note that if A is banded, ℓ need only go from 1 to the bandwidth β and that if any diagonals are identically zero, they can be easily identified and all computation for them in (1) and (2) can be omitted.

The diagonal-based scheme has been selected as the foundation for this work for several reasons:

a. Nonzero structure of real problems - Many matrices arising from finite difference or finite element formulations naturally lead to a sparsity pattern in which most of the nonzeros lie along a few of the diagonals. The 5 diagonal matrix arising from central differencing of Poisson's equation is an extreme example. Of course, there the pattern is so predictable that special storage techniques are not needed; but for irregular grids, or more complex equations with more complicated differencing, the sparsity is not so easily specified. This is especially true in finite element formulations where one of the strengths of the method is the ability to use nonuniform elements.

b. Vectorization - The $n - \ell$ multiplications and additions in equations (1) and (2) can be carried out by vector operations of length $n - \ell$.

c. Symmetry of diagonals - The ℓ^{th} subdiagonal is also the ℓ^{th} superdiagonal. Since equations (1) and (2) are identical in form, the storage and computation most appropriate for the subdiagonal is also most appropriate for the superdiagonal.

Storage Tradeoffs

The vector computations implied in equations (1) and (2) assume $A(\ell)$ is available as a vector of length $n - \ell$. However, if the diagonal is relatively sparse, one might not want to store the entire diagonal with all its zeros. In fact, if the diagonal is very sparse, neither vector storage nor vector computation is likely to be very efficient.

Described below are three types of diagonal storage and their associated computation to execute equations (1) and (2).

Full Vector (Type 1) - Here the entire diagonal is stored including any zeros. Vectors of length $n - \ell$ are used. This mode will be most efficient when $A(\ell)$ is very dense.

Compressed Vector Plus Bit Pattern (Type 2) - Here only the nonzeros are stored along with a bit vector to give positional information within the diagonal. The computation is identical to that with type 1 diagonals after an expand is performed to generate the full diagonal $A(\ell)$. The extra expand makes type 2 CPU requirements always exceed type 1, but the storage can be considerably less.

Compressed Vector Plus Row Pointers (Type 3) - Here the assumption is that $A(\ell)$ is so sparse that it will be inefficient to expand the compressed vector. Equations (1) and (2) are executed serially making use of the row indices stored for positional information.

Figures (2) and (3) show the CPU and storage requirements for a diagonal of length 1000 as a function of density. A comparison of the two figures shows that, unfortunately, one cannot identify intervals of density where a particular diagonal type is most efficient with respect to both resources. For instance type 3 CPU is least for $d < 0.11$ but has a greater storage requirement than type 2 for $d > 0.02$. Even in those regions where one diagonal type is most efficient for both resources (type 1 for very dense and type 3 for very sparse), the boundaries of these regions vary with the length of the diagonal.

Since the minimization of both resources is frequently not possible, and since different users may attach different importances to the two resources, it was decided to let the user influence the storage selection through resource weighting factors. To implement this the initialization subroutine, CMPACT, does the following for each diagonal:

(1) Estimates the CPU and storage requirements for each of the three candidate types.

(2) Applies a user-supplied weight to compute the weighted resource requirement for each method.

(3) Selects the storage type that minimizes the sum of the two weighted resource requirements.

That is, denoting the predicted storage and CPU requirements for the j^{th} diagonal type by s_j and c_j respectively, their minimum by s_m and c_m, the users specified weighting by s_w and c_w, then the normalized and weighted resource, r_j, for the j^{th} diagonal type is computed as

$$r_j = \frac{s_j}{s_{min}} s_w + \frac{c_j}{c_{min}} c_w \qquad j = 1,2,3$$

Subroutine CMPACT computes r_j and selects the diagonal type which yields the minimum value of r.

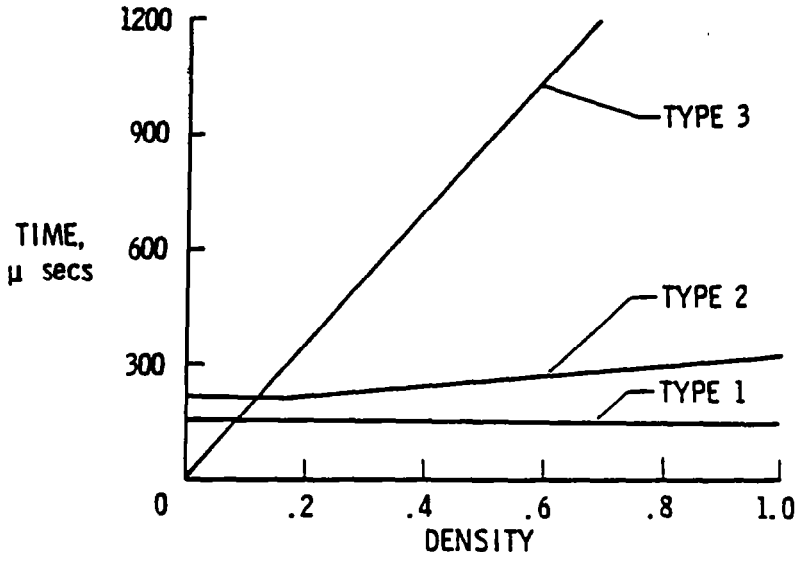


FIGURE 2. CPU TIME FOR DIAGONAL WITH LENGTH 1,000.

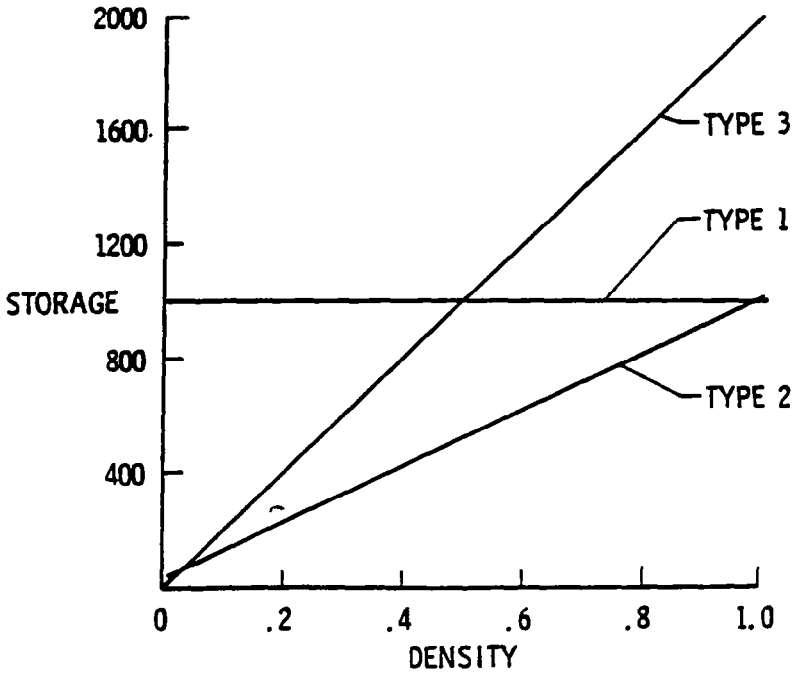


FIGURE 3. STORAGE REQUIREMENTS FOR DIAGONAL WITH LENGTH 1,000.

For this approach, CMPACT must be able to estimate s_j and c_j for all n and d. The storage estimates are easily made in terms of a diagonal of length n having z nonzeros.

$$s_1 = n$$

$$s_2 = z + w$$

$$s_3 = 2z$$

where w is the least number of 64-bit words needed to hold n bits.

The CPU estimates were obtained by timing the computation for a range of n and density d. For type 1 and 3 diagonals, single formulas were obtained, but the complexity of the expand used in type 2 diagonal computation required a table of values. The time in microseconds to perform the computations implied in equations (1) and (2) for a single diagonal can be estimated by

$$C_1 = 29 + 0.122\ n$$

$$C_2 = \text{See Table I}$$

$$C_3 = 7 + 1.74\ z$$

Since these values are used only in a selection process, their accuracy to a percent or two is sufficient.

Table I.- Type 2 diagonal CPU times (microseconds) as a function of diagonal length n and density d.

n	d						
	0.	.1	.2	.4	.6	.8	1.0
100	53	53	53	57	60	63	68
500	123	123	124	141	160	176	197
5000	901	901	918	997	1134	1280	1429

Implementation

The matrix is received in subroutine CMPACT in its expanded form as an N by IB array. Each of the IB diagonals is treated individually as the compact representation, array C, is formed. C is a linear array in which the pertinent data for the L^{th} diagonal is stored behind that for the $L - 1^{st}$ diagonal. As illustrated in figure 4, this can be, for types 1, 2, or 3 respectively, either the entire diagonal, the nonzero bit pattern for the diagonal followed by the nonzeros, or the nonzeros and index data. A vector compare with broadcast zero generates the bit pattern and provides the number of nonzeros and density. If the weighting procedure determines that the diagonal should be type 2 or 3, a compress is performed. In addition, two integers for each diagonal are stored in a separate array. The first identifies the diagonal type and the second the number of nonzeros in the diagonal.

The subroutine returns to the user the CPU and storage estimates for the user provided weights. In addition the estimates for combinations $s_w = 1$, $c_w = 0$ and $s_w = 0$, $c_w = 1$ are returned to aid the user to adjust his weights in subsequent computations.

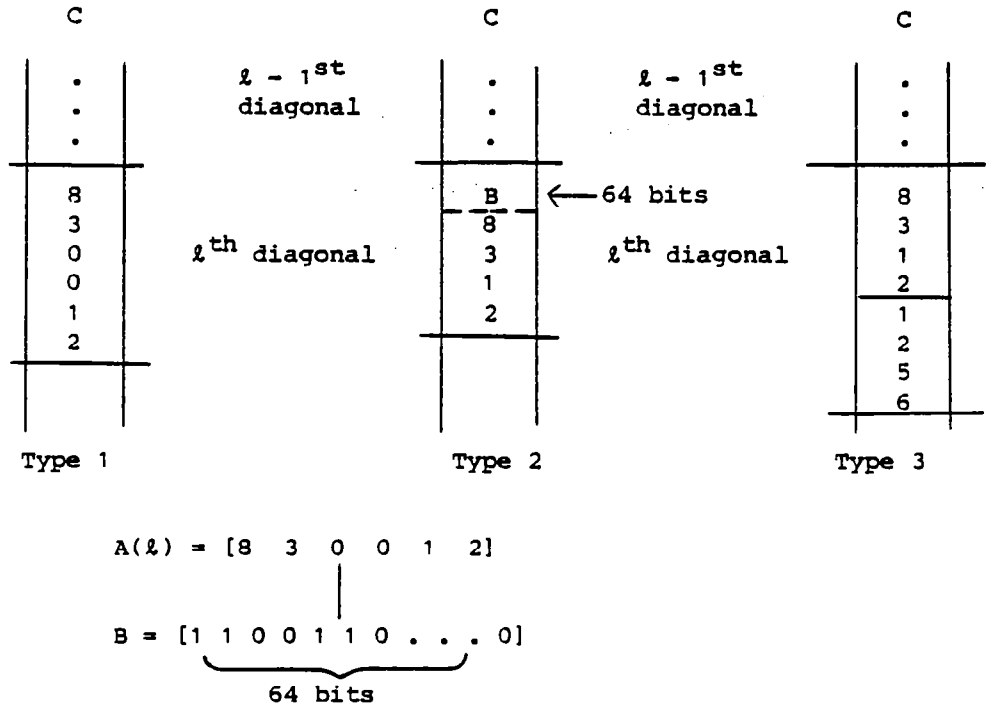


Figure 4 - Storage for A(ℓ) (n - ℓ = 6).

Results

Results from two test matrices are presented here to demonstrate the effect and control the user has on the matrix storage and computational requirements by giving the statistics for different combinations of s_w and c_w. Refer to Tables II and III.

Case 1 - This is a randomly generated matrix with 400 equations and a bandwidth of 21. The densities are approximately uniformly distributed between 0. and 1. The average density is 55.7%. The storage selection that minimizes the CPU time (1.57 msec; mostly type 1) yields the largest storage requirement. The selection to minimize storage (4713 words; mostly type 2) yields the largest computation time.

Case 2 - This is a sparse matrix resulting from a finite element formulation with triangular elements and 3 degrees of freedom at each node. The matrix has 1086 equations, a bandwidth of 81, and an average density of 7.8%. Most of the diagonals are sparse. Of the 81 diagonals, 57 are less than 5% dense and approximately half of the nonzeros are on the four diagonals closest to the main diagonal. Because of the relatively few dense diagonals, most of the diagonals are type 2 (to minimize storage) or type 3 (to minimize CPU).

Both examples demonstrate the conflicting goals of minimizing both resources. They also show that use of the weighting factors can give the user a rather wide range of resource distributions. For instance, in the second example a weighting of 1 for c_w leads to a CPU time that is minimum but a storage requirement which is 1.73 times that if one set $s_w = 1$. However, setting $s_w = 1$ yields a CPU time which is 2.6 times the minimum. A reasonable middle ground occurs when $s_w = c_w = 0.5$. In this case, the CPU is 1.09 times the minimum and the storage is 1.2 times the minimum.

Table II.- Case 1; 21 × 400 random matrix.

Weights		Resources		Diagonal Selection		
c_w	s_w	CPU (Secs)	Storage	1	2	3
0	1	.00271	4713	1	20	0
.3	.7	.00217	4950	7	13	1
.5	.5	.00193	5481	11	9	1
.7	.3	.00174	6053	14	5	2
1	0	.00157	7495	19	0	2

Table III.- Case 2; 81 × 1086 finite element matrix.

Weights		Resources		Diagonal Selection		
c_w	s_w	CPU (Secs)	Storage	1	2	3
0	1	.01680	8032	1	72	8
.3	.7	.00800	9200	3	17	61
.5	.5	.00703	9622	3	8	70
.7	.3	.00682	9820	3	4	74
1	0	.00646	13883	8	0	73

Summary

This paper has described a computational and storage algorithm for sparse matrix multiplication on the CYBER-203. The multiplication is performed using diagonals of the matrix as the candidate vectors since this is where nonzero patterns predominate in many scientific applications. Three types of diagonal sparsity patterns are identified (roughly speaking, either dense, moderately sparse, or sparse) and storage and computational procedures developed for each.

Since, for most densities, no single diagonal type minimizes both storage and CPU requirements, an initialization subroutine selects the most "efficient" type for the diagonal based on estimated resource requirements and user-provided weights which indicate the relative importance the user attaches to each resource.

Examples are given which illustrate that, for a given matrix, the weights can be used to achieve minimal CPU time (at the expense of storage) or minimal storage (at the expense of CPU time) or some compromise between the two.

References

1. Hestenes, M. R. and Steifel, G., "Methods of Conjugate Gradients for Solving Linear systems", NBS Journal of Research, 49, 1952.

2. Kershaw, D. S., "The ICCG Method for the Iterative Solution of Systems of Linear Equations", J. Computational Physics, 26 (1978), pp. 43-65.

3. Wilkinson, J. H., The Algebraic Eigenvalue Problem, p. 388, Oxford University Press (Clarendon), London and New York, 1965.

MODELING MATERIAL FAILURE WITH A VECTORIZED ROUTINE

S. M. CRAMER
AND
J. R. GOODMAN

DEPARTMENT OF CIVIL ENGINEERING
COLORADO STATE UNIVERSITY

FORT COLLINS, COLORADO

MODELING MATERIAL FAILURE WITH A VECTORIZED ROUTINE

S. M. Cramer
Research Associate
Dept. of Civil Engineering
Colorado State University
Ft. Collins, Colorado

J. R. Goodman
Professor
Dept. of Civil Engineering
Colorado State University
Ft. Collins, Colorado

ABSTRACT

The computational aspects of modeling material failure in structural wood members are presented with particular reference to vector processing aspects. Wood members are considered to be highly orthotropic, inhomogeneous, and discontinuous due to the complex microstructure of wood material and the presence of natural growth characteristics such as knots, cracks and cross grain in wood members. The simulation of strength behavior of wood members is accomplished through the use of a special purpose finite element/fracture mechanics routine, program STARW (STrength Analysis Routine for Wood). Program STARW employs quadratic finite elements combined with singular crack tip elements in a finite element mesh which accounts for the complexities inherent in wood structural members. The need to use a highly refined finite element mesh to adequately model material behavior, results in the formulation of thousands of simultaneous equations which must be generated and solved repeatedly to model the nonlinear failure process which occurs. The availability of the CYBER 205 at Colorado State University has made implementation of program STARW at the level described not only possible, but also relatively economical. Vector processing techniques are employed in mesh generation, stiffness matrix formation, simultaneous equation solution, and material failure calculations. The paper addresses these techniques along with the time and effort requirements needed to convert existing finite element code to a vectorized version. Comparisons in execution time between vectorized and nonvectorized routines are provided.

INTRODUCTION

Accurate knowledge of the strength of a structural member is essential information to the design engineer concerned with structural safety and efficient material use. A means to predict material strength is necessary, since all materials exhibit some variability in strength and it is not feasible to pnysically test every structural member to determine its load carrying capacity. The sophistication of strength prediction models have generally advanced, not only with the discovery and refinement of new computational methods, but also with the increase in computer capabilities which enable erficient application of the new methods.

In the case of wood structural members, the current strength prediction method is a highly approximate procedure based on empirical concepts from the 1930's. This results in a strength prediction that is relatively uncertain. The current strength prediction procedure is based on the results of physical tests because until now it has not been possible to mathematically model wood member failure and rationally predict strength. The most obvious difficulties; orthotropic material properties, the presence of knots and associated grain deviations, and the presence of cracks from seasoning and partial material failure, can now be successfully modeled with program STARW (STrength Analysis Routine for Wood) (2).

The nature of the nonlinear failure modeling process, presents a computational problem of such a large magnitude that it can not be efficiently accomplished on computers that do not have the capacity of a CYBER 205. Program STARW represents a case where modest effort in invoking vector processing syntax has not only made implementation of the program possible, but has also resulted in a relatively economical solution.

AN OVERVIEW ON MATHEMATICALLY MODELING WOOD MEMBER FAILURE WITH STARW

Program STARW uses two-dimensional orthotropic finite elements to model behavior in the longitudinal-transverse plane of a loaded wood member. Tensile load is applied in the longitudinal direction as shown in Fig. 1.

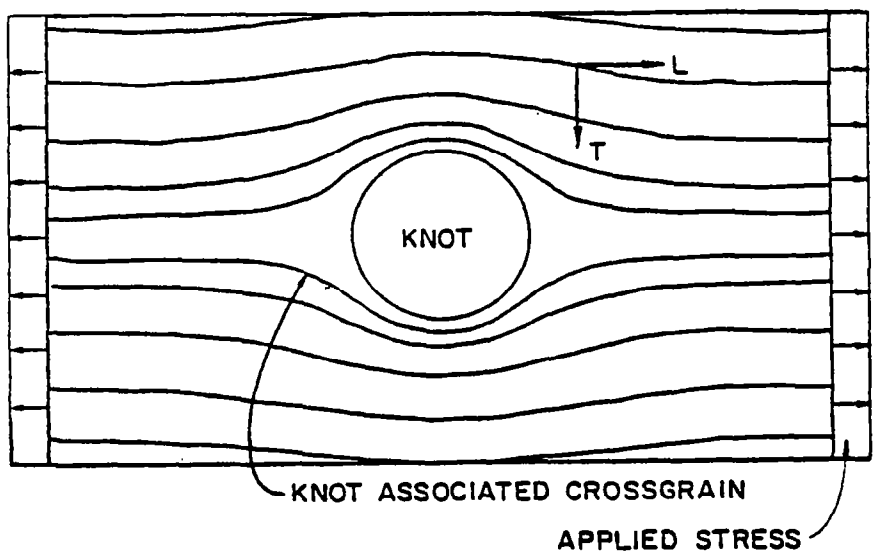


Figure 1. Loaded Wood Structural Member
(Longitudinal-Transverse Plane)

A knot in a structural specimen of wood creates localized grain deviation as indicated in Fig. 1. This grain deviation has an extremely important effect on stress distributions at locations near the knot (3). An iterative procedure to locate mesh coordinates corresponding to the grain deviation around a knot is employed in program STARW. This procedure relates distortion of wood grain around a knot to streamlines of laminar fluid flow around an elliptical object and has therefore been named the "flow-grain analogy" (4).

Utilizing the flow-grain analogy, a representative finite element mesh is automatically constructed of eight node quadrilateral elements, six node triangular elements, and eight node singular elements. Since tangential elastic stiffness of wood may be as little as 1/20 of the longitudinal elastic stiffness, all three types of finite elements are required to model different elastic material behavior in the longitudinal and tangential directions. Appropriate elastic stiffness values for each element are automatically assigned.

