Comp. & Maths with Appls. Vol. 8, No. 5, pp. 367–377, 1982
Printed in Great Britain.

0097–4943/82/050367-11$03.00/0
Pergamon Press Ltd.

FAST ORTHOGONAL DERIVATIVES ON THE STAR

SHAHID H. BOKHARI[†]
Department of Electrical Engineering, University of Engineering & Technology, Lahore-31, Pakistan

M. YOUSUFF HUSSAINI[†]
Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hampton, VA 23665, U.S.A.

and

STEVEN A. ORSZAG[‡]
Department of Mathematics, Massachusetts Institute of Technology, Cambridge, MA 02139, U.S.A.

Communicated by Steven A. Orszag

(*Received November* 1981)

Abstract—In many numerical problems there is the need for obtaining derivatives in the X and Y directions of m variables at each point on an $n \times n$ plane. We consider the case where these derivatives are obtained using spectral methods (i.e. n fast Fourier transforms of length n are taken for each component, multiplied by the wave numbers and reverse transformed).

On the CDC STAR-100 all data points corresponding to a plane must be stored in contiguous locations if advantage is to be taken of the powerful pipeline hardware of the machine. This means that derivatives in one direction are obtained very efficiently while derivatives in the orthogonal direction require either the substantial overhead of transposition or the use of scalar operations with no benefits of pipelining.

An algorithm is described that overcomes this problem by taking derivatives of all components simultaneously. This is made possible by perfect shuffling of data to effect a pseudo-transposition that permits the FFT routine to take transforms of all m components on a plane at one time. Practical experience with this algorithm for $m = 5$ and $n = 32$ shows a 10% speedup for X-derivatives and a 32% speedup for Y-derivatives over the conventional algorithms (in which X and Y derivatives are taken one component at a time and Y derivatives require transposition of data).

A theoretical analysis based on available STAR-100 vector instruction timing data predicts that this algorithm is superior to the conventional algorithm for $M \geq 2$, $n \leq 128$ (problem sizes of practical interest). We show how further improvement in running time may be obtained if derivatives of several components on more than one plane are required.

This analysis is applicable to the new generation of STAR computers (the CDC Cyber 203s) since vector instruction timings are essentially unchanged in the new machines.

1. INTRODUCTION

Supercomputers such as the CDC STAR-100, CRAY-1 and ILLIAC IV are intended primarily for the solution of large numerical problems. Because of their vector architecture, design and analysis of algorithms for these computers is a complex task, requiring more intimate knowledge of machine architecture than is the case for scalar machines. At the same time, design of efficient algorithms is all the more important for these machines because they are scarce resources—there is only one ILLIAC IV, a few STAR 100s and several dozen CRAY-1s in the word.

In a supercomputer installation running several production jobs of a few hours each, every day, a reduction in running time of 10–25% in a few jobs may mean that a new daily job can be accomodated or that more time can be made available for program development. For programs that process data taken from the physical world and whose usefulness depends on the speed with which results are made available (e.g. weather prediction), a reduction in running time increases the value of the final output.

Thus for supercomputers it is especially important to search for algorithms that are efficient

[†]Work supported by NASA Contract NAS1-14101 at ICASE, NASA Langley Research Center, Hampton, Virginia, U.S.A.
[‡]Work supported by NASA Contract NAS1-16237.

in a practical sense. For such machines a new algorithm that reduces the running time of a program or subroutine by an appreciable amount *for a problem size of interest* even if its asymptotic run time is poorer than an older algorithm. Clearly, an algorithm that is superior to all others for a "sufficiently large" problem size is of little use if the "sufficiently large" problem size is not of interest or cannot be accomodated in the machine.

In this paper we describe one such algorithm that we have developed as part of a large program to solve 3-*D* Navier–Stokes equations on the CDC STAR-100 at NASA Langley Research Center. Our program solves these equations for a cuboid of space of size $32 \times 32 \times 64$ points in the *X*, *Y*, and *Z* directions respectively. At each point in this cuboid we have the *X*, *Y* and *Z* components of velocity (denoted *U*, *V* and *W*), and pressure (*P*) and density (*R*). During the course of our program we need the derivatives of *U*, *V*, *W*, *P* and *R* in the *X* and *Y* directions on each of the planes. These derivatives are obtained spectrally, i.e. we take *n* fast Fourier transforms of length *n* each in the *X*(*Y*) direction on each plane, multiply by the "wave numbers" $\sqrt{(-1)}\, k_x(k_y)$ and then take reverse FFTs in the *X*(*Y*) direction. The conventional algorithms for obtaining these derivatives are called *DX* and *DY* in this paper.