Singular elements are used to model material behavior around the tip of cracks that form as the load on the member is increased. These elements were developed using theory from linear elastic fracture mechanics (1). Experimental investigations have indicated that cracks in structural lumber will usually form and propagate along a grain line. Thus, cracks are modeled by program STARW by "unzipping" the finite element mesh along the material separation and placing the singular elements around the crack tip. A resulting finite element mesh is shown in Fig. 2. The "unzipping" process and placement of the singular elements are performed automatically upon cue by the user when the appropriate failure conditions are indicated in the program output.

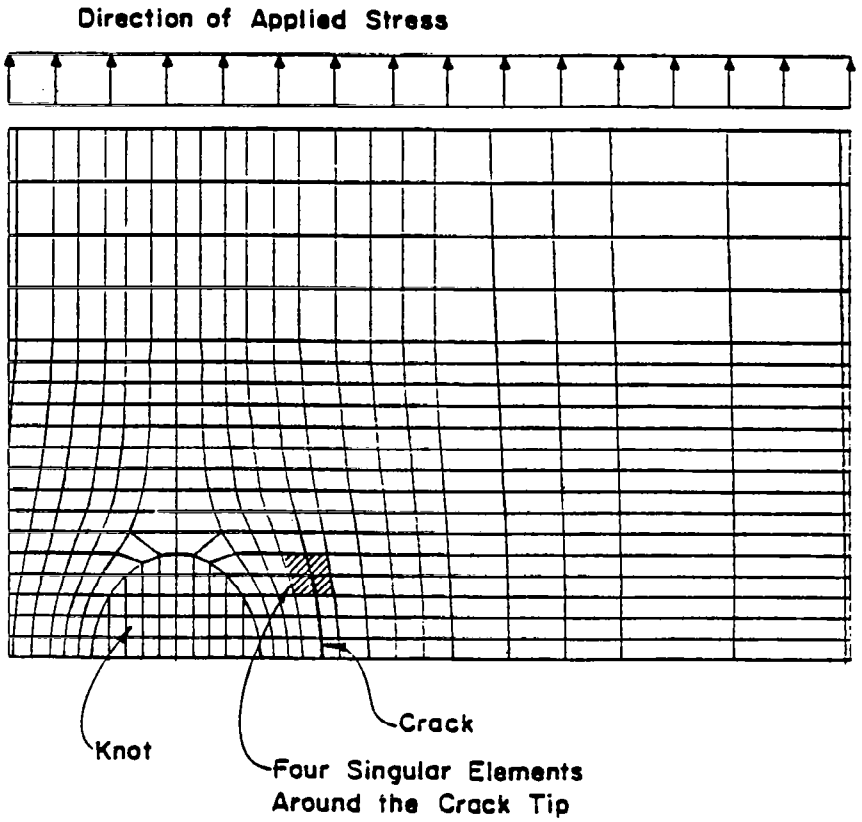


Figure 2. Example Finite Element Mesh Including Crack

The output directly calculated from each analysis is as follows:

1) Horizontal and vertical displacement at each node in the mesh.

2) Stresses for each element, parallel-to-grain, perpendicular-to-grain, and shear.

3) Stress intensity factors resulting from the use of singular elements.

4) A failure summary that indicates to the user what appropriate action should be taken to model the next step in the failure process.

The stress intensity factors directly reflect the strength of the stress field around the crack tip. The stress intensity factors are compared within the program to a fracture criteria for structural wood members to determine if the existing crack propagates at a given applied load. The element stresses are compared to a failure criteria for structural wood members to determine if a crack will form near the element under consideration. The results of these comparisons are expressed in the failure summary.

Analyses are performed repeatedly with stress and stress intensity factors monitored at each step and compared within the program logic to the fracture/failure criteria. As the load on the member is increased, more cracking and material failure occurs. The user, based on the information in the failure summary and the overall stress picture, gives the program the necessary information to model the successive step in the failure process. In the future, as research progresses, program logic will be expanded to include the decision making process the user currently makes based on the failure summary. Failure may be continually modeled in this fashion until the member under consideration has failed to the point where it cannot resist an increase in load. At this point, the predicted strength is realized. In studying the behavior of a wood member, 30 analyses may typically be performed before the member reaches its capacity. A simplified diagram of the failure model is contained in Fig. 3.

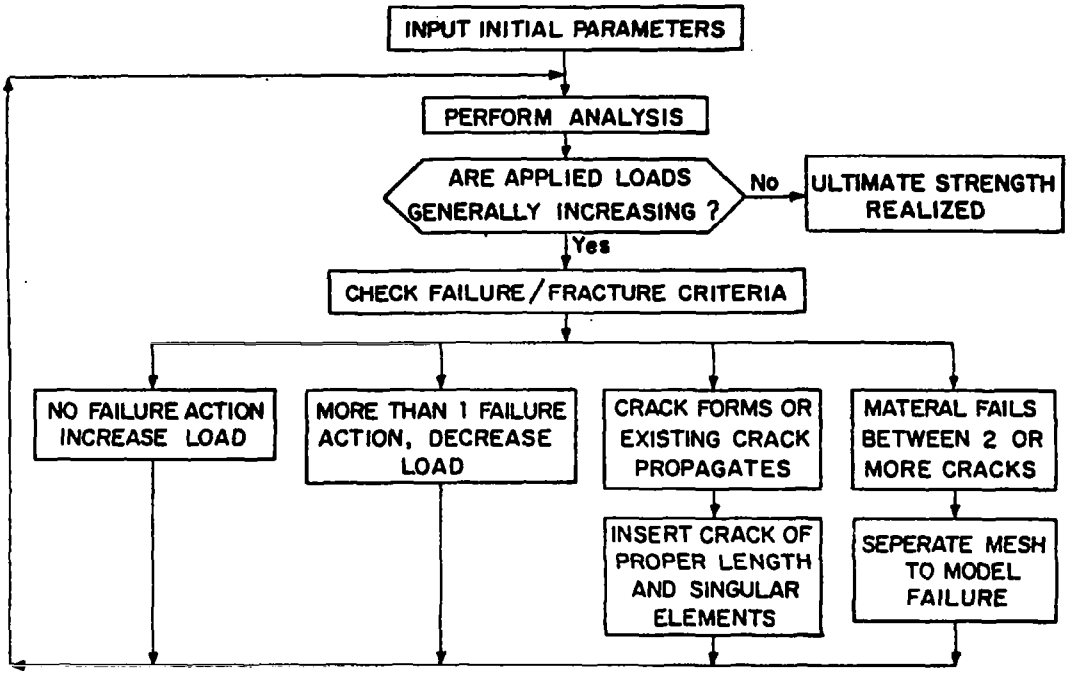


Figure 3. Strength Prediction Model

ASPECTS AND IMPLICATIONS OF VECTORIZATION

For each analysis, program STARW performs five general sets of computations:

1) Generation of a suitable finite element mesh using the flow-grain analogy and an unzipping process to include cracks.

2) Formation of a set of simultaneous equations which may be 2000 to 5000 equations in length.

3) Solution of the simultaneous equations using Gauss elimination.

4) Calculation and coordinate transformation of element stresses based on the solution vector and the element grain angles.

5) Computations with the failure/fracture criteria using element stresses and stress intensity factors as input.

Routines included in items 1 through 4 existed in limited form and were executed for small problems on a CYBER 720 prior to application on the CYBER 205. Failure calculations in item 5 and additional mesh generation capabilities were added and designed specifically for use on the CYBER 205. After compiler inaduced vectorization proved to be inadequate, in significantly reducing execution time, it became apparent that it was essential to explicitly vectorize selected portions of the existing routines. At the same time, it was not the primary goal of the project to expend unlimited effort to achieve the maximum in vector processing, rather the goal was to produce a powerful research tool that could be economically implemented. The bulk of the conversion (and execution time savings) were achieved with modest effort after becoming familiar with vector processing syntax.

To date, a means to vectorize the iterative solution of the fluid mechanics equations contained in the flow-grain analogy has not been established. This is not of great concern since, as in many finite element routines, mesh generation does not account for a significant portion of the total execution time. However, the unzipping of the finite element mesh to model cracks involves, in part, a uniform renumbering of nodal points. This renumbering is easily accomplished with basic vector commands since nodal coordinates are stored in vector form.

Formation of the set of simultaneous equations can typically take from 5 to 50 per cent of total execution time in a unvectorized finite element analysis. In program STARW, a 16 by 16 element stiffness matrix must be constructed for each element and properly combined with other element stiffness matrices to form the coefficient matrix (global stiffness matrix) of the simultaneous equations. Formation of the 16 by 16 matrix involves dot products or vectors of length 16. Some time savings is attained here through the use of the CYBER Q8SDOT command even though the vector length is rather small.

Solution of the simultaneous equations typically requires 40 to 90 percent of the total execution time of a finite element analysis. The 90 percent figure is not uncommon for large two-dimensional analyses. Therefore, large time savings can be attained by vectorizing the solution algorithm alone. In program STARW, Gauss elimination is used to decompose the global stiffness matrix, followed by a back substitution to obtain the solution. For the problem under consideration the stiffness matrix is banded and symmetric, and therefore, only the upper diagonal half of the matrix is stored. Furthermore, if the global stiffness matrix is stored in columns rather than rows, then adjacent terms in a row of the global stiffness matrix will be stored contiguously. Since Gauss elimination involves operations of one row upon another, by storing the matrix as described, each row will be a vector. "Gather" and "scatter" vector formation commands are unnecessary. Gauss elimination involves operations on the matrix rows in a number of nested DO loops. Vectorization of even the inner most loop results in large time savings. Back substitution involves repeated dot products of previously formed vectors. This can again be easily accomplished with the CYBER Q8SDOT command. An unvectorized and otherwise identical vectorized portion of the back substitution is shown in Fig. 4 to illustrate typical vectorization.

```
      DO  460  J = 2, JEND
      J1 = I1 + J - 1
      B(I1) = B(I1) - A(J,I1) * B(J1)
460   CONTINUE
```

```
      LE = JEND - 1
      J1 = I1 + 1
      B(I1) = B(I1) - Q8SDOT (A(2, I1; LE), B(J1; LE))
```

Figure 4. Example DO Loop and Corresponding Vector Syntax

With the solution of the equations established, element strains and stresses can be calculated in global coordinates. Since this calculation is essentially the same for every element, and care is taken to store the necessary quantities in vector form, basic vector operations accomplish this task. The solution vector is found in the global coordinate system and thus the calculated stresses are also expressed in this system. It is desireable, however, to know the stresses in the coordinate system of each element or the perpendicular-to-grain and parallel-to-grain directions. The element stresses must be transformed according to the element grain angle. Since the element grain angles are stored contiguously and in order, this computation can be accomplished with basic vector commands.

To complete an analysis, the stresses and stress intensity factors for cracks must be inserted into the failure/fracture criteria. The failure/fracture criteria interfaces the mathematical results from an analysis

to the real life failure actions. Required information includes the maximum stresses and their locations within the flow-grain mesh. Since stresses are stored in element order in vectors, this information can be obtained much quicker and more easily by using CYBER Q8 commands than with scalar search algorithms.

To put the vectorization discussed into perspective, a typical problem was analyzed using unvectorized and vectorized routines. Since unvectorized versions of the mesh generator (item #1) and the maximum stress searching routine (item #5) do not exist, vectorized routines had to be used for both sides of the example. The example problem consisted of 4180 degrees of freedom (equations) and for simplification no cracks were included. The corresponding CPU execution times for different phases of the analysis are shown in Table 1.

TABLE 1. EFFICIENCY OF EXECUTION TIME FOR VECTORIZED ROUTINES

	UNVECTORIZED TIME IN SEC.	VECTORIZED TIME IN SEC.	EFFICIENCY UNVECT/VECT
MESH GENERATION	1.90	1.90	1.00
STIFFNESS MATRIX FORMATION	4.84	2.80	1.73
SOLUTION OF EQUATIONS	97.87	4.91	19.90
MISCELLANEOUS COMPUTATIONS	5.05	4.60	1.10
TOTAL	109.66	14.21	7.70

As clearly shown for this problem, the vectorized equation solver was 20 times faster than its otherwise identical unvectorized version. This savings, along with other vectorization, reduced analysis time by nearly a factor of

eight. One will note that while the miscellaneous computations were somewhat insignificant in the unvectorized analysis, they take on new importance in the vectorized analysis. Additional effort may be well spent in further vectorization of the miscellaneous computations.

CONCLUSIONS

Failure in wood members is being successfully modeled and analytically investigated in greater detail than before possible through implementation of program STARW on the CYBER 205 (2). An understanding of material failure is essential to accurately predict member strength and to safely and efficiently use the material in engineering application.

Vectorization of program STARW has reduced an unwieldly and expensive, nonlinear failure modeling method into an efficient research tool. Vectorization of existing routines need not be a lengthy and laborious effort to achieve execution time savings. It has been shown that careful organization of operands into vectors and modest effort in invoking vector syntax can cut program execution time by a factor of nearly 8 for a typical problem in this research. The largest savings is realized in the solution of the simultaneous equations.

While use of program STARW is expected to provide new information on fracture and failure in wood members, the availability of machines with the capabilities of the CYBER 205, in general holds promise for advances in the analytical modeling of all materials. These advances in research will initiate new applications of materials and more efficient and reliable use of materials in existing applications.

ACKNOWLEDGMENTS

The authors gratefully acknowledge the support of this research by the National Science Foundation under Grant No. CME-79-18170 and the Institute for Computational Studies at Colorado State University.

REFERENCES

1. Atluri, S. N., A. S. Kobayshi and M. Nakagaki, "An Assumed Displacement Hybrid Finite Element Model for Linear Fracture Mechanics", Int. Journal of Fracture Mechanics, Vol. 10, p. 1281-1287, 1975.

2. Cramer, S. M., "Analytical Strength and Fracture Prediction in Lumber", Ph.D. Dissertation in progress, Civil Engineering Department, Colorado State University, Fort Collins, Colorado, 1983.

3. Goodman, J. R. and J. Bodig, "Tension Behavior of Wood - An Anisotropic Inhomogeneous Material", Final Report to the National Science Foundation (Grant No. ENG 76-84421), Structural Research Report No. 32, Department of Civil Engineering, Colorado State University, Fort Collins, Colorado, 1979.

4. Phillips, G. E., J. Bodig and J. R. Goodman, "Flow-Grain Analogy", Wood Science, 14(2):55-64, 1981.

ALGORITHMS FOR SOLVING LARGE SPARSE SYSTEMS OF SIMULTANEOUS LINEAR EQUATIONS ON VECTOR PROCESSORS

RONALD E. DAVIS

CONTROL DATA CORPORATION

SUNNYVALE, CALIFORNIA

ALGORITHMS FOR SOLVING LARGE SPARSE SYSTEMS OF SIMULTANEOUS LINEAR EQUATIONS ON VECTOR PROCESSORS

R. E. Davis

Control Data Corporation
Sunnyvale, CA

ABSTRACT

Very efficient algorithms for solving large sparse systems of simultaneous linear equations have been developed for serial processing computers. These involve a reordering of matrix rows and columns in order to obtain a near triangular pattern of non-zero elements. Then an LU factorization is developed to represent the matrix inverse in terms of a sequence of elementary gaussian eliminations, or pivots.

In this paper we show how to adapt these algorithms for efficient implementation on vector processors. Results obtained on the CYBER 200 Model 205 are presented for a series of large test problems which show the comparative advantages of the triangularization and vector processing algorithms.

PRELIMINARY RESULTS IN IMPLEMENTING A MODEL OF THE WORLD ECONOMY ON THE CYBER 205: A CASE OF LARGE SPARSE NONSYMMETRIC LINEAR EQUATIONS

DANIEL B. SZYLD

INSTITUTE FOR ECONOMIC ANALYSIS
NEW YORK UNIVERSITY

NEW YORK, NEW YORK

Preliminary Results in Implementing a Model of the World Economy on the Cyber 205: A Case of Large Sparse Nonsymmetric Linear Equations

Abstract

Daniel B. Szyld
Institute for Economic Analysis
New York University

A brief description of the Model of the World Economy implemented at the Institute for Economic Analysis is presented, together with our experience in converting the software to vector code.

For each time period, the model is reduced to a linear system of over 2000 variables. The matrix of coefficients has a bordered block diagonal structure, and we show how some of the matrix operations can be carried out on all diagonal blocks at once.

We present some other details of the algorithms and report running times.

1. Description of the Model

The first input-output model of the world economy was originally developed for the United Nations by Leontief, Carter and Petri [1977] as a tool for evaluating alternative long-term economic policies. The most recent version that has been implemented spans the period 1970-2030 in 10-year intervals. The model is dynamic in the sense that the solution for each 10-year period requires information obtained from the solution for the previous period. In this paper we focus on the solution of a single time period.

In the current version of the model, the world is divided into 16 regions (r=16) and for each of the regions the detailed economic activities are described by a set of linear algebraic equations of the form

$$A_i \underline{y}_i + S_i \underline{w} = 0 \quad (i = 1,\ldots,r). \qquad (1)$$

The components of the vectors $\underline{y}_i$ correspond to levels of domestic production, imports, and exports of goods and services, and so on, for each region, and $\underline{w}$ is the vector of total world exports. In addition there are global constraints described by the equation

$$\sum_{i=1}^{r} G_i \underline{y}_i = 0 \quad , \qquad (2)$$

which imposes the consistency among regional trade relations.

A more detailed description of the model can be found in Leontief, Carter and Petri [1977], Duchin and Szyld [1979], and Szyld [1981].

All the matrices involved are very sparse. For example

A_i could be 200 × 250 with 2500 nonzeros.

S_i could be 200 × 50 with 50 nonzeros.

G_i could be 50 × 250 with 100 nonzeros.

Each matrix A_i has more columns than rows and therefore some components of $\underline{y}_i$ have to be prescribed.

If $\underline{x}_i$ are the vectors of unknown components of $\underline{y}_i$ and M_i and E_i are the corresponding submatrices of A_i and G_i, the whole model for a single time period can be regarded as a linear system of equations of over 3000 variables with a nonsymmetric bordered block diagonal matrix of coefficients of the form:

$$\begin{vmatrix} M_1 & & & & S_1 \\ & M_2 & & & S_2 \\ & & \ddots & & \vdots \\ & & & M_r & S_r \\ E_1 & E_2 & \cdots & E_r & O \end{vmatrix} \begin{vmatrix} \underline{x}_1 \\ \underline{x}_2 \\ \vdots \\ \underline{x}_r \\ \underline{w} \end{vmatrix} = \begin{vmatrix} \underline{b}_1 \\ \underline{b}_2 \\ \vdots \\ \underline{b}_r \\ \underline{O} \end{vmatrix} \qquad (3)$$

where the blank blocks in the matrix are zero blocks.

When the model was first implemented, the program for the solution of (3) inverted the matrices M_i and stored the inverses. The approximate computer time to perform this task was 4 hours on a PDP-11. The (dense) inverses were saved for subsequent runs during which they were updated depending on the components of $\underline{y}_i$ prescribed and on changes in the matrices A_i. Each of these subsequent runs required 110 seconds on an IBM 370 for each time period.

The set of prescribed components of $\underline{y}_i$ and the matrices are used to determine a scenario, i.e., a set of economic assumptions. Studies carried out with the World Model compare

results of different scenarios, i.e., the implications of the different assumptions. The consequences of the introduction of new technologies, different development strategies, or shifts in trade patterns are among the numerous scenarios that can be analyzed. Thus, the World Model is a flexible tool to analyze alternative policies. Several large scale empirical studies have been carried out with this model. The most recent ones are reported in Leontief and Duchin [1983], Leontief and Sohn [1982], Leontief, Koo, Nasar and Sohn [1983] and Leontief, Mariscal and Sohn [1982].

To make this tool much more flexible we needed to greatly reduce the computational resources required to run a scenario. A first step in that direction was the application of sparse matrix techniques for the solution of (3). In the present implementation the matrices A_i are stored using a sparse scheme, i.e., only the nonzero elements are stored, together with some integer arrays indicating their locations. A single array of approximate length 3200 contains all vectors $\underline{x}_i$, i=1,...r. Other such arrays contain the vectors $\underline{b}_i$, the nonzero values of the matrices S_i and G_i, or other data objects. Similarly, objects like the nonzeros of the matrices M_i appear in single arrays of length close to 5000.