A major problem with these algorithms arises because of the requirement on the STAR that elements of a vector occupy contiguous locations in memory. Thus if the 32×32 matrix containing the *U* components for a plane is stored in row order, vector operations can only be done along the rows of *U*. Vector operations along the columns of *U* can be done only after the substantial overhead of matrix transportation. In our program data is stored such that algorithm *DY* requires the overhead of transposition and is thus considerably slower than *DX*.

We have developed a new algorithm that permits derivatives on all five components to be taken simultaneously. We call this algorithm *DS* (for "simultaneous"). This algorithm outperforms both *DX* and *DY* because instead of taking 5 separate derivatives with 32 FFTs of length 32 each (plus transposition overhead in the case of *DY*), it finds derivatives on all 5 components simultaneously with 160 FFTs of length 32 each. This is done by using perfect shuffles to effect a pseudo-transposition as described in Section 3. The reduction in run time per FFT brought about by taking 160 instead of 32 transforms outweighs the overhead of shuffling so that in the end *DS* is 10% faster than *DX* and 32% faster than *DY*. *DS* is essentially the same whether used for *X* of *Y* derivatives, the only difference lies in the way shuffling is done. This is explained in Section 3.

Theoretical predictions of run times of algorithms *DS*, *DX* and *DY* for problems with *m* components and plane sizes of $n \times n$ are given in Section 4. All three algorithms are shown to have $0\ (mn^2 \log_2 n)$ running time and algorithms *DS* has the poorest *asymptotic* run time. However we show in Section 5 that *DS* is superior to *DY* over most of the range of problem sizes of interest ($16 \leq n \leq 256$, $m \geq 2$) and is better than *DX* over about 40% of this range.

It is possible to obtain further improvements in performance by using algorithm *DS* to take derivatives of all components on several successive *X*–*Y* planes simulataneously. This technique is described in Section 5.

We start with a brief overview of the STAR-100 in Section 2.

2. THE STAR-100

The Control Data Corporation STAR-100 is a pipelined vector computer designed primarily to solve large numeric problems. It has a 40 nsec minor clock cycle and the wordlength may be either 32 or 64 bits. The physical memory size may be either 0.5 or 1 Megaword and the virtual memory address space is 2^{42} words.

The specific machine at NASA Langley Center on which the algorithm described in this paper was developed had a 524288 word memory. This machine has since been upgraded to a Cyber 203 with 1 Megaword memory and improved scalar instruction times. However the vector instruction times are essentially unchanged and the results of this research are equally applicable to the upgraded machine.

In common with all pipelined machines, the time required to perform a vector operation on the STAR is described by the relationship $T_{op} = S_{op} + Nf_{op}$. Here S_{op} is the startup time associated with the operation "op" and equals the time that must elapse before the first result is available. *N* is the length of the vector being processed. f_{op} is the time per result, once the first result has been computed. If all timings be measured in minor clock cycles (40 ns) then S_{op}

varies from 30 to about 300 and S_{op} varies from 1/2 to 8. Details of timings of interest to us are given in Table 1. (Note—All timings are for full word instructions since our program was developed for full word data.)

Clearly, it is always advantageous to work with as long a vector as possible in order to minimize the startup overhead per vector element. Very careful programming and data layout are required to achieve long vector lengths.

Two hardware constraints are of relevance. Firstly the length of a vector cannot exceed 65536 elements. Secondly, the elements of each vector must occupy contiguous locations in memory.

3. PRACTICAL EXPERIENCE WITH THE THREE DERIVATIVE ALGORITHMS

During the course of our Navier–Stokes program we need to obtain derivatives in the X and Y directions for each of the components U, V, W, R, and P on each plane. This problem is made difficult by the contiguous storage requirement for vectors on the STAR. The usual result of this constraint is that derivatves in one direction are obtained very efficiently while derivatives in the orthogonal direction require the overhead of transportation or the use of scalar operations (with no benefits of pipelining).

We have developed an algorithm which overcomes this problem by taking derivatives of all five components simultantously. This is made possible by reverse shuffling our data appropriately to effect a pseudo-transposition. The end results are that (1) derivatives in the X and Y directions require almost the same amount of time for our particular problem size and (2) the time required to obtain all derivatives simultaneously is less than the time required when they are obtained individually. This section describes our approach in detail.

The U, V, W, P and R components on each plane are stored as shown in Fig. 1. It is necessary to store each component in a contiguous run because these components are required individually for computation elsewhere in the program. ALL, the vector made up of the concatenation of vectors U, V, W, P and R is thus of length 5120, the individual components being of length 1024 each.

The straightforward way of obtaining X-derivatives, given the data organization described in Fig. 1 is as follows:

```
    Subroutine DX:
    take 32 FFTs of length 32 each in the X-direction;
    multiply by wave numbers $\sqrt{-1}\, k_x$;
    take 32 inverse FFTs of length 32 each in the X-direction;
    return;
    -------------------
for each component do Call DX;
```

We have available to us the powerful FFT routine developed for the STAR by Korn and Lambiotte[5]. An important property of this routing is that the time per transform decreases

Table 1. Selected vector instruction timings (minor cycles) for the STAR[1–4]

INSTRUCTION	TIMING FORMULA (N = Vector length)
Move	91 + N/2
Compress	92 + N
Merge	123 + 3N
Multiply	157 + N
Transpose (Matrix of size $\sqrt{N} \times \sqrt{N}$)	16.6N

```
 ────────────► Y
│
│  0  U[ 0, 0]  U[ 0, 1] . . . U[ 0,31]        U
│     U[ 1, 0]  U[ 1, 1] . . . U[ 1,31]
│
│       .          .      . . . .
│       .          .      . . . .
▼       .          .      . . . .
X     U[31, 0]  U[31,1(  . . . U[31,31]      1023
                                                          STORAGE IS
                                                          CONSECUTIVE BY ROWS.
1024  V[ 0, 0]  V[ 0, 1] . . . V[0,31]         V
      V[ 1, 0]  V[ 1, 1] . . . V[1,31]

        .          .      . . . .
        .          .      . . . .
        .          .      . . . .
      V[31, 0]  V[31,1]  . . . V[31,31]      2047

2048  W[ 0, 0]  W[ 0, 1] . . . W[ 0,31]        W
      W[ 1, 0]  W[ 1, 1] . . . W[ 1,31]

        .          .      . . . .
        .          .      . . . .
        .          .      . . . .
      W[31, 0]  W[31, 1] . . . W[31,31]      3071

3072  P[ 0, 0]  P[ 0, 1] . . . P[ 0,31]        P
      P[ 1, 0]  P[ 1, 1] . . . P[ 1,31]

        .          .      . . . .
        .          .      . . . .
        .          .      . . . .
      P[31, 0]  P[31, 1] . . . P[31,31]      4095

4096  R[ 0, 0]  R[ 0, 1] . . . R[ 0,31]
                                               R
      R[ 1, 0]  R[ 1, 1] . . . R[ 1,31]

        .          .      . . . .
        .          .      . . . .
        .          .      . . . .
      R[31, 0]  R[31, 1] . . . R[31,31]      5119
```

Fig. 1. Storage of the five components *U*, *V*, *W*, *P*, *R* in one vector ALL.

with the number of transforms being taken. This reduction is due in part to the pipeline architecture of the STAR and in part to ingeneous algorithm design whereby the initial overhead of taking several transforms is distributed over all transforms. Korn and Lambiotte describe[5] how the algorithm can lead to substantial reductions in time even on a scalar machine.

This routine requires that data be passed to it such that, if *M* transforms of N points each are required, the first *M* elements of the input vector contain the first points of each of the *N* input lines and so on.

Referring to Fig. 1 we see that our data is set up such that all the FFTs in the *X*-direction can be taken simultaneously.

Figure 2 shows the measured time per transform for obtaining sets of FFTs with the Korn-Lambiotte routine.

Note. Since our input data is all real, we use the well-known technique of presenting two rows of real data to the routine as real and imaginary parts of one row of complex data. Our input vector thus appears to the FFT routine to be 16 rows of complex data of length 32 each.