2. Method of Solution

The algorithmic details of the solution of (3) are given in Duchin and Szyld [1979], Szyld [1981], and Furlong and Szyld [1982]. Here we enumerate the operations for the solution of (3) very schematically.

loop 1. For i=1,...,r

1.1. Read A_i, G_i, S_i, and the prescribed elements of $\underline{y}_i$

1.2. Produce M_i, E_i and $\underline{b}_i$

1.3. Obtain factorization of M_i

loop 2. For i=1,...,r

2.1. Prepare different right hand sides with columns of S_i

2.2. Solve systems with matrix M_i

loop 3. Obtain $\underline{w}$

loop 4. For i=1,...,r

4.1. Compute $\underline{b}_i - S_i\underline{w}$

4.2. Solve $M_i\underline{x}_i = \underline{b}_i - S_i\underline{w}$

The factorization of the matrices Mi (in step 1.3) and the solution of several linear systems with them (in steps 2.2 and 4.2) are performed with routines from the MA28 set developed by Duff [1977].

We report the running times for a single time period with this method of solution without any vector code in Table 1.

Table 1.

System/compiler options	CPU sec.
IBM 370/168	~38
IBM 3033	~20
Cyber 205, no options	11.46
Cyber 205, vectorization by the compiler	9.04

Architectural features combined with the sparse matrix techniques resulted in running times three to ten times faster than the 110 seconds that subsequent runs required after computation of the inverses in the first implementation of the World Model. The goal is now to obtain vector code for the Cyber 205 that will further reduce the overall running time.

3. Code vectorization

The redesign of the World Model software for its efficient use on the Cyber 205 was conceived in three phases:

I. Elementary operations over all regions

II. The MA28 package inner loops

III. New concepts for MA28

Phase I consists essentially of the vectorization of all operations except those associated with the factoring of the matrices M_i and solutions of the corresponding linear systems. Those operations correspond to steps 1.2, 2.1, and 4.1. Each of these steps has a different structure but they all are loops operating on vectors of length about 200, inside another loop of length 16. The basic idea was to split the outer loop and perform simultaneously the operations on all vectors of the different regions, i.e., on vectors of length of about 3200. Cyber 205 FORTRAN commands such as scatter, gather and bit operations were used throughout.

We illustrate the vectorization of step 4.1. The length of $\underline{w}$ is about 50. S_i is a rectangular matrix of about 200 rows, with only one nonzero entry per column. It is stored as a vector with an accompanying integer array indicating in which

row each nonzero entry lies. The following FORTRAN statements are part of sequential code for step 4.1.

```
      DO 100 II=1,NREG
      IBEG=(II-1)*NTRADE
      IBEGB=IPNTB(II)-1
      DO 50 I=1,NTRADE
         INDEX=KTRDBG(IBEG+I)+IBEGB
         B(INDEX)=B(INDEX)-EXPSH(I+IBEG)*W(I)
 50    CONTINUE
100 CONTINUE
```

The running time for these loops was 1008 μsec. Different vectorization options were analyzed. One of them consisted of scattering the vectors that contain the nonzero values of S_i and $\underline{w}$ to vectors of length of about 3200 and then performing the triad operation. This required 9514 clock cycles, or about 190 μsec. The version adopted performs the multiplication of the vectors containing the nonzeros of S_i and $\underline{w}$ first, a vector operation of length about 800, scatters that vector and performs the final subtraction in 7250 clock cycles or 145 μsec, a gain of a factor of 7 from the sequential code.

Similar gains have been achieved in the other portions of the code vectorized in phase I. Unfortunately only a small portion of the total running time of the World Model is spent in the code vectorized in phase I. Thus the overall gain was relatively small.

About 30% of the total running time of the World Model is spent on routines of the MA28 package in which the matrices M_i are factored (step 1.3), and solutions with many right hand sides computed (steps 2.2 and 4.2). At the present time we have completed only part of phase II, the vectorization of some of the inner loops in the MA28 set.

Due to the startup time in any vector operation, it is common practice to look into the length of the vectors involved in the operation to decide if the vectorization is really worthwhile. In codes for sparse matrices, the vector length for an operation is usually the number of nonzero elements in a particular row or column, and thus varies within the code. The technique used in this case is to assess if the vector length is above a particular value and branch the process of that particular row or column to vector or sequential code. The running time of the code incorporating these features is 7.33 CPU seconds, cf. Table 1.

Phase III, not yet implemented, consists of reconceptualizing the MA28 set. We will investigate the possibility of solving several right hand sides simultaneously, as well as other features like special treatment of right hand sides with few nonzero elements.

Acknowledgment. I would like to thank Valdimir Roytman for his assistance throughout the project and Faye Duchin for useful comments on an early draft of the manuscript.

References

Faye Duchin and Daniel B. Szyld, Application of Sparse Matrix Techniques to Inter-Regional Input-Output Analysis, Economics of Planning 15(1979) 142-167.

Iain S. Duff, MA28- a set of FORTRAN Subroutines for Sparse Unsymmetric Systems of Linear Equations, Report R.8730, A.E.R.E. Harwell, HMSO, London, 1977.

Kenneth E. Furlong and Daniel B. Szyld, World Model Solution Programs, Release 1, Institute for Economic Analysis, New York University, July 1982.

Wassily Leontief, Anne P. Carter and Peter A. Petri, The Future of the World Economy, Oxford University Press, 1977.

Wassily Leontief and Faye Duchin, Military Spending: Facts and Figures, Worldwide Implications and Future Outlook, Oxford University Press, New York, 1983.

Wassily Leontief, James Koo, Sylvia Nasar and Ira Sohn, The Future of Non-fuel Minerals in the U.S. and World Economy: Input-Output Projections from 1980-20000, Lexington-Heath, Lexington, Mass., 1983.

Wassily Leontief, Jorge Mariscal and Ira Sohn, The Prospects for the Soviet Economy to the Year 2000, Journal of Policy Modeling 5(1983) 1-18.

Wassily Leontief and Ira Sohn, Economic Growth, in Just Faaland (editor), Population and the World Economy in the 21st Century, Basil Blackwell, Oxford, 1982, pp. 98-127.

Daniel B. Szyld, Using Sparse Matrix Techniques to Solve a Model of the World Economy, in Iain S. Duff (editor), Sparse Matrices and their Uses, Academic Press, London, 1981, pp. 357-365.

MONTE CARLO CALCULATIONS OF ELEMENTARY PARTICLE PROPERTIES

G. S. GURALNIK
BROWN UNIVERSITY
PROVIDENCE, RHODE ISLAND

TONY WARNOCK
CRAY RESEARCH
MINNEAPOLIS, MINNESOTA

AND

CHARLES ZEMACH
LOS ALAMOS NATIONAL LABORATORY
LOS ALAMOS, NEW MEXICO

MONTE CARLO CALCULATIONS OF ELEMENTARY PARTICLE PROPERTIES

G.S. Guralnik
Brown University and Los Alamos National Laboratory

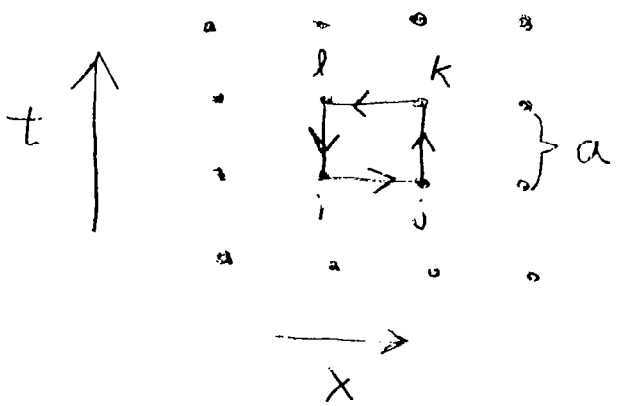
Tony Warnock
Cray Research

Charles Zemach
Los Alamos National Laboratory

The object of our project is to calculate the masses of the "elementary particles". This ambitious goal apparently is not possible using analytic methods or known approximation methods. However, it is probable that the power of a modern super computer will make at least part of the low lying mass spectrum accessible through direct numerical computation. Initial attempts by several groups at calculating this spectrum on small lattices of space time points have been very promising. Using new methods and super computers we have made considerable progress towards evaluating the mass spectrum on comparatively large lattices. Even so, we are examining regions of space just barely large enough to contain the particles being examined. Only more time and faster machines with increased storage will allow calculations of systems with guaranteed minimal boundary effects. In what follows we outline the ideas that currently go into this calculation

While a long time ago it was believed that there were only a relatively small number of such objects (for example, protons, neutrons,electrons, photons and so on) it is now known that there is a virtual alphabet soup of so called elementary particles. A partial listing of these in terms of standardized short hand description is: $\pi^{\pm}$, π^{0}, η, $K^{\pm}$, K^{0}, $\bar{K}^{0}$, $D^{\pm}$, D^{0}, $\bar{D}^{0}$, $F^{\pm}$, P, N, Λ, $\Sigma^{\pm}$, Ξ^{0} Ξ^{-} Ω^{-} γ ν_{e}, $e^{\pm}$, ν_{μ}, $\mu^{\pm}$, ν_{τ}, τ
We emphasize that this list is but a fraction of the particles observed to date. fortunately, the properties of these particles suggest a pattern consistent with them in turn being made out of a "small" number of more elementary objects called quarks. To date, despite many attempts, there are no reliable reports of an isolated quark actually being observed.

Clearly, a theory is needed that explains the rich particle spectrum in terms of quarks and yet is compatible with quarks being unobservable if isolated from other matter. Further, from past experience with mathematical formulations, it is natural to insist that this description be reasonably simple and elegant. There is exactly one existing candidate for such a description. It is called Quantum Chromodynamics or Q.C.D. It is based on the very successful quantized description of the electromagnetic field interacting with electrons or Q.E.D. Q.C.D. is more complicated than Q.E.D. because the several species of quarks needed to explain the group structure of the observed particles as well as the confinement of single quarks allows for a very rich mathematical structure. This structure is carried in a partition function like object which is the exponential of an action made of qlue fields (designated by the symbol A and quark fields designated by the symbol ψ. Here we have suppressed the space time dependence of these fields as well as the fact that each symbol is actually a vector with at least 12 components. The interaction described by the action is highly non-linear but any term contains either zero or two quark fields which somewhat simplifies the formulation. The primary content of the assumption that system examined be a quantum field theory is that at any given time every point in space has assigned to it independent quantized degrees of freedom associated with the qlue and quark fields. It is thus very natural to describe space time mathematically as a discrete lattice of points with separation a that approaches zero.



The object U(i,j) defined as

$$U(i,j) = e^{i g_0 A \cdot (\vec{i} - \vec{j})}$$

plays a primary role in this theory. It has the property that $U(i,j) = U^{\dagger}(j,i)$. Further the $U(i,j)$ are members of the qroup of unitary unimodular matrices SU(3). For these fields alone we have

the (effective) partition function

$$W = e^{S_g}$$

Here $$S_g = \frac{1}{g_c^2} \sum S_\square$$

where the sum is taken over all independent square plaquettes and

$$S_\square = tr[U(i,j)U(j,k)U(k,l)U(l,i)]$$

We could stop with this form for the partition function and have more work to do than current available machine power will allow. However, to calculate the elementary particle spectrum (except for glueballs) we must include the quark fields in our action. The form used because of various symmetry and guage principles is

$$S_f = -\sum_i \bar{\psi}(i)\psi(i) + K\sum_{i,j} \bar{\psi}(i)B(i,j)\psi(j)$$

Here K is a numerical parameter. The matrix B depends explicitly on the glue field A (of course leaving out gravity and weak interactions) S_f is then taken to be

$$S = S_f + S_g$$

Physics is obtained by calculating the correlation functions or vacuum expectations of polynomials of the field (quark and glue) of the partition function formed from this action. The general problem that must be confronted is the evaluation using the appropriate group measure of the following type of integral.

$$\langle P(\psi,\bar{\psi},U)\rangle \propto \int [\Pi\, dU][\Pi\, d\bar{\psi}\, d\bar{\psi}]\, P(\psi,\bar{\psi},U)\, e^{S}$$

This has many variables . Since each U(i,j) is an SU(3) matrix it is specified by 12 numbers. If we study a hypercubic lattice with N points in each space-time direction we are dealing with the order of N**4*12*4 numbers just associated with the glue fields. The quark fields are characterized by (for our discussion) 12 complex numbers at each lattice point. However, this is just the beginning. The quantities are in fact not numbers! They have the property that $\psi(i)\psi(j) = -\bar{\psi}(j)\psi(i)$. This anticommutivity property is essential in order that the quarks describe objects with intrinsic half integral spin. Because the action S is quadratic only in quark fields it is possible (using very natural

definitions) to explicitly perform the integration over quark fields and leave the problem of evaluation of correlation functions expressible entirely in terms of integrals over glue fields. For example, if we examine the correlation function of four quark fields we have $<\bar{\psi}(a)\psi(b)\bar{\psi}(c)\psi(d)>=$

$$-\int[\Pi dU]\left[(1-KB)^{-1}_{bc}(1-KB)^{-1}_{ad}\det(1-KB)e^{S_g}\right]$$

Note that 1-KB is a (N**4*12)**2 complex matrix. Det(1-KB) is more or less unspeakable for any reasonable size of N. Evaluation of the correlation function above is essential for determining meson masses (such as the pion) in this theory. Calculation of correlations of expectations of six quark fields is needed to evaluate properties of baryon fields (such as the proton). As a practical matter, numerical evaluation of six quark correlations is not much more difficult than four quark correlations. Clearly as N gets larger the problem gets more complicated. However, we are really only interested in the limit when N is very large since this corresponds to the infinite physical world. Indeed, we want to examine the limit were N becomes infinite and the lattice spacing a approaches zero. Under some circumstances it can be argued that neglecting the determinant should not make dramatic changes is the nature of the physical answers we obtain. For this discussion (and the particular project it is outlining) we chose to set the determinant to unity. We are then left with a class of integrals to evaluate which can be handled using Monte Carlo importance sampling methods in conceivable amounts of time for reasonably big lattices. Such systems have been studied extensively using Vax (780) computers on lattices with 6**3*14 points. Using the C.S.U. Cyber 205 it is possible to examine far larger systems. Indeed we are in the process of examining (on several class 6 computers) systems with 10**3*24, 12**3*32 and 20**3*50 lattice sites.

After neglecting the determinant we are left with the basic structure

$$<\bar{\psi}(a)\psi(b)\bar{\psi}(c)\psi(d)>\sim -\int[\Pi dU]\left[(1-KB)^{-1}_{bc}(1+KB)^{-1}_{ad}e^{S_g}\right]$$

We evaluate this numerically in two steps. First, we define a probability

$$dP(U)\equiv\frac{1}{Z}e^{S_g}\,dU$$

Using Monte Carlo (Metropolis) methods we generate a sequence of glue configurations which are are distributed according to $dP(u)$. It is important that these distributions be thermalized and "statistically independent". By careful tuning of the way the Monte Carlo hits are made taking into consideration the nature of the group measure we can enormously speed up the decorrelation of consecutive lattice configurations. Indeed for most cases, it is not difficult to obtain a factor of four increase in speed of lattice generation over conventional methods through careful tuning. Even careful tuning of the physics of this problem does not give reasonable run times for large lattices unless full advantage is taken of the possibility of vectorizing the code. To do this efficiently we use red black methods of sweeping through lattice configurations. In addition, the memory requirements for large lattices rapidly become excessive so we use time slicing to control our memory allocations. We must do this since the demand paging algorithm on the 205 does not work efficiently with the codes which are naturally written for this problem.

After a collection of independent lattices are generated we continue to evaluate the basic integral for the problem by evaluating the inverse of 1-KB for the guage configurations of each lattice. This is somewhat simplified since this inverse need be evaluated for only one base site-that is a fixed row of the matrix. However, it turns out that this inversion must be carried out for three or four different values of the parameter K. The method that has been most commonly used to invert the matrix employs a Gauss Seidel method. This is slow, taking almost an order of magnitude more time than the lattice generation. We have other methods under study which for the particular systems involved promise to be much faster. The Gauss Seidel method is used in a form first applied to this problem by Weingarten. We need to evaluate the form

$$f = (1 - KB)^{-1} h$$

Here h is at a fixed lattice point but can vary through the 12 values associated with the indices of the quark field at that point. This equation is now re-written in the form

$$f = h + KBf$$

$$\Rightarrow f = h + KBf + (1-\lambda)(f - h - KBf)$$

$$= (1 - \lambda + \lambda KB) f + \lambda h$$

λ is a parameter which can be tuned in order to obtain the

fastest convergence in the solution of this equation by iteration in f. In practice we code this procedure using red black ordering and time slicing to obtain vectorization and efficient memory management.

After the matrix inversion is performed and the correlations are evaluated through weighting over the available lattices we must extract physical information from the output functions. The easiest information obtained is the masses of the particles described by this formalism. It is ,for example, a general property of the theory that we are dealing with that if we look at correlation functions depending on only two space time points and then sum over all spatial directions that the resulting time dependent functions depend only on sums of exponentials with the exponent linear in the masses of the appropriate particles and the time separations. It is an easy matter to fit to exponentials and extract numerical values for the masses. However to do this we must tune the parameters of the theory to match the physical mass spectrum at some value of the mass. In effect we have a two parameter fit for the entire mass spectrum. It is found however that the Gauss Seidel method fails to converge for the physical value of the pion mass and hence the need to do the extrapolation in K mentioned earlier. After this is done, it has been found that on smaller lattices a fairly accurate fit can be obtained to the relatively light particles. We expect to find much better fits for a large lattices where edge effects should ha ve a smaller effect on the calculated results.

REFERENCES

Many workers have contributed to this new field. Without any attempt to be complete we list some references which will give the the interested reader a starting point for study of the topics discussed in this talk.

1. K. Wilson, Phys. Rev. D10, 2445 (1974)
2. M. Creutz, L. Jacobs and C. Rebbi, Phys. Rev. D20, 1915 (1979)
3. M. Creutz, Phys. Rev. D21, 2308 (1980)
4. D. Weingarten, Nucl. Phys. B215¢fs7*, 1 (1983).
5. F. Fucito, E. Marianari, G. Parisi, and C. Rebbi, Nucl. Phys. B180, 369 (1981).
6. W. Duffy, G. Guralnik, and D. Weingarten, Phys. Lett 125B,311, (1983)

VECTORIZED MULTIGRID POISSON SOLVER FOR THE CDC CYBER 205

DAVID BARKAI
AND
MAYNARD A. BRANDT

CONTROL DATA CORPORATION
INSTITUTE FOR COMPUTATIONAL STUDIES
AT COLORADO STATE UNIVERSITY

FORT COLLINS, COLORADO

VECTORIZED MULTIGRID POISSON SOLVER FOR THE CDC CYBER 205*

D. Barkai**, A. Brandt***
**Control Data Corporation, Institute for Computational Studies at Colorado State University, PO Box 1852, Fort Collins, Colorado 80522; ***Weizmann Institute, Department of Applied Mathematics, Rehovot, Israel 76100.

ABSTRACT

The full multigrid (FMG) method is applied to the two dimensional Poisson equation with Dirichlet boundary conditions. This has been chosen as a relatively simple test case for examining the efficiency of fully vectorizing of the multigrid method. Data structure and programming considerations and techniques are discussed, accompanied by performance details.

April 1983

1. INTRODUCTION

The multigrid (MG) method has been shown to be a very efficient solver for discretized PDE boundary-valve problems on serial (scalar) computers. However, it was not clear how well can the MG approach be adapted to execute effectively and efficiently on a vector processor, such as the CDC CYBER 205, where considerations other than operations-count may play an important role. The purpose of this paper is to report our experience in implementing an MG code on the CDC CYBER 205. More specifically, the test-case considered is the two-dimensional Poisson equation with Dirichlet boundary conditions. It will be assumed here that the reader has some familiarity with the philosophy, the motivation and the basic computational processes of MG as a fast solver. These processes are described in detail in a number of papers in these proceedings and [1] and [2] and references therein. The algorithm described in this paper is basically the same as the one given in the appendix of [3], whose description is detailed in sections 8.1 and 6.4 of [3]. Therefore, no full description of the MG algorithm is given here, but the relevant details are included in the appropriate context. The main emphasis of this paper is the vectorization of these processes. Thus, we will not assume an in-depth knowledge or experience in applying MG solvers on a vector-processor type of a computer system.

---

* Presented at the International Multigrid Conference, Copper Mountain, Colorado, April 6-8, 1983.

Consequently, Section 2 contains a brief summary of architectural and conceptual features of a vector processor (specific to the CDC CYBER 205), which are relevant to this application, as well as software tools available for a tight correlation between the hardware and the computational process. Sections 3, 4 and 5 are devoted to the description of the techniques used for vectorizing the procedures for the relaxation, the residual transfer calculation and the interpolation, respectively. The total full multigrid (FMG) process and various parameters and constraints are described in Section 6 interleaved with convergence and timings (performance) details. Finally, Section 7 contains some concluding remarks and comments regarding future plans.