The measured time to transform 16 complex vectors of length 32 each is about 1640 μsec. A

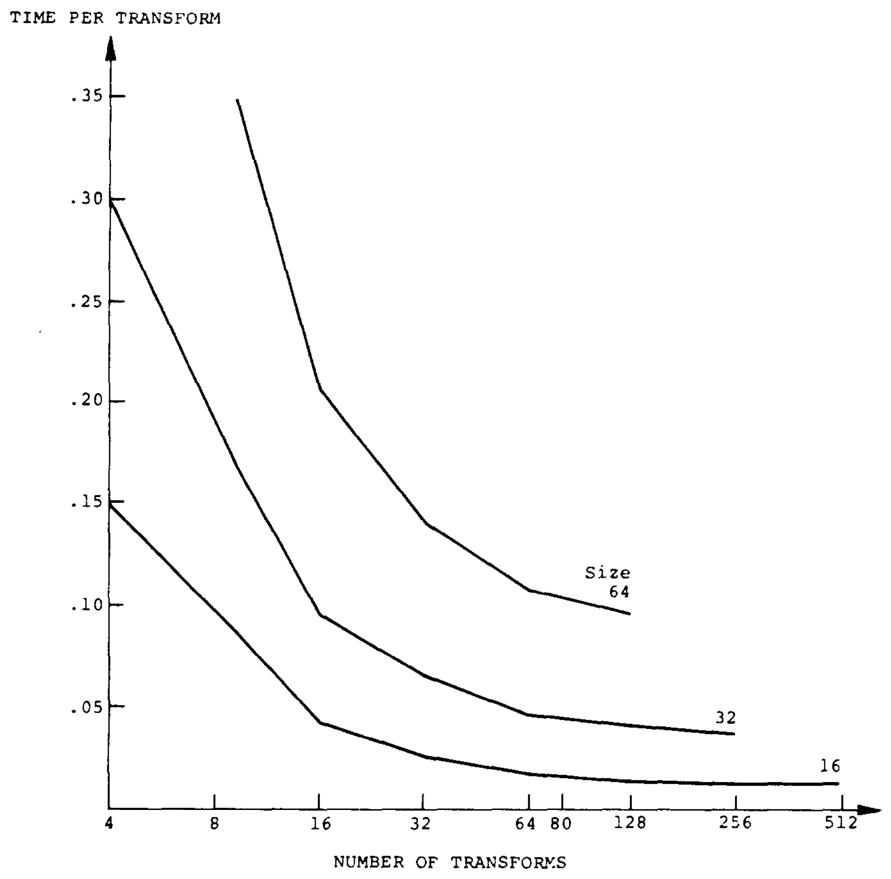

Fig. 2. Measured times per transform for obtaining sets of FFTS of various sizes using the Korn–Lambiotte algorithm.

forward and a reverse transform are required for each derivative. The time required to multiply the transform by $\sqrt{(-1)}\,k_x$ plus some additional overhead is 365 μsec, giving a total of 3625 μsec per derivative. The time for 5 derivatives is thus 18,125 msec. This information is tabulated in Table 2.

To obtain derivatives in the Y direction on each component we need to transpose the data so that it may be presented to the FFT routine in the required format. The following procedures accomplish this.

```
Subroutine DY;
transpose input vector of length 32 × 32;
call DX;
transpose vector of length 32 × 32;
return;
---------------
for each component do Call DY;
```

Although a special vector instruction for transposing is available on the STAR, the overhead is quite substantial (almost 30%) as shown in Table 2.

We have eliminated the inefficiency inherent in this approach by using perfect shuffles [6] to preprocess the input data. The technique we have developed reverse shuffles the vector ALL so that it appears to the FFT routine to be a collection of 80 complex vectors of length 32 each.

After transforming, multiplying by wave numbers and inverse transforming, the output vector is forward shuffled back to its original configuration. By taking all transforms at once, we take full advantage of the Korn–Lambiotte FFT routine. This more than compensates for the

Table 2. Measured execution times (μs) for the three derivative routines

Routine	FFT Size	Time for 2 FFTs	Multiplication + misc	2 Transposes	2 pseudo-transposes (10 shuffles)	Time for 1 derivative	Time for 5 derivatives
DX	16 × 32	3277	365	-----	-----	3625	18125
DY	16 × 32	3270	307	1021	-----	4598	22990
DS(X)	80 × 32	7225	1386	-----	7614	----	16225
DS(Y)	80 × 32	6931	1151	-----	7613	----	15695

overhead of shuffling and the total time for obtaining all X derivatives is 10% less than the time required if they are done separately. Similarly the time for Y derivatives is 32% less. We call this shuffle based routine *DS* and Table 2 includes its timings when used for X and Y derivatives.

Figure 3 illustrates the reverse shuffling required for the case of X-derivatives. The shuffling is done a row at a time (each element in the shuffle is a row of ALL). After 5 reverse shuffles the vector ALL assumes the form shown on the right hand side of Fig. 3. Here we have the zeroth rows of U, V, W, P, R followed by the first row, etc. This shuffled vector is presented to the FFT routine as a collection of 80 complex vectors of length 32 each.

To obtain Y derivatives we shuffle an element at a time to have the first column of U followed by the first column of V etc. This also takes five shuffles and permits 80 complex transforms of length 32 in the Y direction to be taken simultaneously.

Shuffles are done on the STAR using the Compress and Merge instructions with appropriate control vectors. The speed of these instructions varies slightly with the composition of the control vector. Different control vectors are required for row at a time and element at a time shuffles. This accounts for the fact that when used for Y derivatives, routine *DS* is slightly faster than when used for X derivatives.

Details of routine *DS* are as follows.

```
Subroutine DS;
loop $\log_2(32)$ times; Reverse Shuffle ALL endloop;
take $5 \times 32/2$ transforms of length 32 each;
multiply by wave numbers;
take $5 \times 32/2$ transforms of length 32 each;
loop $\log_2(32)$ times: Forward Shuffle ALL endloop/return;
```

4. THEORETICAL PREDICTIONS OF RUNTIME

In this section we discuss generalizations of the algorithms described in Section 3. We analyze the time required to obtain derivatives of m components on $n \times n$ planes. We use

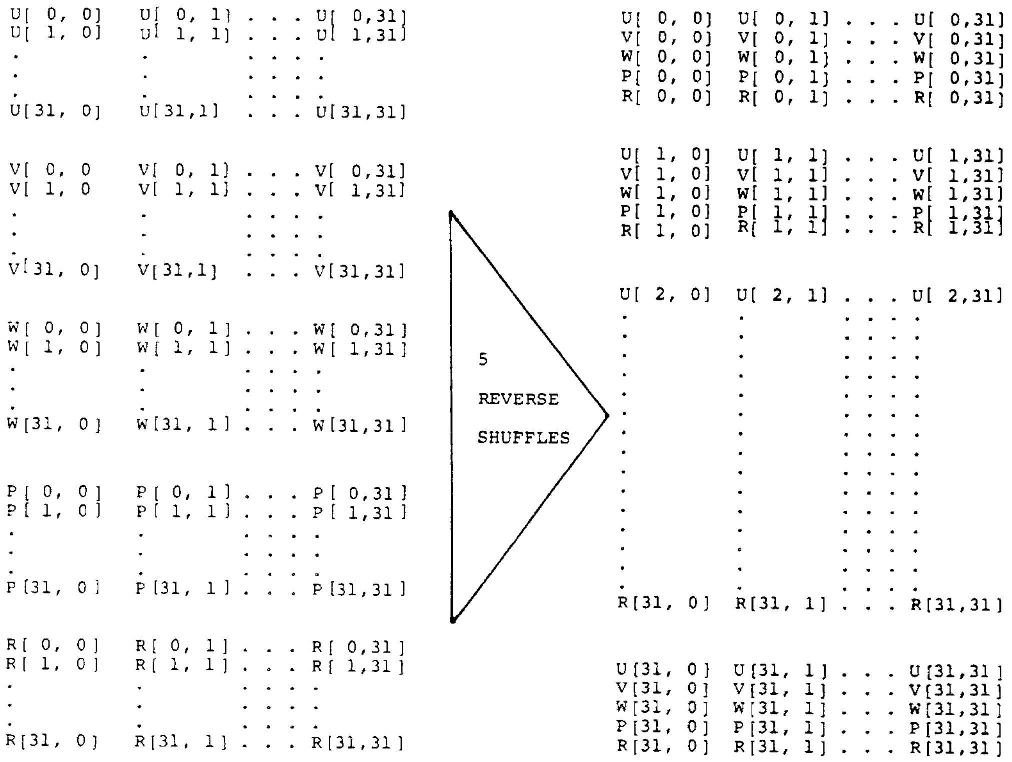
U[0, 0] U[0, 1] . . . U[0,31]
U[1, 0] U[1, 1] . . . U[1,31]
U[31, 0] U[31,1] . . . U[31,31]
V[0, 0 V[0, 1] . . . V[0,31]
V[1, 0 V[1, 1] . . . V[1,31]
V[31, 0] V[31,1] . . . V[31,31]
W[0, 0] W[0, 1] . . . W[0,31]
W[1, 0] W[1, 1] . . . W[1,31]
W[31, 0] W[31, 1] . . . W[31,31]
P[0, 0] P[0, 1] . . . P[0,31]
P[1, 0] P[1, 1] . . . P[1,31]
P[31, 0] P[31, 1] . . . P[31,31]
R[0, 0] R[0, 1] . . . R[0,31]
R[1, 0] R[1, 1] . . . R[1,31]
R[31, 0] R[31, 1] . . . R[31,31]
5
REVERSE
SHUFFLES
U[0, 0] U[0, 1] . . . U[0,31]
V[0, 0] V[0, 1] . . . V[0,31]
W[0, 0] W[0, 1] . . . W[0,31]
P[0, 0] P[0, 1] . . . P[0,31]
R[0, 0] R[0, 1] . . . R[0,31]
U[1, 0] U[1, 1] . . . U[1,31]
V[1, 0] V[1, 1] . . . V[1,31]
W[1, 0] W[1, 1] . . . W[1,31]
P[1, 0] P[1, 1] . . . P[1,31]
R[1, 0] R[1, 1] . . . R[1,31]
U[2, 0] U[2, 1] . . . U[2,31]
R[31, 0] R[31, 1] . . . R[31,31]
U[31, 0] U[31, 1] . . . U[31,31]
V[31, 0] V[31, 1] . . . V[31,31]
W[31, 0] W[31, 1] . . . W[31,31]
P[31, 0] P[31, 1] . . . P[31,31]
R[31, 0] R[31, 1] . . . R[31,31]