2. VECTOR PROCESSING

The most significant difference between a traditional, serial computer and a vector processor is the ability of the latter to produce a whole array ("vector") of results upon issuing a single hardware instruction. The input to such a vector-instruction may be one or two vectors, one or two elements ("scalars"), or a combination of the above. The instructions fall into two main categories—those that perform floating-point arithmetic (including square root, sum, dot-product, etc., as well as the basic operations), and those which may be collectively called "data-motion" instructions. These may be used, for example, to "gather" elements from one array into another using an arbitrary "index-list"; to "compress" or "expand" an array; to "merge" two arrays into one (with arbitrary "interleaving" patterns), etc.

The need for vector data-motion instructions becomes apparent when one considers the definition of a vector on a CDC CYBER 205. A vector is a set (array) of elements occupying consecutive locations in memory. It means, by the way, that a vector may be represented in FORTRAN by a multi-dimensional array; i.e., a two- or three-dimensional array may be used in computations as a single vector. The reason for this vector definition is that when performing vector operations on the CDC CYBER 205 the input elements are streamed directly from memory to the vector pipes and the output is streamed directly back into memory without any intermediate registers.

The timing formula for completing a vector instruction contains two components. One is fixed, i.e., independent of the number of elements to be computed, and is called "start-up" time. In fact, it amounts to start-up and shut-down; it involves fetching the pointers to the input and output streams, aligning the arrays so as to eliminate bank conflicts and getting the first pair of operands to the functional unit (the pipe-line) and the last one back to memory. Typical time for the "start-up" component is 1 microsecond, or about 50 cycles (clock periods). The other component of the timing formula is the "stream-time" which is proporational to the number of elements in the vector. The result rate for a 2-pipe CDC CYBER 205 for an add or multiply is 2 results per cycle. It is apparent now that in order to offset the "wasted" cycles of start-up times it is beneficial to work with longer vectors. The system is better utilized if a single operation is performed on a long vector, rather than several operations to compute the same number of results. Given a vector length, N, one can evaluate the efficiency of the computation as the ratio between the number of cycles used to compute results and the total number of cycles the instruction has taken; i.e., $(N/2)/(N/2 + 50)$. The maximum vector length

the CDC CYBER 205 hardware allows is 65,535 elements. The start-up time becomes quite negligible long before that.

The vector "arguments" for vector instructions are inserted through a construct called Descriptor. It is a quantity occupying 64 bits which fully describes a vector through two integer values: one is the virtual address of the starting location of the vector, the other is the number of elements, or the length, of the vector. An element may be a bit, a byte, a half-word (32-bits) or a word (64-bits) depending on the intruction and the argument within the instruction. The CDC CYBER 205 FORTRAN provides the ability to declare variables of "type" Descriptor and Bit, as well as, extensions for assigning Descriptors to arrays and syntax for coding vector instructions without such an explicit association. Bit arrays occupy exactly one bit per element, since the CDC CYBER 205 is bit-addressable. Bit vectors are used for creating a "mapping" between an array containing numerical values and a subset of it. A Bit vector may be used to control a vector floating-point operation (hence the term "control-vector" which is commonly used for a Bit vector) as follows: Take, for example, an add operation. All the elements of the two input arrays are added up, but only those result elements where the corresponding element of the control-vector is 1 will be stored into the results vector. The other elements will not be modified. Alternatively, one may specify storing on zeros in the control-vector, and discarding results corresponding to a 1.

Another common use of bit vectors is associated with some of the data-motion instructions. Two examples will be given here: The "compress" instruction is used to create a vector which is a subset of another vector. This operation has two input descriptors—one points to a numeric vector, the other to a bit vector. Whenever a 1 is encountered in the bit-vector the corresponding numeric element is moved to the next location of the output vector, i.e., the input array is "compressed" (the reverse process may be accomplished with an "expand" instruction). A single bit-vector may also be used to "merge" two numeric vectors into one. The bit-vector is scanned and when a 1 is encountered the next element of the first input vector is put into the next location of the output vector, when a zero is found in the bit-vector the next element of the second input vector is moved into the next location of the output vector. The timing for both these instructions is dictated by the total length of the bit-vector. The result-rate is the same as that of vector arithmetic, i.e., on a two-pipe CYBER 205 it is two elemets per cycle (whether they are moved or not). It will be noted here that there are vector instructions for creating repeated bit patterns at a rate of 16 bits per cycle.

Before concluding this section let us briefly mention the existence of an "average" instruction, which computes an average of two vectors, or adjacent means of a single vector, at the rate of a single floating-point operation. One can also "link", for example, an add and a multiply operation, provided at least one of the three inputs is a "scalar", and perform the two operations as if it were only one. All the instructions mentioned above are directly available through Fortran in-line function calls.

3. RELAXATION

Now we are ready to examine the ways in which to utilize the tools and the vector processing concepts discussed in the previous section for vectorizing the Multigrid application. The success of such an exercise

hinges, to a large extent, upon the efficiency with which the relaxation process may be accomplished.

Discretization of the two-dimensational Poisson equation is achieved via the 5-points differencing scheme. Thus, assuming geometric interpretation of the indices for the moment, the set of the simultaneous equations to be solved may be written as

$$u_{i,j-1} + u_{i-1,j} + u_{i+1,j} + u_{i,j+1} - 4 * u_{i,j} = h^2 F_{i,j}$$

where u is the unknown function, h is the interval between two grid points (in either direction) and F is the right-hand side function. i varies from 2 to N_1-1 and j from 2 to N_2-1, where N_1 and N_2 are the number of grid points along the two directions.

One may want to consider the usual (lexicographic) Gauss-Seidel relaxation procedure. This, however, will be in conflict with vectorization, as may be easily deduced. The Gauss-Seidel relaxation is characterized by the use of updated values as soon as they become available. Vectorization means processing many such values in parallel, i.e., not waiting for the previous element to be updated. The obvious alternative is the red-black or checker-board ordering, where all the four neighbors of each point belong to the other "color". The convention used here is that the "color" of the grid points at the corners of the rectangle is red. The grid may accordingly be divided into two vectors and the relaxation performed in two stages: first, the values at red points are updated using "old" values, then the values at black points are updated using the "new" red values. Throughout the code the two vectors of the unknown function (and of the RHS function) are stored consecutively following each other, where inside each vector the values are stored column-wise as shown in Figure 1. This storage applies, of course, to all the grids used.

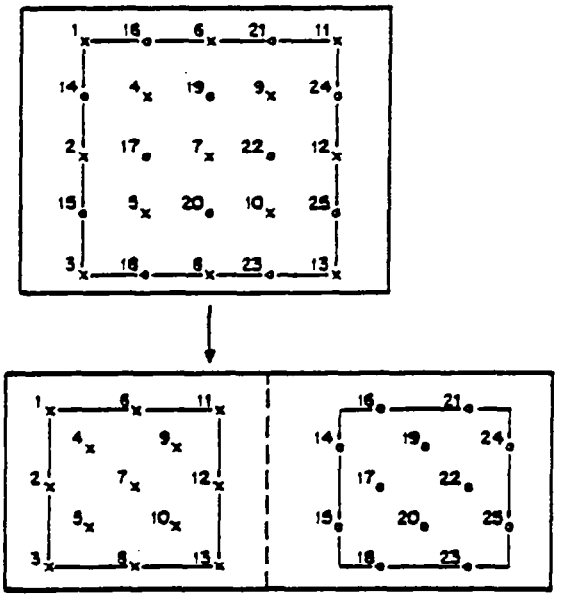


Figure 1. Mapping of the Lexicographic into the "Red-Black" Ordering. The dotted line indicates the separations of the grid points into two vectors.

The reader will notice that the vectors thus created are not confined to one column, but extend over the entire grid. It was done in order to achieve longer vectors in line with the desire expressed in Section 2. This, however, introduces the hazard of overwriting values residing on the boundary of the grid. To avoid this a bit control-vector was created for each grid, in a set-up routine, which contains zeros where boundary points exist and ones for interior points. We use this "boundary control vector" to assure storing new values only into the interior of the grid.

The computation requires the sum of the 4 neighbors for each grid point. One can easily verify that, using vector add operations this can be done with two operations only. One to add a vector into itself, with some offset (e.g., start with elements 2 and 5 in Figure 1) and the second to add the resultant vector into itself (with some other appropriate offset). The remaining calculation involves subtracting the result from the RHS values and multiply by a constant (being -0.25), which is accomplished as a linked-triad operation; the result is then stored into place under the control of the boundary bit-vector. Thus, each of the two stages (two "colors") requires three floating-point operations using vector length of, approximately, $(N_1 * N_2)/2$ elements long. In fact, some more savings in the computations occur in the first relaxation sweep after moving to a coarser grid, since the sum of the "neighbors" need not be computed for the first "color," being known to be zero. This is because we are beginning to compute a correction-function whose first approximation is zero. The vector-operations count for this relaxation sweep is thus reduced from 6 to 4. Also, when transferring a solution-function (not "correction") to a finer grid, as part of the FMG process, an interpolation can be used which will save the relaxation on the first "color" (see Sec. 5).

In conclusion, the relaxation process can obviously be done extremely fast on the CYBER 205. Timing details will be given in Section 6.

4. FINE TO COARSE RESIDUAL TRANSFER

Residuals have to be computed at those fine-grid points which also belong to the coarser grid. These residuals are directly transferred to the corresponding coarse-grid points weighted by 1/2 ("half injection"; the factor of 1/2 is motivated by the fact that the fine-grid residual is zero at black fine-grid points, hence the other residuals should be multiplied by 1/2 to represent the correct average). See Figure 2.

The computation involves four floating-point operations (two of them are linked triads) for evaluating the residuals of the red points on the finer-grid and multiplying them by 1/2. This, however, does not conclude the procedure. At this stage we need to apply the "compress" operation three times as follows: using a pre-defined bit-vector we extract the residual values corresponding to coarse-grid points, i.e, belonging to odd-numbered columns of the red section of the finer grid. (Note that we have thrown away half the calculated residuals. This procedure is both simpler and a little faster than having to perform all the compress operations needed for computing only the required residuals.) Now, as is evident from Figure 2, we have all the desired values for the coarser grid stored in lexicographic order. To separate them into "red" and "black" sections the "compress" instruction is applied twice (once for each color) using a pre-defined "picket fence" bit-vector. The procedure as described here produces optimum performance even though some redundant operations are

performed. The alternatives are to perform different (more "costly") data motions or to operate on much shorter vectors. Finally, another vector operation is executed to zero out the unknown function of the coarser grid in preparation for evaluating the correction function. In total the procedure requires 8 vector "start-ups" associated with 5 operations of approximate length of $(N_1 * N_2)/2$, and 3 operations of length $(N_1 * N_2)/4$, where N_1 and N_2 are the dimensions of the finer grid.

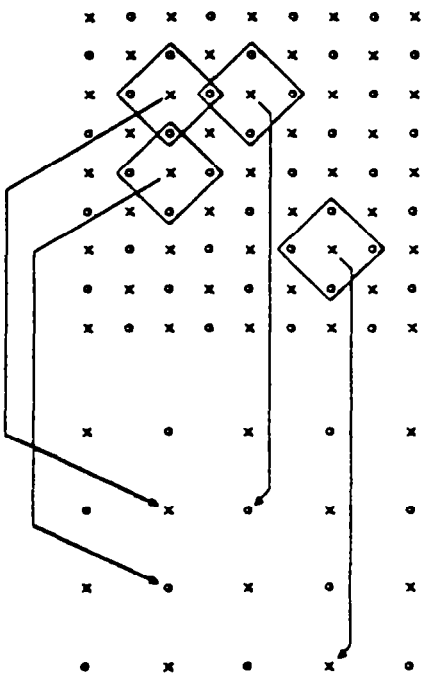


Figure 2. Transfer to a Coarser Grid: The residual calculation. Each "Box" contains the fine grid points involved in the computation for the corresponding coarse grid point.

5. INTERPOLATION

Interpolation, in the context of this paper, is the process by which we transfer from a given grid to a finer one. Two types of interpolations are employed here: Type I interpolation is used when a correction is interpolated from the coarser grid and added to the finer grid. The Type II interpolation is used to compute a first approximation on the finer grid, based on existing values on the coarser grid. The use of the red-black ordering, combined with the fact that a relaxation always follows an interpolation, implies that only one color of the finer-grid points need to be interpolated (the other color will be computed by a relaxation pass on that color).

Type I interpolation is bilinear employing points as shown in Figure 3. Only interior black points on the finer grid need to be evaluated. Due to the required averaging of the coarse grid values it is convenient to first merge the red and black points of this grid using the "picket-fence" bit vector to produce the lexicographic ordering. Next, two averages are

computed. The average over the coarse grid, where the two input vectors are offset by a column, will produce the quantities to be added into black points on even-numbered columns on the fine grid. A second average, where the offset between the two vectors is one element, is executed for fine grid black points corresponding to odd numbered columns. This last operation produces redundant values (at the end of each coarse grid column) which are thrown away using the "compress" operation with an appropriate pre-defined bit vector. The two resultant coarse grid "average-vectors" are then interleaved, using a "merge" instruction, under the control of the bit vector where the "1's" and "0's" correspond to odd and even columns, respectively. Finally, the merged values are added to "black" points of the finer grid under the control of the "boundary" bit-vector which inhibits storing values into the boundary of the grid. The whole procedure amounts to 3 floating-point operations, 2 "merges" and 1 "compress." The 6 vector operations may also be divided into 4 operations of length $(N_1 * N_2)/4$ and 2 operations of length $(N_1 * N_2)/2$, approximately. (N_1 and N_2 are the dimensions of the finer grid.)

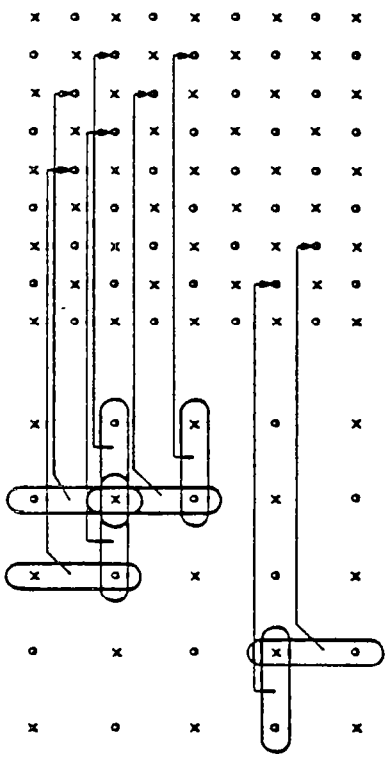


Figure 3. Type I Interpolation. It shows where averages of coarse grid values are added into "Black" points on the fine grid.

Type II interpolation is a 4th order one, described, for example, in section 6.4 of [3]. It produces new red unknown-function values on a finer grid using rotated difference operators. The values at the black points are produced by half a relaxation sweep, i.e., a relaxation pass over the fine-grid black points. (This pass may be regarded as part of the interpolation process. In the timing tables below, however, the time spent in this pass is counted as relaxation time.) The process is described pictorially in Figure 4. All the interior coarse grid values are moved to occupy

the corresponding fine-grid points. The relaxation operator is applied to these values in order to compute interior red points of the even-numbered columns on the fine grid. The only difference between the relaxation here and the one described in Section 3 is that the operator is the "rotated" 5-point Laplacian and the interval between each point and its neighbors is changed from h to $\sqrt{2}$*h. The RHS function values required for this relaxation are available from the fine grid RHS array (a "compress" operation is performed to retrieve even-numbered column values). The whole procedure, thus, requires 2 "merges" (one for merging red-black values of the coarse grid, the other for merging the "transferred" and "relaxed" values of the red fine grid points); 3 floating-point operations for the relaxation; 2 "compress" operations (one for throwing away redundant, incorrect averages and one for collecting RHS values); and, finally, one vector-move operation under the control of the boundary bit-vector for storing the new red fine grid values into place. Five out of the 8 vector operations have length of about $(N_1 * N_2)/4$, the other 3 are associated with a length of $(N_1 * N_2)/2$; N_1 and N_2 being the dimensions of the finer grid.

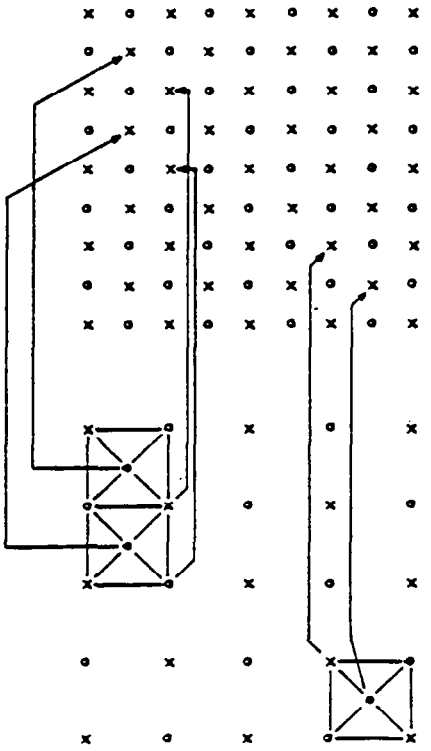


Figure 4. Type II Interpolation. Coarse grid values are transferred to odd numbered columns on the fine grid. These values are used to compute, via the relaxation operator, the even numbered column values.

6. PERFORMANCE AND CONVERGENCE

The basic computational procedures, studied in the previous three sections, can now be linked together to form the FMG process. Figure 5 is a schematic description of the sequence of events which leads to an approximate solution of the difference equations. The finest grid (where a solution is sought) is assigned the highest level number. The example

depicted in Figure 5 describes an FMG with 5 levels where the process starts at level number 2. This may not be necessary, as will be argued below, and one may visualize the FMG starting at a higher level simply by deleting the left-hand-side of the figure. This starting level is a parameter controlled by the user. The FMG shown in Figure 5 is composed of what is known as "V" cycles. In each "V" cycle one performs relaxation-residual calculation-relaxation...until reaching the coarsest grid, then a sequence of interpolation-relaxation is executed. The transfer from one "V" cycle to the next is achieved via Type II interpolation. More specifically, the FMG we implemented may be characterized as FMG (M,L,R1,R2,R3,R4), where M is the number of levels and L is the starting level; R1 and R2 indicate the number of relaxations before moving to a coarser grid and before moving to a finer grid, respectively. R3 and R4 have the same meaning and apply to the last "V" cycle only. All these parameters are provided by the user. The user may also specify the size of the coarsest grid to be used. It must have an even number of intervals in each direction. (In our experiments the coarsest grid had 3 by 3 points; i.e., 2 by 2 intervals.) The user also specifies the mesh size h (assumed to be the same in both directions) on the finest grid.

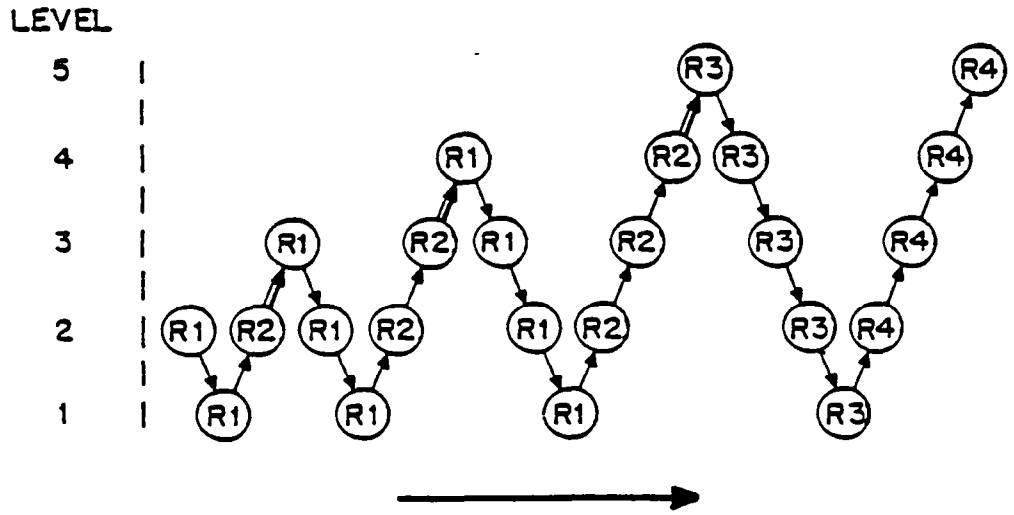


Figure 5. The Full Multigrid (FMG) Process: FMG (5, 2, R1, R2, R3, R4). The circles indicate the number of relaxations performed at a given level. Downwards arrow signifies residual calculation between relaxations, upwards arrow implies interpolation. (When a level is encountered for the first time the interpolation is of Type II, indicated by a double line above, otherwise it is of Type I.) When level 1 contains only one interior point only one relaxation sweep is performed thereon, regardless of the values given to R1 and R3.