Fig. 3. Vector ALL is reverse shuffled five times to permit all FFTs in the X direction to be taken simultaneously.

available STAR vector instruction timing data to obtain expressions for run times for algorithms *DS*, *DY* and *DX*.

Timing formulae for STAR vector instruction of interest to us are given in Table 1. It is to be emphasized that a number of subtle factors such as the exact placement of vectors in memory, composition of control vectors, etc. influence the run times and are difficult to account for. In particular the Compress and Merge instructions may require up to 20% more time than that predicted by the given formulae.

The timing formula for the Korn–Lambiotte FFT routine is taken to be $950N + 3.5MN\log_2 N$ for M transforms of length N each[5].

We give below our generalization of algorithm *DS* for taking derivatives of m components on a plane of size $n \times n$. Next to each statement of this algorithm is given the corresponding STAR vector instruction and the number of cycles required. We ignore in this analysis the few scalar instructions that are required for loop control and vector setup functions.

ALGORITHM *DS*

Operation	Instruction	Cycles
1. Copy input vector ALL into SCRATCH_1. (Note A)	Move (Note B)	$91 + mn^2/2$
2. Reverse Shuffle SCRATCH_1 $\log_2 n$ times (Note C)	2 Compresses (Note D)	$2\log_2 n(92+mn^2)$
3. Copy SCRATCH_1 into SCRATCH_2. (Note E)	Move	$91 + mn^2/2$
4. Take mn/2 FFTs of length n each on SCRATCH_2.	(FFT routine)	$950n + 1.75mn^2\log_2 n$
5. Multiply by wave numbers	Multiply	$157 + mn^2$
6. Multiply by $\sqrt{-1}$ = interchange alternative elements	2 Moves	$2(91+mn^2/2)$
7. Take mn/2 inverse FFTs. of length n each on SCRATCH_2.	(FFT routine)	$950n + 1.75mn^2\log_2 n$
8. Forward Shuffle SCRATCH_2 $\log_2 n$ times	Merge	$\log_2 n(123 + 3mn^2)$

Notes.