The process described above is deterministic, in the sense that the user defines the steps to be taken, based on prior knowledge of the characteristics and smoothness of the function to be solved. It is also known that if L=2 the FMG guarantees a solution error smaller than the truncation error (introduced by the differencing scheme), for L_2 norm, for example. We have allowed, however, as a user-option, the evaluation of

the L_1, L_2 and L_∞ norms of the residual at various points. Testing was done for problems which have solution of the form:

$$C * \cos(k\ (x + 2y))$$

with and without the addition of a 6th degree polynomial which vanishes on the boundary. In all these cases the FMG process with L=2 indeed produced a solution with an algebraic error (error in solving the difference equations) much smaller than the truncation error, in the L_1, L_2 and L_∞ norms.

Only "V(2,1)" cycles were used for the results and timings to be quoted here. This turns out to be the optimum combination for the Poisson equation. More relaxations at each stage do not improve the final result enough to justify the additional work, less relaxations may cause deterioration in the accuracy. (If full weighting were used instead of half injection, the optimal cycle would be "V(1,1)". This would, however, be less efficient than the present procedure since full weighting is substantially more costly than a relaxation sweep.) In the performance details which follow, we will give results for various values of L since, in many cases, in particular when a reasonable initial guess is available, high values of L, even L=M, may provide sufficient accuracy. This is, in particular, the situation when the Poisson solver is used within some external iterative process, or at each time step of an evolution problem.

Before discussing the timings we should briefly mention some set-up procedures. A routine is provided for re-ordering the initial array (from lexicographic to red-black) if it is not so structured yet. This is done through two "picket-fence compress" operations and amounts to 0.185 msecs. for a 65 by 65 grid, for example. Putting the solution back into lexicographic order is done with a single "merge" instruction and takes half as long. Next, there is a routine which defines various pointers and lengths for all the grids used, as well as the bit-vectors discussed earlier. For many applications, where the solver is used many times with the same grid definition, this will be done only once. It will not, therefore, be included in the total times quoted below (it takes 0.29 msecs. for a 65 by 65 grid with 6 levels). The last set-up routine is included in the timings information. This routine defines the boundary values and the RHS for all the levels between L and M-1. It also sets the initial guess on the level L grid.

The code was run with grid sizes of 33 by 33, 65 by 65 and 129 by 129 (M = 5, 6 and 7, respectively) with L=2,...,M. Total execution times are given in Table 1. It shows, for example, that a 65 by 65 grid may be solved in as little as 1 msec., and, at most, in 2 msecs. By examining the processing time per grid-point one can see the effect of vector-instructions start-up times or the dependence of the performance upon vector lengths. On a serial processor the time per element would have been, approximately, a constant across each line in Table 1. We observe, however, that the processing of the 129 by 129 grid is roughly twice as efficient as that of the 33 by 33 grid. This is due to the fact that even though the number of vector "start-ups" remains nearly the same (across a given line), the number of elements solved for has increased by a factor of 16. Hence, more time is spent doing useful arithmetic in the vector pipelines.

TABLE 1. Execution times for various parameters of the FMG. The entries on the left are total times in milliseconds. The entries enclosed in parenthesis are the execution times in microseconds per grid-point (only interior points are taken into account).

M-L+1 (No. of "V"'s)	33 by 33 (M = 5)	65 by 65 (M = 6)	129 by 129 (M = 7)
1	0.360 (0.37)	1.006 (0.25)	3.293 (0.20)
2	0.604 (0.63)	1.552 (0.39)	4.910 (0.30)
3	0.729 (0.76)	1.810 (0.46)	5.440 (0.34)
4	0.801 (0.83)	1.947 (0.49)	5.687 (0.35)
5		2.009 (0.51)	5.807 (0.36)
6			5.875 (0.36)

Tables 2 and 3 present a more detailed analysis of timings for a single example, namely for solving a 129 by 129 grid with 7 levels and starting at level 2. The entries in Table 2 show timings in msecs. by level and by procedure. One notices that the total time spent performing relaxations is less than 50% of the total time. This is to be compared against the 80-90% of total time used for relaxations on a serial processor. This is, of course, due to the fact that the vectorized relaxation is extremely efficient and does not involve any data-motion operations. The interpolation and the residual calculations, though fully vectorized, involve some data-motion operations, and, therefore, consume a relatively higher proportion of the execution time than they would on a "scalar" computer. Another observation worth mentioning is that the contributions to all the procedures arising from levels 2 to 4 is roughly the same, even though the amount of work differs by a factor of 4 between levels. This is a consequence of the relatively short vectors which characterize the coarser grids. It also explains the larger weight the coarse grids have in the vectorized code compared to that of the serial process.

TABLE 2. Execution times in milliseconds for solving a 129 by 129 grid with starting level 2. Breakdown by procedure and by level. For the residual calculation and the interpolations the entry in the table corresponds to the finer grid involved.

Level	Grid Initialization	Relaxation	Residual Calculation	Interpolation Type I	Interpolation Type II	Total
1 (3x3)		0.010				0.010
2 (5x5)	0.011	0.179	0.014	0.011		0.215
3 (9x9)	0.015	0.160	0.060	0.049	0.024	0.308
4 (17x17)	0.034	0.189	0.068	0.053	0.028	0.372
5 (33x33)	0.106	0.320	0.117	0.095	0.053	0.691
6 (65x65)	0.388	0.690	0.261	0.194	0.141	1.674
7 (129x129)		1.257	0.497	0.357	0.494	2.605
TOTAL	0.554	2.805	1.017	0.759	0.740	5.875

In Table 3 we have measured the time in microseconds for each time a procedure is executed for a given level, accompanied by the number of times the procedure is performed. It should be noted here that when level 1 is involved in any of the procedures a scalar code was used, since it has only one interior point. Again, the effect of vector lengths is such that the level 3 relaxation is comparable to that of level 2, for example. Only when we get to the finest grids do we observe timing ratios which correspond to the ratios of the number of elements processed. The reader should be reminded that the average time of the relaxation procedure is not fully accurate, since some relaxations are not quite "complete" as was explained in Section 3 (i.e., after Type II interpolation and after residual calculation). The residual calculation takes longer than the relaxation (in contrast to the scalar case), which is understandable from the discussion in Sections 3 and 4.

TABLE 3. Procedure-calls count and average times in microseconds per call. Breakdown by levels for the 129 by 129 problem with starting level 2.

Note: Some of the relaxations are not "complete." (See Section 3)

					Interpolation			
	Relaxation		Residual		Type I		Type II	
Level	No.	Time	No.	Time	No.	Time	No.	Time
1 (3x3)	6	1.7						
2 (5x5)	18	9.9	6	2.3	6	1.8		
3 (9x9)	15	10.7	5	12.0	5	9.8	1	24.0
4 (17x17)	12	15.8	4	17.0	4	13.3	1	28.0
5 (33x33)	9	35.6	3	39.0	3	31.7	1	53.0
6 (65x65)	6	115.0	2	130.5	2	97.0	1	141.0
7 (129x129)	3	419.0	1	497.0	1	357.0	1	494.0

To conclude the performance discussion we will mention that the vectorized code executes about 15 times faster than the scalar version on the CDC CYBER 205, and roughly 500 times faster than the CDC CYBER 720.

The lesson from what was said above is that relaxations are relatively "cheap" in terms of execution times, and computations on the coarser grids are realtively "costly" (compared with the ratios found on scalar processors).

7. CONCLUDING REMARKS

One important lesson, known very well to those involved in vector processing, is that it demands careful data structuring and analysis of the "mapping" between the data and the operations to be performed, if the vector capabilities of the processor are to be efficiently utilized. We have also demonstrated that the traditional operations-count as a measure of processing time is not sufficient. On a vector processor one has to take into account the number of vector operations (or the lengths of the vectors) and the data-motion operations (which occur on a serial processor, too, but are often ignored when algorithms are evaluated). The result of the above is that one may have to re-examine the various parameters of the algorithm when migrating the Multigrid application from a serial to a vector processor. This aspect requires further investigation.

We feel that the experiment with the model-case studied in this paper was successful and the performance achieved very pleasing. It certainly warrants continuation work. Some obvious areas we intend to engage in are the following: Extending the application to three-dimensional Poisson equations; code a similar application to cater for the, more general, Diffusion equation; and implement "full-weighting" residual calculation and cubic interpolation. In addition one may, of course, generalize this work in many directions. More general boundary conditions (Neumann, etc.) can be implemented. The solution of non-linear problems (using FAS multigrid version) and systems of equations can also be vectorized in a similar fashion. More difficult, but potentially important, is the extension to general domains, which will require a lot of thought about data structures and data motion. As a last comment, it will be noted that all the timings quoted here were achieved using 64-bit arithmetic. On the CDC CYBER 205 one can use 32-bit arithmetic as well, and, thus, double the result rate for vector operations while halving the memory requirements. For the purpose of obtaining albebraic errors smaller than truncation errors in solving second order equations, the 32-bit arithmetic is indeed enough. We intend to examine this option.

REFERENCES

[1.] A. Brandt, "Multi-level adaptive solutions to boundary-value problems", Math. Comp. 31, (1977), 333-390.

[2.] W. Hackbusch and U. Trottenberg, ed., "Multigrid Methods", Proceedings of a Conference (Koln-Porz, Nov. 1981), Springer-Verlag, 1982.

[3.] K. Stuben, K. Trottenberg, "Multigrid Methods: Fundamental algorithms, model problem analysis and applications". In [2] pp. 1-176.

THE VECTORIZATION OF A RAY TRACING PROGRAM FOR IMAGE GENERATION

DAVID J. PLUNKETT,
JOSEPH M. CYCHOSZ
AND
MICHAEL J. BAILEY

PURDUE UNIVERSITY CADLAB

WEST LAFAYETTE, INDIANA

THE VECTORIZATION OF A RAY TRACING PROGRAM FOR IMAGE GENERATION

David J. Plunkett[1]

Joseph M. Cychosz

Michael J. Bailey

Purdue University CADLAB

ABSTRACT

Ray tracing is a widely used method for producing realistic computer-generated images. Ray tracing involves firing an imaginary ray from a view point, through a point on an image plane, into a three dimensional scene. The intersection of the ray with the objects in the scene determines what is visible at that point on the image plane. This process must be repeated many times, once for each point (commonly called a pixel) in the image plane. A typical image contains more than a million pixels making this process computationally expensive. A traditional ray tracing program processes one ray at a time. In such a serial approach, as much as ninety percent of the execution time is spent computing the intersection of a ray with the surfaces in the scene. With the CYBER 205, many rays can be intersected with all the bodies in the scene with a single series of vector operations. Vectorization of this intersection process results in large decreases in computation time.

The CADLAB's interest in ray tracing stems from the need to produce realistic images of mechanical parts. A high quality image of a part during the design process can increase the productivity of the designer by helping him visualize the results of his work. To be useful in the design process, these images must be produced in a reasonable amount of time. This discussion will explain how the ray tracing process was vectorized and gives examples of the images obtained.

1. Authors' Address:
CADLAB, Potter Engineering Center
Purdue University
West Lafayette, IN 47907
(317) 494-5944

GEOMETRIC MODELING AND MECHANICAL DESIGN

In mechanical design, there are two broad reasons for using the computer: (1) predict behavior, and (2) visualize. Behavior that needs to be predicted includes every test that one would normally perform if given a physical prototype of the design: weight, center of gravity, strength, movement, clearances, etc. This is why a computer model of a part is often referred to as a "virtual prototype." Visualization is, in effect, another form of behavior prediction. In this case, knowing the actual appearance of a proposed design is a valuable aid in conceptualizing.

In order to feed information into visualization and analysis routines, a *geometric model* of the design must first be created. In the early days of computer aided engineering, a wireframe database was used to model the part shape. This was deemed inadequate, because the wireframe could only model a part's *edges*, not its *solid volume*.

One of the methods by which we model part shapes in the CADLAB is with a newer technique called *Solid Modeling*. A solid modeling database has sufficient geometric information to completely and unambiguously define the shape of a three dimensional object. One method of building a solid model database is with a technique called *Constructive Solid Geometry*, or CSG. A CSG geometric creation sequence is characterized by applying boolean operators (union, difference, intersection) to groups of primitive shapes (boxes, cylinders, cones, etc). Complex designs may be created in this manner, with the results being sufficient to drive visualization and other analyses. The remainder of this report will discuss the use of the CYBER 205 to produce image information in order to view an object constructed using CSG operations.

INTERSECTIONS OF RAYS WITH A PRIMITIVE

One nice side effect of using a CSG representation is that the resulting object can easily be displayed using ray tracing. Ray tracing involves firing an imaginary ray from a view point, through a point on an image plane, into a three dimensional scene. It is not mathematically feasible to determine the visible surface of an entire *CSG object* in a single computation. However, it is fairly easy to determine the intersection of a ray with each of the ***individual primitives*** which make up a CSG object. Then, a little more calculation produces the point along that ray which is visible. If one ray is fired through every pixel in the image plane, an image of the object is obtained (see Figure 1).

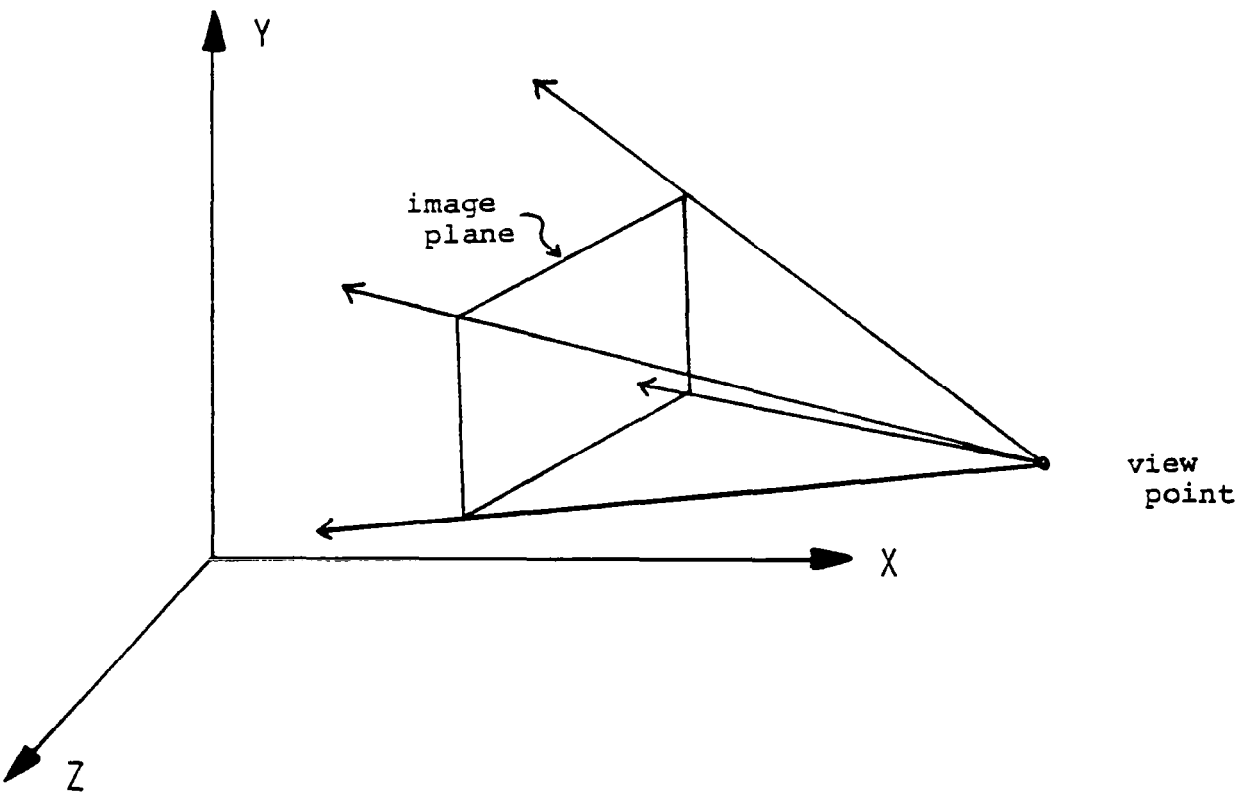


Figure 1. The Image Environment.

The typical (serial) ray tracing program must:

- Intersect all primitives in the scene with one ray.
- Traverse the CSG database to determine which primitive intersection is the visible surface for that ray.
- Determine the surface intensity using the surface relationship between the surface normal, the eye position, and the position(s) of the light source(s).

This is the visible surface algorithm. It is repeated at every picture element (pixel) in the image plane.

The intersection of the ray with the primitives is by far the most time consuming part of the visible surface algorithm. However, it is also the easiest part of the algorithm to vectorize. Instead of just finding the intersection of one ray with a primitive, a queue of rays is built (serially as in a traditional ray tracing program). Then the intersections of each primitive with every ray in the queue is found in a series of vector operations. Table 1 gives computation times for 100,000 rays intersecting a sphere and a cylinder primitive. For the vector results in this table, a queue length of 2000 rays was used.

FINDING A RAY'S VISIBLE SURFACE

The above timings are only for the lowest level in the visible surface algorithm. After all the intersections are found, the CSG database must still be traversed to determine which primitive intersection is the visible surface for that ray. This constrains the length of the ray queue, since it implies that all the ray intersection information must be stored (after the intersection calculation) and then retrieved (for the visible surface calculation). If the ray queue is too long, the time spent page faulting will be enormous. For this reason, the ray queue in our application is

TABLE 1.

CPU Times[2]

Primitive	Cyber 205 Scalar	Cyber 205 Vector	Cyber 720
sphere	.944	.0279	13.1
cylinder	2.729	.1614	51.48
steiner	11.157	1.047	216.0

Speedup[3]

Primitive	$S_{205\ scalar}^{205\ vector}$	$S_{720}^{205\ vector}$
sphere	33.81	469
cylinder	16.91	318
steiner	10.67	206

2 CPU times are in seconds

3 Speedup $= S_{P_2}^{P_1} = \dfrac{\text{CPU time } P_2}{\text{CPU time } P_1}$

approximately 2000 rays. The visible surface algorithm has not yet been vectorized. However, it is apparent that at least parts of this process are vectorizable.

SPECIAL EFFECTS

One of the reasons ray tracing has been so widely accepted is that it can show very realistic image synthesis effects. Shadows are perhaps the easiest extension to the algorithms described above. To determine if a visible surface is in a shadow, one ray must be fired toward each light source from the visible surface. If this ray hits a solid object before it encounters the light source, the visible surface is in a shadow. Reflection can be shown by spawning another ray from each surface such that the angle of reflection equals the angle of incidence. Transparency and refraction can be modeled if a refraction ray is spawned after a hit on a solid, transparent object. What should be clear from these special effects is that the extra rays to be fired do not come in a predictable, vectorizable progression. However, after a serial section of code has determined that another ray must be fired, this ray can be placed in the queue and intersected using vector code when the queue is full.

SURFACE PATCHES

Surface patches are used in computer aided design to sculpt the surface of a part that would be difficult or impossible to model using conventional primitives such as cylinders and boxes. Hence, surface patches play an important role in the design process of parts such as air foils and car bodies. At the CADLAB we are currently investigating the uses of Steiner surfaces as a sculpting device. Ray tracing is then used to visualize the resulting sculpted surface.

A Steiner surface is a bi-quadratic surface. This means that computing the intersection of a ray with a Steiner surface requires the solving of a quartic equation. Approximately 65 precent

of the computation time for this intersection calculation involves the solving of the quartic equation while the rest is attributed to the determination of the coefficients for the quartic equation. The determination of the polynomial coefficients is a straight forward process and is easily vectorized. Vectorizing the process by which a queue of rays may be intersected with a Steiner surface requires the vectorization of the root solver used for solving the quartic. For our application we are only interested in the first positive real root closest to zero. Table 1 shows the results of vectorizing the Steiner intersection process.

To determine the roots of the quartic polynomial the slope and curvature functions (i.e. the first and second derivatives) are examined to determine the intervals over which a possible solution exists. Modified Regula Falsi is then used to determine the roots within these intervals. Once a root is found it is evaluated to see if the root is acceptable.

The vectorized version of the root solver finds the roots of a series of quartic polynomials, each polynomial corresponding to a ray in the ray queue. The roots for all the polynomials must be found before the process can complete. Unlike the scalar version, it is most likely that all four roots will have to be determined and evaluated as it is likely that at least one ray will not intersect the surface. This process is sped up by ensuring that a sign change does not occur before using the Falsi method to determine subsequent roots once an acceptable root has been found for a particular polynomial. Gather-scatters are then used to compress the vectors used during these iterative processes. Convergence occurs when all of the roots being found converge within the specified tolerance.