A. It is necessary to copy the input vector to keep from destroying it.

B. All vector lengths are mn^2 except in the FFT routine.

C. When taking X-derivatives the shuffling is row at a time. For Y derivatives it is element at a time.

D. Each Reverse Shuffle requires two compresses.

E. This instruction is necessitated by the way our program is set up and it may be possible to eliminate it.

The total cycles for routine *DS* are

$$521 + 1900n + 3mn^2 + \log_2 n(307 + 8.5\, mn^2).$$

Details of routine *DY* (for obtaining derivatives on *one* component in the *Y* direction) are as follows.

ALGORITHM *DY*

Operation		Instruction	Cycles
1.	Copy input vector	Move	$91+n^2/2$
2.	Transpose	Transpose	$16.6n^2$
3.	Take n/2 FFTs of length n each	(FFT routine)	$950n + 1.75n^2\log_2 n$
4.	Multiply by wave numbers	Multiply	$157 + n^2$
5.	Multiply by $\sqrt{-1}$	2 Moves	$2(91 + n^2/2)$
6.	Take n/2 FFTs of length n each	(FFT routine)	$950\ n + 1.75n^2\log_2 n$
7.	Transpose	Transpose	$16.6n^2$

The total cycles are $430 + 1900 + n^2\,(35.7 + 3.5\log_2 n)$. When derivatives of m components are required the toral cycles for *DY* are

$$m(430 + 1900n + n^2(35.7 + 3.5\log_2 n)).$$

Total cycles for algorithm *DX*, which does not require the transposes at steps 2 and 7 above are

$$m(430 + 1900n + n^2(2.5 + 3.5\log_2 n)).$$

Thus all three algorithms have $0(mn^2\log_2 n)$ complexity. The asymptotic run time of *DX* is the best, followed by *DY* and then *DS*. However for most problem sizes of interest the run time of *DS* is better than *DY* as discussed in the following section.

5. DISCUSSION

Run times for the three algorithms computed from the formulae derived in the previous section are given in Table 3. This table shows predicted run times for *DS* in microseconds for m (number of components) varying from 1 to 10 and n (size of plane) varying from 16 to 256. These are the problem sizes likely to be of interest in practice. Below each entry are given comparisons with *DX* and *DY*. For example, the entry for $m = 6$, $n = 128$ gives the predicted runtime for *ds* to be 255594 μs. This figure expressed as a fraction of the time required by the routine *DX*(*DY*) is 1.57 (0.87).

Table 3 has been divided into three parts. In the leftmost portion (for m greater than 1 and $n = 16, 32$) routine *DS* is faster than either *DX* or *DY*. For $n = 64$ and 128, *DS* is slower than *DX* but faster than *DY*. For $n = 256$, *DS* is poorer than both *DX* and *DY*.

We can make the following observations about Table 3.

(1) Our shuffle based routine is useful over most of the range of values of n in the table. For $n = 16$ or 32, *DS* can replace both the conventional routines *DX* and *DY*. For $n = 64$ and 128, *DS* can be used instead of *DY*.

(2) Vector lengths on the STAR are limited to 65536 words. This means that the routine *DS* (as implemented by us) is constrained to problems in which mn^2 is less than 65536. Problem sizes that are too large are enclosed by a dashed boundary in Table 3. We see that it will not be possible to use *DS* for any problem in which $n = 256$ or more.

(3) The vector length limitation means that for $n = 128$ we cannot have m greater than 4. However in this case the following approach can be taken. If a problem with $n = 128$ has m greater than 4, the derivatives can be taken in groups of no more than 4 components each. Since *DS* is superior to *DY* for all m less than 5, we can be sure that this approach is always better than using *DY* for any m.

(4) The relative speed of *DS* improves with increasing m. This suggests an interesting

Table 3. Predicted execution times (μs) for the routine *DS* and comparisons with *DX* and *DY*