The quartic root solver can be used for a variety of applications. One extension to the ray tracing program will be the inclusion of tori and other elliptical surfaces as primitives. These primitives will also require solving a fourth order equation to determine the intersection of a ray with their surface.

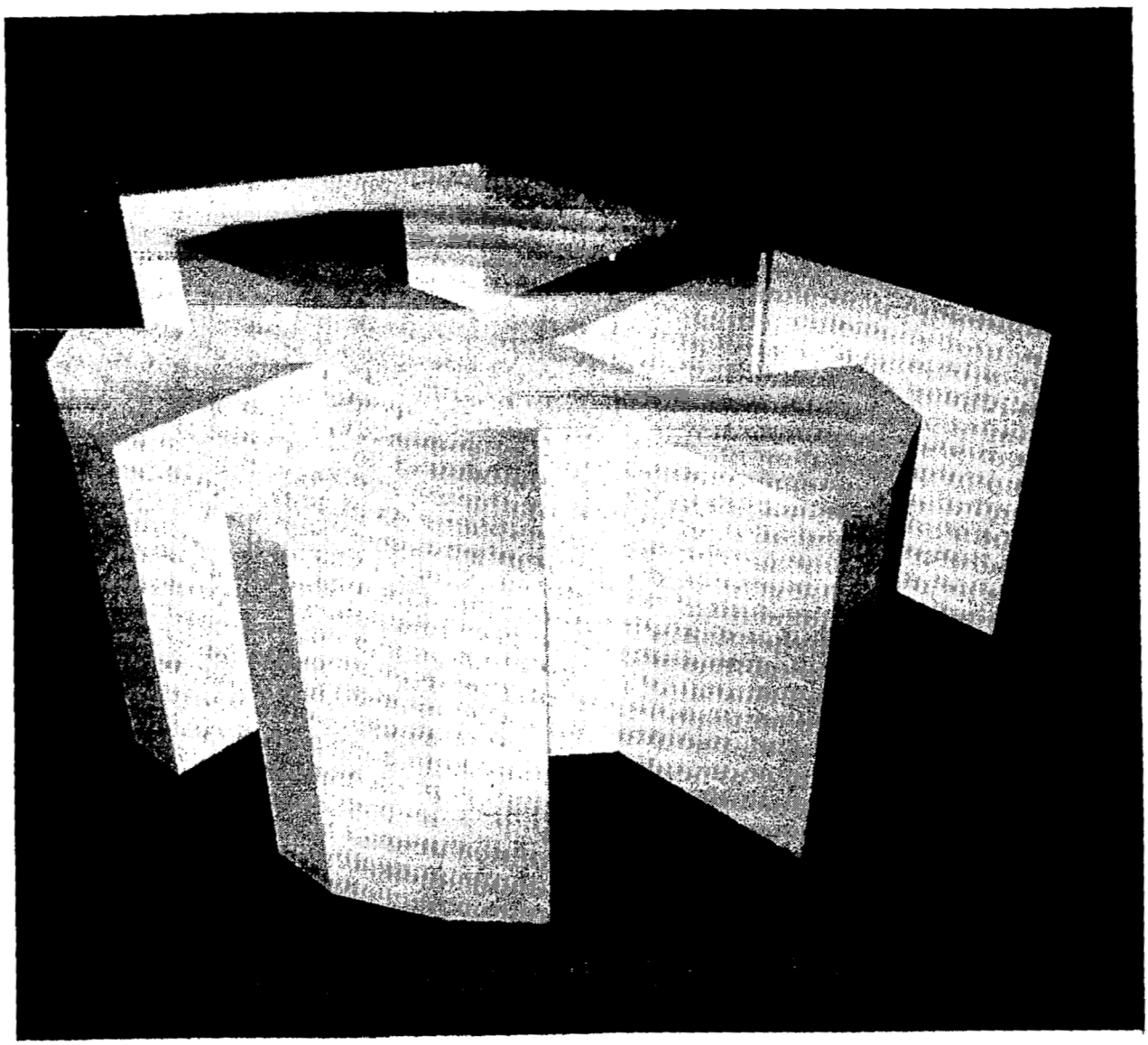
OTHER APPLICATIONS

Another application of ray tracing at Purdue is radiant heat transfer analysis of finned Tubes [MAXW83].4 Rays are fired to determine the radiation shape factor of one or more finned tubes. Unlike the visualization of a CSG object, maximum length vector operations may be used since it is only of interest knowing that the ray strikes the tube and not where on the tube. The computational requirements of this application have been reduced from 600 seconds on a CDC 6600 down to 3 seconds on the CYBER 205.

CONCLUSION

Ray tracing is, in general, a *parallel* algorithm. This paper examined how the parallel algorithm can be modified for use on a vector computer. In design work, the speed with which results are available is often critical. Vectorization of ray tracing programs promises shorter execution times. This will benefit not only visualization, but also such diverse areas as heat transfer, mass properties analysis, and nuclear engineering.

4 [MAXW83] Maxwell, G.M., "Mathematical Modelling of a Gas Fired Swimming Pool Water Heater", Ph.D. Thesis, Purdue University, in preparation.



A KOSLOFF/BASAL METHOD, 3D MIGRATION PROGRAM IMPLEMENTED ON THE CYBER 205 SUPERCOMPUTER

L. D. PYLE
DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF HOUSTON
HOUSTON, TEXAS

S. R. WHEAT
BELL TELEPHONE LABORATORIES
NAPERVILLE, ILLINOIS

Title: A Kosloff/Basal Method, 3D Migration Program Implemented on the CYBER 205 Supercomputer

Authors: L.D. Pyle* and S.R. Wheat**

ABSTRACT:

Conventional finite-difference migration has relied on approximations to the acoustic wave equation which allow energy to propagate only downwards. Although generally reliable, such approaches usually do not yield an accurate migration for geological structures with strong lateral velocity variations or with steeply dipping reflectors. An earlier study by D. Kosloff and E. Baysal (Migration with the Full Acoustic Wave Equation) examined an alternative approach based on the full acoustic wave equation. The 2D, Fourier-type algorithm which was developed was tested by Kosloff and Baysal against synthetic data and against physical model data. The results indicated that such a scheme gives accurate migration for complicated structures. This paper describes the development and testing of a vectorized, 3D migration program for the CYBER 205 using the Kosloff/Baysal method. The program can accept as many as 65,536 zero-offset (stacked) traces. In order to efficiently process a data cube of such magnitude, (65 million data values), data motion aspects of the program employ the CDC supplied subroutine SLICE4, which provides high speed input/output, taking advantage of the efficiency of the system-provided subroutines Q7BUFIN and Q7BUFOUT and of the parallelism achievable by distributing data transfer over four different input/output channels. The results obtained are consistent with those of Kosloff and Baysal. Additional investigations, based upon the work reported in this paper, are in progress.

This research was supported by the Control Data Corporation and the Allied Geophysical Laboratories at the University of Houston.

*Department of Computer Science, Unversity of Houston, Houston, Texas
**Bell Telephone Laboratories, Naperville, Illinois

I INTRODUCTION

1.1 THE KOSLOFF/BAYSAL STUDY

In an attempt to develop a migration technique that did not have the faults of conventional finite-difference migration techniques, Kosloff and Baysal introduced a migration technique based on the full acoustic wave equation [1]. While conventional finite-difference techniques used an approximation to the wave equation, they allowed energy to propagate only downwards. Although these techniques yield reliable migration in most cases, they usually do not yield an accurate migration for geological structures with strong lateral velocity variations or with steeply dipping reflectors. The results of the migration technique developed by Kosloff and Baysal showed their technique to be able to accurately migrate these complicated geological structures. Furthermore, they found that there was no need to invoke complicated schemes in an attempt to correct the deficiencies of one-way equations [2].

1.2 DESCRIPTION OF THE PRESENT STUDY

Although the technique developed by Kosloff and Baysal provides an excellent migration algorithm, it still is a two-dimensional migration technique. The object of this research was to extend the 2D migration technique of Kosloff and Baysal into a 3D migration technique that would migrate a cube of 65,536 (or less) traces, each of length 1,024 samples. This goal immediately imposed several problems that were much greater than extending the numerical methods of Kosloff and Baysal. Of these problems, execution time and data motion were the most significant. Although the 2D migration of Kosloff and Baysal was implemented on a Digital Equipment Corporation VAX-11/780 incorporating a FPS-100 array processor, with favorable processing time, it was observed that this hardware was much too small to expect it to handle the 3D technique in a reasonable amount of time. Consequently, for its high rate of computation, the CDC CYBER 205 located at Colorado State University (CSU) was chosen to be the target machine. In Chapters II, III and IV, the following aspects of the 3D migration technique are developed: (1) the numerical methods involved; (2) the major features of the program implementing the 3D migration technique; and (3) the results of numerical tests of the program.

II THE KOSLOFF/BAYSAL FOURIER TECHNIQUE

2.1 INTRODUCTION

Conventional finite-difference migration has relied on approximations to the wave equation which allow energy to propagate only downwards. Although generally reliable, such equations usually do not give accurate migration for structures with strong lateral velocity variations or with steep dips. The migration technique presented here is a three-dimensional extension of a two-dimensional migration technique developed earlier by Kosloff and Baysal [3]. The migration technique presented here, referred to in this paper as the KBF migration technique (for Kosloff/Baysal Fourier type), is based on the full acoustic wave equation, (2.1).

$$\frac{\partial}{\partial x}\left[\frac{1}{\rho}\frac{\partial P}{\partial x}\right]+\frac{\partial}{\partial y}\left[\frac{1}{\rho}\frac{\partial P}{\partial y}\right]+\frac{\partial}{\partial z}\left[\frac{1}{\rho}\frac{\partial P}{\partial z}\right]=\frac{1}{c^2\rho}\frac{\partial^2 P}{\partial t^2} \qquad (2.1)$$

2.2 INPUT

It is assumed that input to the KBF program consists of a "cube" of zero-offset traces in (x,y,z=0,t) space. The KBF technique presented here is designed to handle Nx * Ny such traces corresponding to Nx * Ny uniformly spaced points in the x and the y directions. The implementation discussed is designed so that the following must be true:

$32 <= Nx <= 256$ and $Nx = 2^i$ for some integer i

$32 <= Ny <= 256$ and $Ny = 2^j$ for some integer j

These restrictions were chosen so as to test program efficiency; they do not apply, in general, to the KBF scheme.

For each (x, y) pair, there will be N_t sample points in time, t_m, $m = 1, \ldots, N_t$, at which values of pressure, $P(x,y,z=0,t_m)$ are given. N_t must also be a power of two.

In equation (2.1) it is assumed that the density, ρ, is constant and that the velocity function, c(x,y,z), will be provided by the user. For testing purposes, velocity is given by a Fortran function subprogram in the code presented in Appendix. Other forms representing the velocities may be used to replace the supplied function.

2.3 THE KOSLOFF/BAYSAL TECHNIQUE, IN 3D

OBJECT OF THE PROGRAM

Given $P(x, y, z=0, t)$ for $t = 0, 1DT, 2DT, \ldots, TMAX$

obtain $P(x, y, z, t=0)$ for $z = 0, 1DZ, 2DZ, \ldots, ZMAX$

BASIC NUMERICAL METHOD

Equation (2.1) is Fourier transformed with respect to time, assuming density, ρ, is constant. The second order transformed equations can then be reduced to a system of first order equations in the usual manner. If density is constant, then we can write the following series of equations:

$$\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} + \frac{\partial^2 p}{\partial z^2} = \frac{1}{c^2}\frac{\partial^2 p}{\partial t^2}$$

$$P(x,y,z,t) = F^{-1}p(x,y,z,w)$$

$$\frac{\partial p}{\partial t} = jwF^{-1}p$$

$$\frac{\partial^2 p}{\partial t^2} = -w^2F^{-1}p$$

where

$$W = \begin{bmatrix} w_0 & & & \\ & w_1 & & \\ & & \ddots & \\ & & & w_{n-1} \end{bmatrix}$$

$$\rightarrow \quad \nabla^2 F^{-1}\tilde{p} + \frac{\partial^2}{\partial z^2} F^{-1}\tilde{p} = -\frac{\kappa^2}{c^2} F^{-1}\tilde{p}$$

$$\rightarrow \quad \nabla^2 \tilde{p} + \frac{\partial^2}{\partial z^2}\tilde{p} = -\frac{\kappa^2}{c^2}\tilde{p}$$

$$\rightarrow \quad \frac{\partial}{\partial z}\begin{bmatrix} \tilde{p} \\ \frac{\partial \tilde{p}}{\partial z} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ \frac{\kappa^2}{c^2} - \nabla^2 & 0 \end{bmatrix}\begin{bmatrix} \tilde{p} \\ \frac{\partial \tilde{p}}{\partial z} \end{bmatrix} \qquad (2.2)$$

where

$$\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \qquad (2.3)$$

which is of the form

$$\frac{\partial v}{\partial z} = f(z,v) \qquad (2.4)$$

where

$$v = \begin{bmatrix} \tilde{p} \\ \frac{\partial \tilde{p}}{\partial z} \end{bmatrix} \qquad (2.5)$$

The expression "transformed with respect to time" means that the functions $P(x,y,z,t_m)$ are represented by Discrete Fourier Transforms:

$$P(x,y,z,t_m) = \sum_{i=1}^{N_t} \tilde{P}(x,y,z,w_i) e^{jwit_m} \qquad (2.6)$$

where

$$t_m = \begin{cases} (m-1)\ DT \text{ for } m = 1, 2, \ldots, \frac{N_t}{2} + 1 \\ (m-(N_t+1))DT \text{ for } m = \frac{N_t}{2} + 2, \ldots, N_t \end{cases}$$

$\tilde{P}$ is given by the Inverse Discrete Fourier Transform:

$$\tilde{P}(x,y,z,w_i) = \frac{1}{N_t} \sum_{m=1}^{N_t} P(x,y,z,t_m)\ e^{-jwit_m} \qquad (2.7)$$

where

$$w_i = \begin{cases} \frac{2\pi}{DTN_t}\ (i-1) & \text{for } i = 1, 2, \ldots, \frac{N_t}{2} + 1 \\ \frac{2\pi}{DTN_t}\ (i-(N_t+1)) & \text{for } i = \frac{N_t}{2} + 2, \ldots, N_t \end{cases}$$

DT is the sampling interval in time; $j = \sqrt{-1}$. Equation (2.6) is then substituted in (2.1). This results in (2.2), which must be satisfied for each w_i, for $i = 1, \ldots, \frac{N_t}{2} + 1$.

Thus, the N_t partial differential equations which provide a discrete approximation to (2.1), involving unknown functions $P(x,y,z,t_m)$ are replaced by $\frac{N_t}{2} + 1$ partial differential equations involving unknown functions $\tilde{P}(x,y,z,w_i)$. Note that in the transformed equations, dependence on time, t, has been eliminated.

With an appropriate approximation to $\nabla^2 \tilde{P} \equiv \frac{\partial^2 \tilde{P}}{\partial x^2} + \frac{\partial^2 \tilde{P}}{\partial y^2}$

the "classical" 4^{th} order Runge-Kutta algorithm is applied to integrate equation (2.2) numerically in z. The (vector) computational equations are summarized below:

$$K1 = Dz * f(z, v_{old})$$

$$K2 = Dz * f(z + \frac{Dz}{2}, v_{old} + \frac{K1}{2})$$

$$K3 = Dz * f(z + \frac{Dz}{2}, v_{old} + \frac{K2}{2})$$

$$K4 = Dz * f(z + Dz, v_{old} + K3)$$

$$v_{new} = v_{old} + (K1 + 2K2 + 2K3 + K4) / 6$$

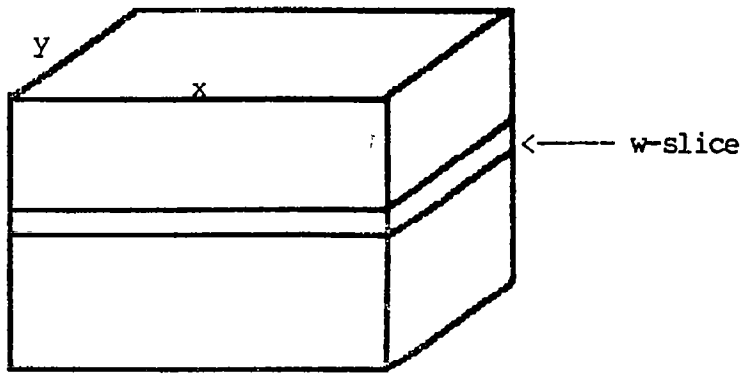
2.4 KBF DESIGN OUTLINE

The program has four main subdivisions, whose tasks are summarized below:

Part I: For each pair of (x,y) values, the corresponding zero-offset trace of $P(x,y,0,t)$ values is converted to another "trace" of $\tilde{P}(x,y,0,w)$ values by application of the discrete Fourier transform (2.7).

Part II: For each w_i value ($i=1,2,\ldots,N_t$) the $\tilde{P}(x,y,0,w_i)$ values are re-ordered into w_i-slices organized either sequentially in y for each x, or sequentially in x for each y, as appropriate for further transformations.

Part III: Each w_i-slice, from the transformed input cube of $\tilde{P}(x,y,0,w_i)$ values (see Figure 2.1), is developed into an (x,y,z,w_i) cube of $\tilde{P}(x,y,z,w_i)$ values. This development is performed by integrating equation (2.2) numerically. The resulting $\tilde{P}(x,y,z,w_i)$ values are accumulated for all w_i for each (x,y,z) combination. Since all the related exponential multipliers $e^{jm_it_1}$ equal 1 in magnitude (see equation (2.6)), this results in the generation of $P(x,y,z,t=0)$ values, as required. (Note: $t_1 = 0$)



Note: z = 0 throughout this data cube.

Figure 2.1
Transformed Input Cube

There are two sub-problems of Part III:

Part III.1: Initial values for $\frac{\partial \tilde{\tilde{P}}}{\partial z}$ are obtained by the application of a two-dimensional Fourier transform to $\tilde{P}$ followed by multiplication by SQRT[-1 * $(\frac{w^2}{c^2} - \nabla^2)$]. Evanescent energy components are then eliminated and $\frac{\partial \tilde{P}}{\partial z}$ is obtained by the application of a 2-dimensional inverse Fourier transform to $\frac{\partial \tilde{\tilde{P}}}{\partial z}$.

Part III.2: $\tilde{P}(x,y,z,w)$ and $\frac{\partial \tilde{P}}{\partial z}(x,y,z,w)$ are propagated from z to z+ ∇z using the Runge-Kutta 4^{th} order method to integrate equation (2.2) numerically. To do this

$$\nabla^2 \tilde{p} = \frac{\partial^2 \tilde{p}}{\partial x^2} + \frac{\partial^2 \tilde{p}}{\partial y^2}$$

must be approximated four times for each ∇z. This is achieved by the use of a two-dimensional Fourier transform, followed by multiplication by $-(k_x^2 + k_y^2)$. Evanescent energy is eliminated from $\tilde{P}$ by applying a two-dimensional Fourier transform to $\tilde{P}$, obtaining $\tilde{\tilde{P}}$. For all (K_x, K_y) pairs such that $K_x^2 + K_y^2 > w_i/c(x,y,z)$, $\tilde{\tilde{P}}$ is set to zero. Then a two-dimensional inverse Fourier transform is applied to yield $\tilde{P}'$, which is input to the next step of numerical integration. Evanescent energy is also removed from $\frac{\partial \tilde{P}}{\partial z}$ in the same manner.

Part IV: For each (x,y), the P(x,y,z,t=0) values in Part III are retrieved so as to be contiguous in Z. These space traces are each Fourier transformed and the downgoing energy is eliminated by filtering out components with negative wave numbers K_z. The resulting filtered traces are inverse Fourier transformed, retaining only the real part of the result, which is the desired 3D depth migration.

III PROGRAM DESIGN FEATURES

3.1 INTRODUCTION

The speed and capacity of the computer available to an individual researcher imposes certain restrictions on the types of problems that can be solved. The CYBER 205's vector features and high speed scalar processor provide a tool for solving problems in a matter of minutes that would take on the order of days on a conventional scalar machine (this speed increase depends, to a considerable extent, on the degree to which it is possible to "vectorize" the scalar code). Of the problems that can now be solved using the CYBER 205, the migration application presented here makes extensive use of the CYBER 205's vector facilities. This chapter contains an overview of vector processing on the CYBER 205 and an in-depth discussion of the data-flow required by the KBF migration algorithm.

3.2 CONCEPTS OF VECTOR PROCESSING

This section deals primarily with the concept of vector machines; however, it is not within the scope of this paper to bring the novice up-to-date on vector computing. Several texts and papers have been written to perform that task. Hockney and Jesshope [4] present a comprehensive text covering vector and parallel processors as well as vector and parallel algorithms. Section 2.3 of Hockney and Jesshope [5] is dedicated to the CDC CYBER 205. For more information on the CYBER 205, see also Kascic [6].

THE CDC CYBER 205, HISTORY

The CYBER 205, announced in 1980, replaced its predecessor, the CYBER 203. In turn, the CYBER 203, introduced in 1979, was a re-engineered version of the STAR 100. Conceived in 1964, the first STAR 100 became operational in 1973. The instruction set for the vector operations in the STAR 100 were based, primarily, on the APL language. The STAR 100 was designed to execute at a rate of 100 Mega-flops (1 Mega-flop = one million floating point instructions executed per second).

THE CDC CYBER 205, DESIGN

The CYBER 205 is a member of the family of "pipelined" machines. Pipeline refers to an assembly-line style of performing certain operations; thus more than one set of operands can be operated upon at a time. The vector processor of the CYBER 205 has what are known as vector pipes. These vector pipes are designed to stream contiguous data elements (vectors) through their pipelines. Presently, the CYBER 205 can have as many as four vector pipes, all of which can operate concurrently. A four pipe CYBER 205, processing 32-bit words, can operate at a peak rate of 800 mega-flops.

The various data types utilized by the CYBER Fortran 2.0 language include the following:

Type		Comments
Bit	:	the machine is bit addressable
Half-word	:	32-bit floating point
Full-word	:	64-bit floating point; 64-bit integer
Double-precision	:	128-bit floating point
Complex	:	two consecutive 64-bit words

Vectors on the CYBER 205 are "pointed to" by vector descriptors. A vector descriptor is a 64-bit entity with the following two fields: (1) Vector length, which consists of 16 bits and (2) Virtual address of the first vector element, which consists of the remaining 48 bits. Thus, a vector can have a length ranging from 0 to 65,535. Note that a bit vector can be no longer than 65,535 elements even though it consists of only 1024 64-bit memory words.

Vector operations come in a variety of forms on the CYBER 205, some of which are displayed in Table 3.1.

Table 3.1. Vector Operation Examples.

```
                DIMENSION A(100), B(100), C(100)

                L = 100

EXAMPLE
NUMBER   VECTOR CODE                          EQUIVALENT
                                              SCALAR CODE
-------  ----------                          ----------

  (1)    A(1; L) = Q8VINTL(0, 1; L)              DO 10 I = 1, L
                                           10  A(I) = I - 1

  (2)    B(1; L) = A(1; L) * 20.0                DO 20 I = 1, L
                                           20  B(I) = A(I) * 20.0

  (3)    C(1; L) = A(1; L)*2.0+B(1; L)           DO 30 I = 1, L
                                           30  C(I)=A(I)*2.0+B(I)
```

The examples in Table 3.1 are rather simple but resemble many operations in scientific programs. Examples 1 and 2 show a vector function call and a vector-scalar operation. Example 3 shows a "linked triad" operation. A linked triad operation takes advantage of CYBER 205 hardware which supports such operations. As one can see in Table 3.2, the linked triad operations are quite efficient. An operation is generally considered a linked triad when it consists of two vector operands and one scalar operand.

In certain situations, the results of some elements of a vector operation need not be saved. In this case, there is a mechanism for avoiding storage which involves a control vector. A control vector is a bit vector that specifies the storage of vector results. The control vector will be the same length as the result vector and where it has a value of one the corresponding result vector element will be saved and where it has a value of zero the corresponding result vector element will not be saved. The programmer also has the choice of reversing the meaning of the one's and zero's in the control vector.

A certain number of clock cycles are needed to set up the vector pipes. As this setup time is constant for a given operation, it is more efficient, in terms of total execution time, to reduce the number of vector operations by increasing the vector lengths whenever possible. Table 3.2 shows the set-up times, as well as the timings for the actual operations for various operations on the CYBER 205.

Table 3.2. Vector Timing Information

Vector Instruction	Number of Set-up Cycles	Number of Operating Cycles
Addition, Subtraction	51	N / 4
Multiplication	52	N / 4
Division, Square root	80	N / .61
Linked triad	84	N / 4

Where:
N = Vector length
1 Cycle = 20 nano-seconds
The vector operations are on 32-bit words

3.3 A NOTE ON THE APPLICATION OF VECTOR PROCESSING TO THE KBF METHOD

The KBF migration technique is such that almost all of the necessary operations can be vectorized. When working with a particular w-slice, all of the operations, including the two-dimensional FFT's, are vector operations. The computations performed at any given point of the omega-slice must be performed at all of the points. If there is a certain criteria that causes something different to occur at a given omega-slice point, a control vector can be created, dynamically, and the operation can still be performed in a vector manner. An example of this may be found in the routine CUTOFF where the evanescent energy is eliminated. In summary, there is no particular operation in the KBF migration scheme that can not be treated as a vector operation. To emphasize this point, one should examine the technique presented in chapter 2 and notice that there are no tricky operations that would prevent vectorization. In particular, it is important to note that there are no operations that have the following structure:

```
      DO 100 I = 1, N
         X(I) = F(Y(I))
         IF (X(I) .LT. VAL) GO TO 200
100   CONTINUE
200 CONTINUE
```

The above code can not be efficiently vectorized because of the inherently sequential nature of the computations.

3.4 DATA CONSIDERATIONS

As previously discussed, a program implementing the KBF migration technique, extended into three dimensions, is easily expressed in terms of vector operations. The program developed here contains very few scalar operations, many of which are operations needed in order to control various vector instructions or vector subroutine calls. Having such a match of software to hardware, one might conclude that there are no remaining barriers to running the program. There are, however, a few major items that one tends to overlook, being overwhelmed by the computational power of the CYBER 205. The greatest of these is the data motion required to keep the CYBER 205 vector pipes busy.

One penalty for the use of vector operations is that the data must be contiguous in memory for greatest efficiency (let alone for some vector operations to run at all). Furthermore, the vectors must reside in main memory as much as possible in order to prevent sure death from thrashing. With this in mind, one must realize that the memory requirement for the vectors that are necessary to perform a single step of the integration of one omega slice is quite large. For example, a (256 by 256) complex XY plane will require eleven vectors of length 131,072 half-words. These, along with various support vectors, comprise 12 large pages (1 large page = 65,536 full-words). This is slightly less than half of the memory available to a user on a

2-megaword 205, however it is about all one can expect to get for any reasonable period in a time-sharing environment. But this is really just the tip of the iceberg - these are just the work arrays. The total data set consists of the input data cube, the work arrays, and the output data cube.

Continuing with the previous example, the input cube could very well be of size 256*256*1024 half-words and the output cube could be as much as twice the size of the input cube (the size of the output cube depends upon the number of ZSTEPS in the migration). This would be a total of 201,326,592 half-words, which is equivalent to 1536 large pages. Obviously, this is much more data than any CYBER 205 can have in memory at any given time. Consequently, the question of how to handle the data-flow arises. A solution that one may consider is to declare the data cubes to be huge arrays and to let the virtual memory mechanism handle the data cubes.

To consider declaring the two data cubes as arrays, one must realize that access to these two arrays would have to be in a contiguous manner. Otherwise severe thrashing would result. In the case of the KBF migration algorithm, access to the data cubes must be done in several ways that would break the rule of contiguous access. Thus, it would be wise to check into at least one alternate method of handling these data cubes as large arrays.

Before presenting the data motion method used in this study, the need for efficiency must be established. Continuing with the previous example and without discussing the code in detail, the subroutine RHS3 takes on the order of 100 milli-seconds to run, each time it is called. In this example, RHS3 would be called on the order of 4*512*512 (1,048,576) times. The time needed for all of these calls is approximately 29 hours. Thus, any time for performing the data-motion is added onto the 29 hours. Therefore, one needs to find a mechanism to perform the data-motion without making the program run for an unacceptable amount of time.

3.5 A FOUR-WAY PARALLEL DATA MOTION TECHNIQUE

CYBER 205 Fortran provides several routines that may be used to implement I/O that runs concurrently with other instructions being executed as well as with other I/O. These routines include Q7BUFIN, Q7BUFOUT, and Q7WAIT. For detailed information on these routines, see the CDC CYBER 200 FORTRAN VERSION 2 manual [7]. A typical use for these routines would be as follows:

```
          .
          .
          .
CALL Q7BUFOUT(.................)
CALL WORK(.............)
          .
          .
          .
```

In this example where the programmer wishes to write information out to a unit and have the routine WORK run concurrently with the I/O. In general, as long as WORK does not use the I/O unit referred to in the Q7BUFOUT call, it can do anything it wishes. Thus, there is CPU activity concurrent to I/O activity.

Another example where two I/O requests cause concurrent I/O, is as follows:

```
          .
          .
          .
CALL Q7BUFIN(.................)
CALL Q7BUFOUT(.................)
          .
          .
          .
```

According to the CDC CYBER FORTRAN 2 manual [8], these calls are legal, so long as they do not access the same data block on the same disk. Also, two Q7BUFIN, two Q7BUFOUT calls, or a Q7BUFIN and a Q7BUFOUT call can be active at one time for a given unit.

It should be obvious that these "Q7" calls are the basis of a solution to the problem of data-flow that was presented in the previous section. Indeed, they are; yet they are only the basis of the method used in this study. Dr. Bjorn Mossberg [9], of Control Data Corporation, wrote a utility known as SLICE4. Mossberg used the "Q7" utilities; however, the scheme he developed is much more elaborate than a series of Q7 calls to a particular I/O unit.

SLICE4

It is not within the scope of this paper to duplicate Mossberg's documentation of SLICE4. However, the concept and the terminology of SLICE4 will be presented as it applies to this study. For efficient operation, SLICE4 must be tightly integrated into the master program. Therefore, its terminology affects the view that one takes of the master program.

In this study, two implementations of SLICE4 were needed and used; one for the input data cube and one for the output data cube. To explain the use of SLICE4, only the input data cube will be treated. The output data cube is handled in a similar manner.

SLICE4 TERMINOLOGY

The first step in using SLICE4 is to impose a coordinate system upon the data cube such that the cube is N1 by N2 by N3 elements in size, where N1 is the number of elements in what one normally considers the Z direction, N2 is the number of elements in the X direction, and N3 is the number of elements in the Y direction. The next step is to define a second coordinate system on the data cube. Instead of being coordinates of individual data items, this second coordinate system gives coordinates of "super-blocks." Super-blocks are small cubes of the original data set. The super-block coordinate system has NS1 super-blocks in the 1-direction, NS2 in the 2-direction, and NS3 in the 3-direction, where NS1 and NS2 must be multiples of four. NS3 does not have this restriction; however, for greatest efficiency, it should be one or a multiple of four. The reason for the multiple of four rule is that the super-blocks will reside on four different I/O units. No matter which direction the cube is accessed, each I/O unit will have one quarter of the super-blocks accessed. This is not the case when only a partial row or column of super-blocks is accessed; thus, it is most efficient to access a complete row or column. If it should happen that more than one I/O unit be controlled by a given controller, then SLICE4 will still execute, but in a less efficient manner (i.e. the parallelism is partially inhibited). Thus, one may access any four adjacent super-blocks at a cost which is one fourth the cost of accessing the same data with conventional techniques.

The super-blocks themselves have a coordinate structure imposed upon them. This coordinate structure is L1 by L2 by L3. Where L1 is the number of elements from the data cube in the 1-direction; L2 and L3 are defined in the same manner for their individual directions.

Summarizing the terminology presented so far, the original data cube is broken up into NS1 by NS2 by NS3 super-blocks. Each super-block has L1 by L2 by L3 data elements. Thus the following rules must apply:

```
N1 = NS1 * L1    with    NS1 = 4 * i,  i => 1
N2 = NS2 * L2    with    NS2 = 4 * j,  j => 1
N3 = NS3 * L3
```

SUPER-BLOCK ACCESS

The rows and columns of super-blocks are referred to as slices. A 1-slice is some column of super-blocks in the 1-direction, a 2-slice is some row of super-blocks in the 2-direction, and a 3-slice is some row of super-blocks in the 3-direction. One may access all, or just some, of the super-blocks of a slice via SLICE4. However, in this study, only the most efficient access is performed - accessing all super-blocks of a given slice. As access can be by any given slice, SLICE4 must have the super-blocks all formatted in the same manner. Thus, when accessing a given slice, the slice is written into a buffer by SLICE4 and the user must re-format the data from the buffer into a work array in the format that corresponds to the direction of access.

DIMENSION CONSIDERATIONS

One needs to be careful to have enough array and buffer space to access the data cube in all the necessary directions. Thus, the size of the super-block comes into question. The larger the super-block, the fewer accesses to the data cube are needed and vica versa. In this study, the L1 dimension was set permanently to the value of 2. The reason for this is that, as one recalls from the migration technique, a complete XY plane is processed at any given time and there is only enough memory space to have two input planes in memory at the same time.

IV RESULTS AND CONCLUSIONS

4.1 EXECUTION TESTS

As discussed in section 3.4, it would take over 29 hours of execution time to migrate the maximum (assumed) data cube; thus for testing purposes, an input cube of size (64x64x64) was used. For both of the test runs discussed here, all of the traces consisted completely of zeros, except the center trace that had a single wavelet peaking at sample 16 (in time). The correctly migrated result, in this case, consists of a hemisphere. The first run (Figures 1 and 2) incorporated a padding in the time direction to delay the wrap-around effect inherent in Fourier algorithms. The second run (Figures 3 and 4) did not incorporate a padding - thus, wrap-around effects appeared. The first run took 240 CPU seconds and the second run took 115 CPU seconds.

Test Run 1: The migration of the input cube described above, using a constant velocity of 3000 m/s, a Dz interval of 6.0 meters, a Dx interval of 12.0 meters, a Dy interval of 12.0 meters, and a time interval of 4.0 milli-seconds, yields the results shown in Figures 1 and 2. Figures 1 and 2 are slices of the output cube in the XZ and in the YZ directions, respectively, intersecting at the center of the output cube (Note the absence of the wrap-around effect).

Test Run 2: The migration of the same input cube used in Test Run 1 using the same sampling rates in all dimensions, but with a velocity interface (see Figure 3; V1 = 4000 m/s; V2 = 3000 m/s), yields the results displayed in Figures 3 and 4. Note the wrap-around effect present in these figures.

4.2 FACTORS AFFECTING SPEED OF COMPUTATION

Until a superior algorithm for performing the I/O required by the KBF migration algorithm appears, SLICE4 will remain the most efficient method available to perform the I/O task. However, should a CYBER 205 ever be equipped with 8, or even 16, I/O channels, SLICE4 should easily be adapted to create SLICE8 and SLICE16 versions. Until then, there is little chance of decreasing the time required to perform the I/O.

Other than I/O, the Runge-Kutta 4^{th} order algorithm employed in the KBF migration technique is the most expensive feature. Consequently, use of a less costly method for numerical integration (e.g., a multi-point method, using the Runge-Kutta method to get started) might result in increased computational efficiency.

4.3 CONCLUSIONS

The 3D KBF migration program, implemented on the CYBER 205 Supercomputer presented in this thesis, yields results that are consistent with those of Kosloff and Baysal [10]. This was confirmed by Kosloff [11]. Thus, a 3D migration program, using the KBF migration technique (based on the full acoustic wave equation) permitting lateral velocity variations is now available for use on the CYBER 205.

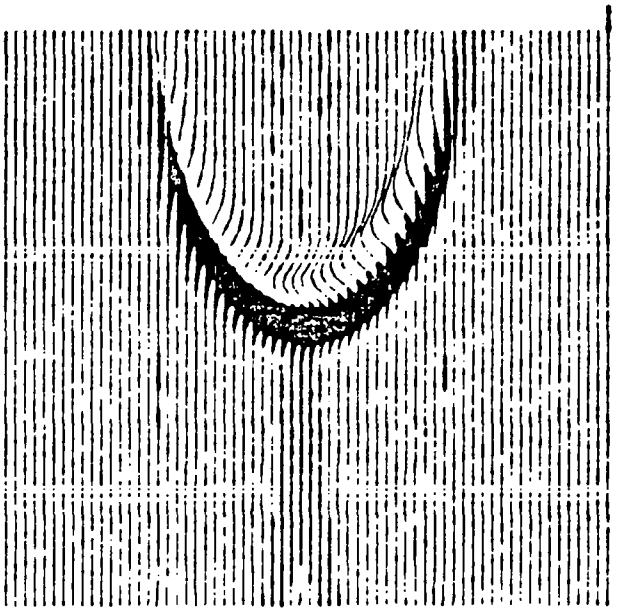


Figure 1

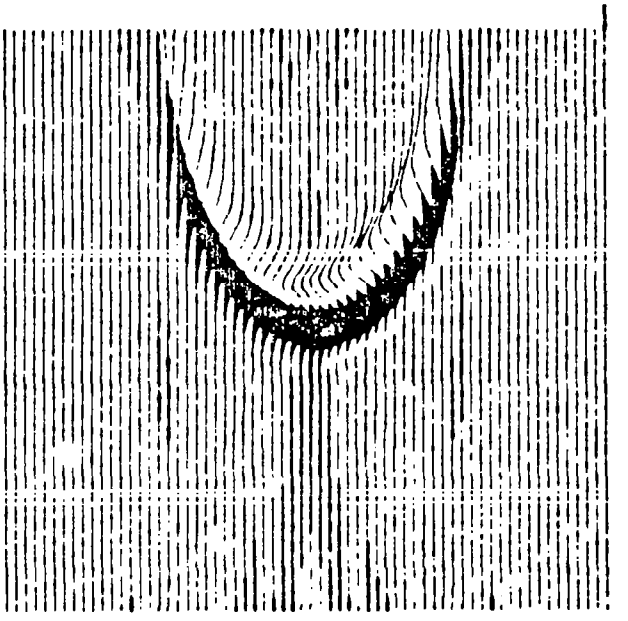


Figure 2

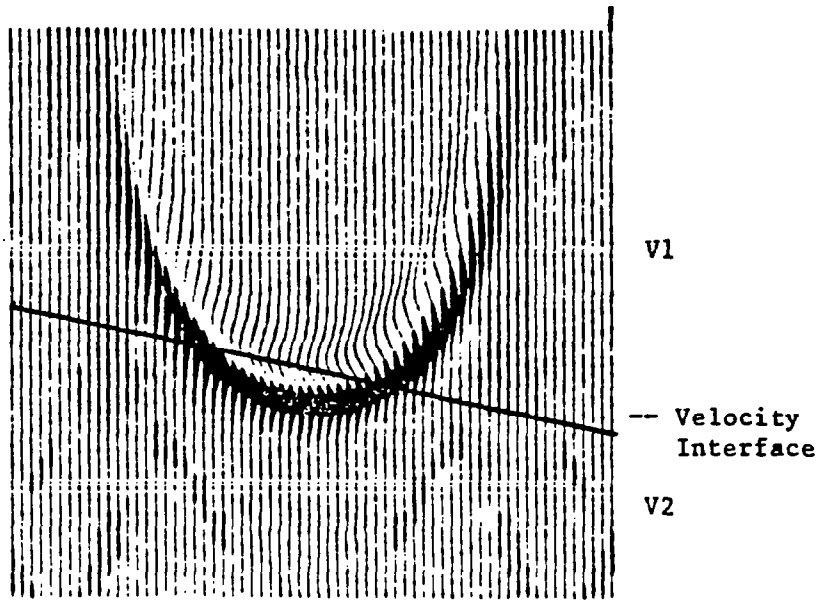


Figure 3

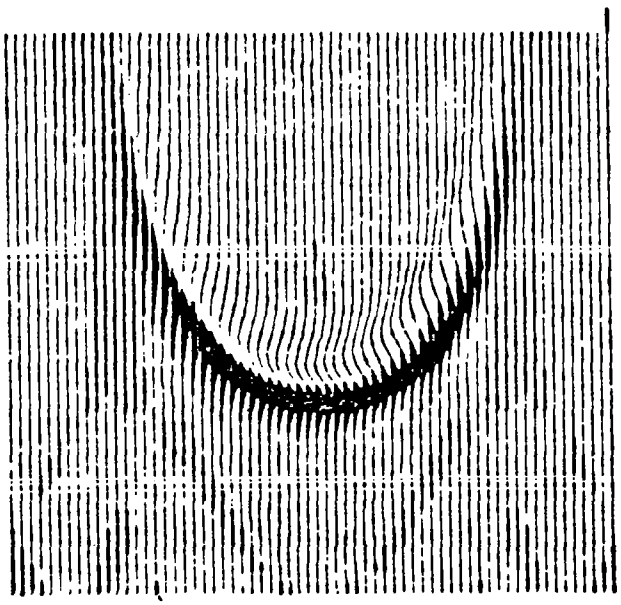


Figure 4

REFERENCES

1. Kosloff, D., and E. Baysal, "Migration With the Full Acoustic Wave Equation," Seismic Acoustics Labortory Fifth Year Semi-Annual Progress Review, No. 9 (1982), pp. 151-165.
2. Kosloff and Baysal, p. 152.
3. Kosloff and Baysal, pp. 151-165.
4. Hockney, R. W., and C. R. Jesshope, Parallel Computers: Architecture, Programming, and Algorithms (Bristol: Adam Hilger Ltd., 1981).
5. Hockney and Jesshope, pp. 95-126.
6. Kascic, M. J. Jr., Vector Processing On the Cyber 200 (St. Paul: Control Data Corporation, 1978).
7. Control Data Corp., CDC Cyber 200 Fortran Version 2 (St. Paul: Control Data Corporation, 1981).
8. Control Data Corp.
9. Control Data Corp., MAGEV Library Utility.
10. Kosloff and Baysal, p. 155.
11. Personal interview with Dan Kosloff, 25 August 1983.