	n=16	n=32	n=64	n=128	n=256
m = 1	1664	4377	13805	50794	205697
	1.19X	1.34X	1.59X	1.86X	2.07X
	0.96Y	0.95Y	0.97Y	1.03Y	1.10Y
2	2043	6241	22653	91754	391819
	0.73X	0.96X	1.30X	1.68X	1.98X
	0.59Y	0.67Y	0.80Y	0.93Y	1.05Y
3	2422	8105	31500	132714	577941
	0.58X	0.83X	1.21X	1.62X	1.94X
	0.46Y	0.58Y	0.74Y	0.90Y	1.03Y
4	2801	9968	40347	173674	764063
	0.50X	0.77X	1.16X	1.59X	1.93X
	0.40Y	0.54Y	0.71Y	0.88Y	1.02Y
5	3180	11832	49195	214634	950185
	0.46X	0.73X	1.13X	1.57X	1.92X
	0.37Y	0.51Y	0.69Y	0.87Y	1.02Y
6	3559	13696	58042	255594	1136307
	0.43X	0.70X	1.11X	1.56X	1.91X
	0.34Y	0.49Y	0.68Y	0.87Y	1.02Y
7	3938	15559	66889	296554	1322429
	0.41X	0.69X	1.10X	1.55X	1.91X
	0.32Y	0.48Y	0.67Y	0.86Y	1.01Y
8	4316	17423	75737	337514	1508551
	0.39X	0.67X	1.09X	1.54X	1.90X
	0.31Y	0.47Y	0.67Y	0.86Y	1.01Y
9	4695	19287	84584	378474	1694673
	0.38X	0.66X	1.08X	1.54X	1.90X
	0.30Y	0.46Y	0.66Y	0.85Y	1.01Y
10	5074	21150	93431	419434	1880795
	0.37X	0.65X	1.08X	1.53X	1.90X
	0.29Y	0.46Y	0.66Y	0.85Y	1.01Y
				Vector Length Limitation	
	DS < DX < DY		DX < DS < DY		DX < DY < DS

Table 4. Time per derivative using routine *DS* for the maximum number of components '*m*' for any '*n*'

n	Maximum m	time per derivative	Relative Speed
128	4	43413	1.59X, 0.88Y
64	16	9157	1.05X, 0.68Y
32	64	1902	0.59X, 0.41Y
16	256	384	0.27X, 0.22Y

possibility for further speedup. If a problem has, say, 5 components, we can improve performance by simply taking the derivatives on 2, 3 or more planes simultaneously, thereby effectively increasing m from 5 to 10, 15 or more. In this case it would be necessary for the vectors containing data for each plane to be stored contiguously. If this be so then the only other constraint is the vector length limitation discussed above. Table 4 shows the maximum speedup that is possible using this approach.

(5) The predicted run times for $n = 32$, $m = 5$ do not agree very exactly with the measured run times of Section 3. There are a number of reasons for this variation. Firstly, our theoretical predictions take into account only vector operations. Scalar instructions, vector setup time and the overhead of subroutine calls have been ignored. Secondly, there are a number of subtle factors that influence vector instruction timing such as the exact starting location of the vector, memory conflicts, composition of control vectors, etc. that are difficult (if not impossible) to account for. Finally the run time formula for the Korn–Lambiotte routine is also very

approximate. Nevertheless the theoretical predictions of Table 3 are useful as guidelines that can lead to substantial improvements in practice, as has been verified by our experience.

It would be very interesting to repeat this analysis for the new CDC supercomputers—the Cyber 203s and 205s. This may be done as soon as vector instruction timing data is available for these machines. However, since 203 has essentially the same vector hardware as the STAR 100, only slight differences (if any) in performance will probably be found for this machine.

Korn and Lambiotte describe how their FFT routine leads to considerable improvement in run time even on a scalar computer, when several transforms of the same length are taken. It thus seems possible that a suitable modification of the approach described in our paper would lead to improvements even on a scalar computer.

Acknowledgements—We would like to thank Drs. E. Rose, R. J. Voigt for discussions and Drs. J. Knight, and J. Lambiotte and Ms. L. Hauser for technical assistance with this work.

REFERENCES

1. Control Data, *STAR-100 Computer Hardware Reference Manual.* Revision 9, 15 Dec. (1975).
2. Control Data, *STAR-100 Preliminary Instruction Execution Timing Manual.* 15 Jan. (1975).
3. H. Nelson, *Star Instruction Times.* Lawrence Livermore Laboratory, 16 Jan. (1976).
4. J. J. Lambiotte, personal communication.
5. D. G. Korn and J. J. Lambiotte, Computing the fast Fourier transform on a vector computer. *ICASE report No.* 78-5. 23 Feb. (1978).
6. H. S. Stone, Parallel processing with the perfect shuffle. *IEEE Trans. Compt.* **C-20**(2), 153–161 (1971).