VECTORIZATION OF A PENALTY FUNCTION ALGORITHM FOR WELL SCHEDULING

ILYAS ABSAR

SOHIO PETROLEUM COMPANY

SAN FRANCISCO, CALIFORNIA

Vectorization of a Penalty Function Algorithm for Well Scheduling

Ilyas Absar

SOHIO Petroleum Co.

San Francisco, California

Abstract:

In petroleum engineering, the oil production profile of a reservoir can be simulated by using a finite grided model. This profile is affected by the number and choice of wells which in turn is a result of various production limits and constraints including, for example, the economic minimum well spacing, the number of drilling rigs available and the time required to drill and complete a well. After a well is available it may be shut-in because of excessive water or gas productions. In order to optimize the field performance a penalty function algorithm was developed for scheduling wells. For an example with some 343 wells and 15 different constraints, the scheduling routine vectorized for the Cyber 205 averaged 560 times faster performance than the scalar version.

Introduction:

Mathematical modelling of the fluid production from a naturally occurring reservoir involves considering the reservoir as a network of interconnected blocks. To each grid block is associated a geologic description through properties, e.g., thickness, porosity, permeability, etc. Each grid block is considered to be in material balance with its surroundings, i.e., the amount of fluid in the block at time $t + \Delta t$ is equal to the amount of fluid in that block at time t plus fluid influx in the time interval Δt minus fluid outflux in the time interval Δt.

In Figure 1A, the reservoir is shown by a curved boundary. Overlaid areally is a rectangular grid. The sizes of the blocks can be chosen to represent the geological features of the reservoir as accurately as possible. Figure 1B shows a two dimensional cross-section of a reservoir and the grid used for its simulation. Notice that the reservoir contains water, oil and gas in various regions, and only some blocks are in communication with the wells by means of perforations in the well bore. To simulate the production profile, the material balance of the grid blocks in which wells are perforated must also take into account the fluid production or injection. In this manner one obtains pressures and saturations for each of the grid-blocks. For details on mathematical modelling of oil reservoirs please refer to a standard text, for example, references 1 and 2.

Once a reservoir simulator is formulated, it can be used in many ways, e.g.:

1. Assist in making economic decisions for field operation, e.g., the investments to date at Prudhoe Bay exceed $9 billion.

2. Design of production strategy. The effect of changes in the number, location, spacing, or timing of wells can be studied.

3. Prediction of reservoir performance.

4. Matching of the production history.

When an oil field is developed, of course the most important objective is to maximize oil recovery. However, this objective is tempered by limitations, economic and physical, e.g., costs and capacities of various installations and devices.

The dashed curve in Figure 2 represents oil production when all wells flow at their maximum capacity. The area under this curve represents cumulative oil production. The ratio of cumulative oil production to in-place oil represented as a fraction or percentage is called the Oil Recovery Factor. If facilities were constructed for this production profile, they would have to be constructed to handle oil production at the maximum rate, q_{max}. Economic considerations give us a target oil rate, q_T, less than q_{max}, at which oil production can be sustained for a period of time. The solid curve in Figure 1 represents this strategy. Note that sometimes this can be achieved without appreciable sacrifice in cumulative oil production.

Well Scheduling Problem:

Once q_T is established, the problem of optimal scheduling, i.e., selecting for operation a given number of wells (say n) can be represented mathematically as follows:

Maximize, $\sum_{i=1}^{n} q_i \leq q_T$

The maximum production rates of oil, gas and water are, however, limited to the capacity of the reservoir facilities. Thus, the field oil production is subject to constraints of the form:

$$\sum_{i} x_i q_i \leq L$$

where,

q_i is the oil production rate from well i,

q_T is the target oil production rate for the field,

x_i is either 1 or the gas-oil ratio or the water-oil ratio for well i,

$x_i q_i$ is then the oil or gas, or water production/injection rate.

and L is the oil or gas or water production of injection constraint.

Some examples of these limits are:

1. Fieldwide gas handling capacity,
2. Water injection limit,
3. Oil production limit at a station due to pipeline size,
4. Gas-lift capacity available.

In order to select wells for production, each well can be assigned a priority. In the penalty function approach priority assignment, is made with a function which becomes large as a particular constraint approaches violation.

Suppose (k-1) wells have been already chosen.

For choosing the k th well subject to a constraint of the form:

$$\sum x_i q_i \leqslant L,$$

a simple penalty function is:

$$p(k) = (\sum_{i=1}^{k-1} x_i q_i + x_k q_k)/ L$$

The penalty function p(k) has a value for each of the available wells, and arranges the set of available wells in order according to this particular constraint.

When there are several (say m) constraints, penalty functions $p_1(k)$, $p_2(k)$---$p_m(k)$ can be obtained similarly.

Since each constraint is individually fatal for well scheduling purposes, the violation of one constraint is as bad as any other.

Hence, an overall penalty function can be of the form:

$$p(k) = \max_{j=1 \ldots m} p_j$$

Results and Discussion:

The implementation of this scheme involves calculating for each available well, m different p_i (k) and then obtaining an overall penalty, p(k) as the maximum of these m values. Thereafter the well with the lowest value of p (k) is selected. This procedure is repeated selecting one well at a time until the target rate q is achieved without violating any of the constraints. If the target rate cannot be achieved without violating one or more constraints, we are on the decline portion of the production curve.

This scheme was programmed into a three dimensional, three phase (oil, gas, water) simulator. The simulator originally used a simple prioritization scheme based on gas-oil ratios. When a scalar version of the penalty function algorithm was introduced, the simulator ran appreciably slower. It was therefore decided to vectorize the penalty function algorithm.

To calculate the penalty function in a case with n wells and m constraints declare an array p (n, m). Usually n is much greater than m.

For each of the m constraints vectorize the penalty calculation, e.g., for constraint i, store the values of $p_i(k)$ in the elements of p (n, m), starting with p (1, i) and ending at p (n, i).

Next, using a WHERE comparison statement pick out the largest of the m values for each well. We now have the priority p (k) for each well. Use the Q8SMINI call to pick out the minimum value. If this value exceeds 1., no well can be chosen without violating a constraint.

TABLE 1 .

	Case 1	Case 2
No. of wells	119	343
No. of constraints	9	15
Average Well Selection Time (sees)		
Scalar:	.14	1.6
Vector:	.001245	.00287
Scalar: Vector Ratio	112	560

A summary of results for two cases is presented in Table 1. For a reservoir with 119 wells and nine constraints, the vector algorithm was on the average 112 times faster than the scalar version. For a larger example, Case 2 in Table 1, 343 wells with 15 constraints, the vector algorithm achieved even more spectacular results, an average acceleration factor of 560.

The details of Case 1 are represented graphically in Figure 3. In the scalar algorithm, the time required for selection of wells increases monotonically for each subsequent selection. The selection of the first well required only .005 secs while the selection of the 65th well required .226 secs. However, in the vector algorithm, each well selection required .001244 secs, except for the first, which required .00155 secs.

Similarly, for Case 2, the vector algorithm took .00287 secs for each well selection, except for the first well, for which it took .00447 secs. The scalar algorithm had a monotonic increase from .0185 secs for the first well, to 2.641 secs for the 220th well. This means that the selection of the 220th well was some 920 times faster in the vector algorithm as compared to the scalar version.

Conclusions:

Clearly as the number of wells and the number of constraints increase, the advantage of the vectorized version over the scalar version becomes greater.

The reservoir simulator with the vectorized well selection scheme, including the more complicated penalty function scheme, ran faster than the original version with the simpler scalar well selection scheme.

In short, judicious use of vectorization can make feasible highly desirable enhancements to large simulators.

References:

1. D. W. Peacemain, Fundamentals of Numerical Reservoir Simulation, Elsevier Scientific Publishing Company, N.Y., 1977

2. K. Aziz and A. Settari, Petroleum Reservoir Simulation, Applied Science Publishers Ltd., London, 1979

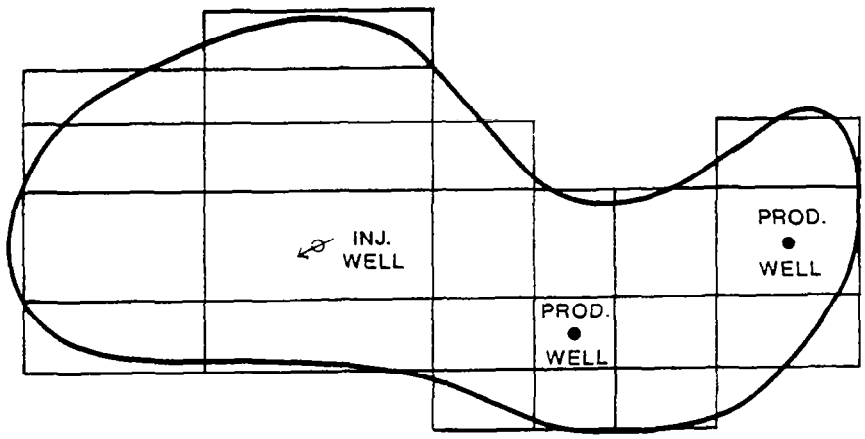


FIGURE 1A.
RECTANGULAR GRID TO REPRESENT A RESERVOIR. EACH BLOCK MAY HAVE DIFFERENT THICKNESS AND POROSITY.

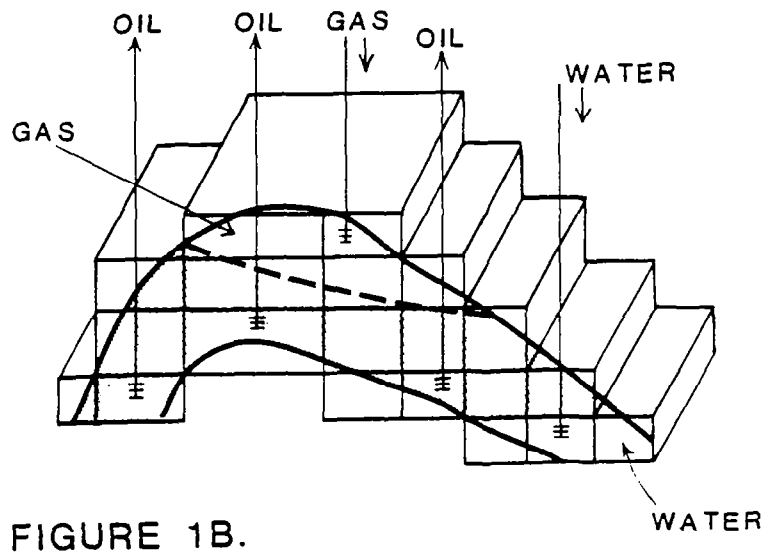


FIGURE 1B.
CROSS-SECTION OR A GRID WITH DIFFERENT TYPES OF WELLS.

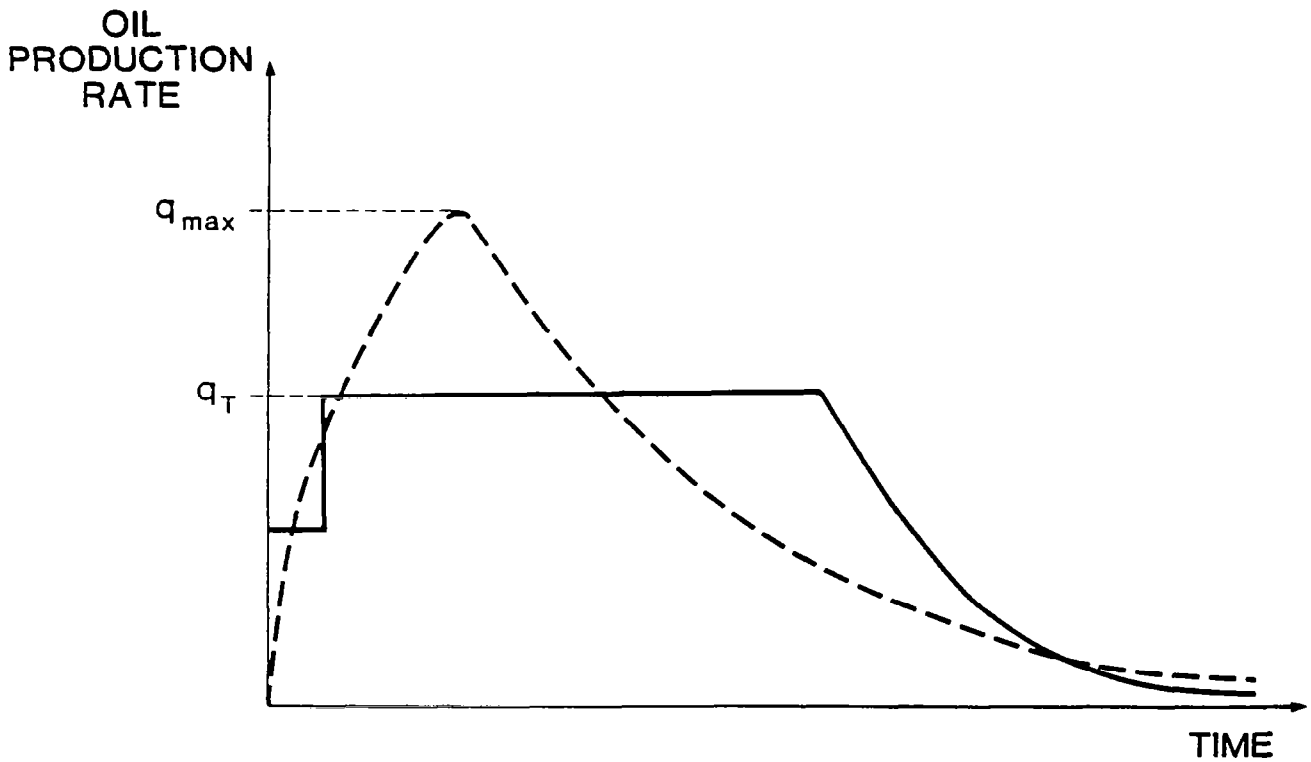


FIGURE 2.

PRODUCTION PROFILE FOR AN OIL FIELD.

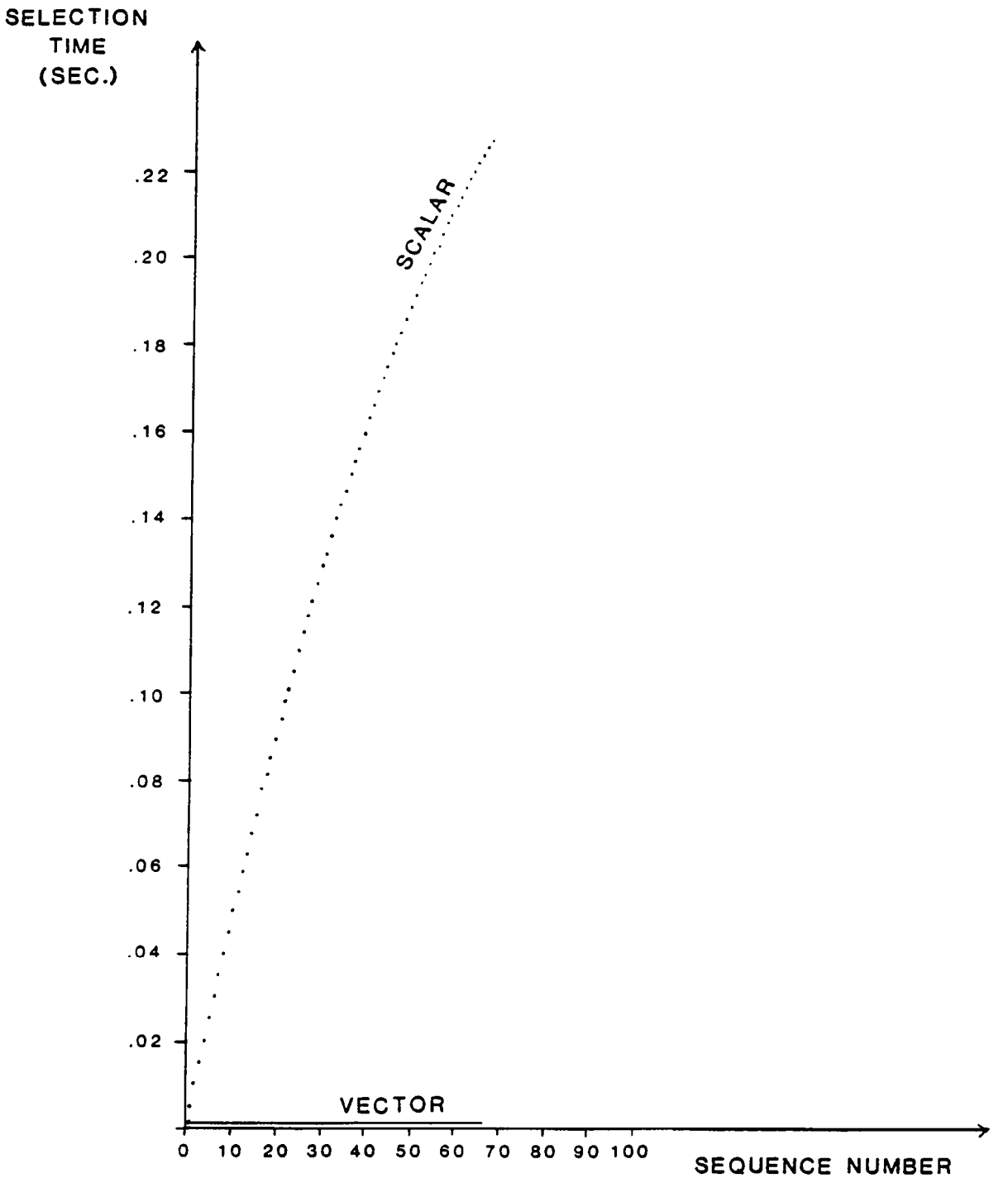


FIGURE 3.

☆U.S. GOVERNMENT PRINTING OFFICE: 1984-739-010/ 59 REGION NO. 4

BIBLIOGRAPHIC DATA SHEET

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA CP-2295		

4. Title and Subtitle	5. Report Date
CYBER 200 Applications Seminar	March 1984
	6. Performing Organization Code: 935
7. Author(s): J. Patrick Gary, Compiler	8. Performing Organization Report No.: 84F5215
9. Performing Organization Name and Address: Goddard Space Flight Center, Greenbelt, Maryland 20771, with Control Data Corporation, Minneapolis, Minnesota 55440	10. Work Unit No.
	11. Contract or Grant No.
12. Sponsoring Agency Name and Address: National Aeronautics and Space Administration, Washington, D.C. 20546	13. Type of Report and Period Covered: Conference Publication, October 10-12, 1983
	14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

This document contains the proceedings of the CYBER 200 Applications Seminar, hosted by Goddard Space Flight Center with Control Data Corporation, held on October 10-12, 1983. The seminar was attended by more than 100 participants, including representatives from the United Kingdom Meteorological Office and CSIRO in Australia, four other countries, fifteen U.S. universities, ten Federal research centers, and numerous private industries. The subjects covered included application topics in Meteorology/Oceanography, Chemistry, Math Algorithms, Fluid Dynamics, Monte Carlo Methods, Petroleum, Electronic Circuit Simulation, Biochemistry, Lattice Guage Theory, Economics, and Ray Tracing. This document is comprised of the majority of the papers presented at the seminar.

17. Key Words (Selected by Author(s))	18. Distribution Statement
Vector Processing Numerical Techniques Supercomputers Vectorization	STAR Category 61 Unclassified-Unlimited

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price*
Unclassified	Unclassified	378	A17

*For sale by the National Technical Information Service, Springfield, Virginia 22161

GSFC 25-44 (10/77)