Library

MET O 11 TECHNICAL NOTE NO 155

FAST REAL FOURIER TRANSFORMS ON THE CYBER 205

by

C Temperton

Met O 11
Meteorological Office
London Road
Bracknell
Berkshire, U.K.

March 1982

1. Introduction

The Fast Fourier Transform (FFT) algorithm is one of the most useful items in the computational physicist's toolkit. The algorithm (of which there are many variants) provides a fast way of implementing the Discrete Fourier Transform (DFT), given by

$$x_j = \sum_{k=0}^{N-1} c_k \exp(2ijk\pi/N) \qquad (1)$$

for $0 \leq j \leq N-1$, where x_j and c_k are complex numbers. Direct calculation of the sums in Eq (1) would require N(N-1) complex additions and N^2 complex multiplications. However, if N can be represented as a product of small integers, then the number of arithmetic operations can be dramatically reduced by the FFT technique. Table 1 gives examples for N = 96; as suggested, there is a distinct advantage in grouping the factors together, though this is obtained at the expense of increased program complexity. The inverse of Eq (1) is:

$$c_k = \frac{1}{N} \sum_{j=0}^{N-1} x_j \exp(-2ijk\pi/N) \qquad (2)$$

which can be implemented in almost exactly the same way.

Table 1. Complex FFT: operation counts for N=96

	real adds	real mults
Direct	36672	36864
$N = 2^5 \times 3$	1762	964
$N = 4^2 \times 6$	1602	644

In many cases of interest the x_j's in Eqs (1) and (2) are real numbers, implying that the Fourier coefficients c_k must satisfy the relationships $c_{N-k} = c_k^*$ (* denotes complex conjugate). Eq (1) is then sometimes rewritten as

$$x_j = \sum_{k=0}^{N/2} \left\{ a_k \cos(2jk\pi/N) + b_k \sin(2jk\pi/N) \right\}$$

where $$a_0 = \mathrm{Re}(c_0) \quad , \quad a_{N/2} = \mathrm{Re}(c_{N/2}) \; ,$$

and $$a_k = 2\,\mathrm{Re}(c_k) \; , \quad b_k = -2\,\mathrm{Im}(c_k) \quad \text{for} \quad 0 < k < N/2 .$$

There are several ways of adapting the FFT algorithm to this case.

Several examples of the use of the FFT can be found in numerical weather prediction. The new operational model run on the Cyber 205 at the UK Meteorological Office is based on a regular latitude-longitude grid on the sphere. The major problem with such a grid is that the convergence of the meridians implies progressively shorter gridlengths in the zonal direction as the poles are approached, requiring a prohibitively short timestep for computational stability. The remedy adopted (Holloway, Spelman and Manabe, 1973; Williamson, 1976) is to Fourier filter the tendencies of the time-dependent variables poleward of a given latitude. These tendencies are expanded in terms of Fourier series which are then truncated at a given wavenumber (depending on latitude) and finally reconstituted at the gridpoints. This device gives a resolution in the zonal direction which remains effectively constant as the poles are approached, without sacrificing the regularity of the grid. Fourier filtering is most efficiently implemented via the FFT algorithm.

Further uses of the FFT can be found in other numerical weather prediction models. In the ECMWF gridpoint model for example, the use of Fourier filtering has been replaced by a form of implicit diffusion in the zonal direction, again most efficiently implemented via the FFT. This model also uses a semi-implicit time integration scheme which requires the solution at each timestep of a three-dimensional discrete elliptic equation. This can be decoupled into a set of two-dimensional Helmholtz equations over the sphere, which are solved by a direct method (fast elliptic solver) applicable to spherical geometry (Swarztrauber, 1974) together with a technique specially developed for large out-of-core problems (Burridge and

Temperton, 1979). Much of the work in the direct elliptic solver consists of FFT's.

The most extensive use of the FFT in numerical weather prediction is found in spectral models, where the prognostic variables are not gridpoint values but coefficients of spherical harmonics. All the nonlinear terms are evaluated by transforming the fields to an appropriate grid, computing products and transforming back to spectral space (Bourke, 1972; Orszag, 1971). The zonal part of the transformation to and from gridpoint space is performed using the FFT. ECMWF's quasi-operational T63* spectral model requires almost 30,000 real Fourier transforms per timestep, each of length N = 192 (Burridge, personal communication).

This paper reports on a real Fast Fourier Transform package implemented on the Cyber 205 for the new operational model. Section 2 briefly describes some mathematical aspects of the algorithm used, while Section 3 deals with the implementation on a vector computer, with particular reference to the Cyber 205. Timings are presented in Section 4, together with some comparisons with a similar package implemented on the Cray-1. Documentation of the package is included in Section 5.

2. Mathematical Characteristics

As mentioned in the Introduction, there are many variants of the FFT algorithm. The version used here can be briefly described as mixed-radix, self-sorting and real.

"Mixed-radix" refers to the fact that N can be represented as the product of several different factors, in contrast to the simplest and most restrictive case $N = 2^p$. The present package allows the case $N = 2^p 3^q 5^r$. In order to decrease the operation count, there is scope for grouping the factors together, so that in addition to the factors 2, 3 and 5 there are sections of coding for factors 4, 6 and 8. The provision for factor 6 is unusual, but it does lead to a worthwhile decrease in the amount of arithmetic.

* triangularly truncated at wavenumber 63; ~ 4000 degrees of freedom per horizontal field.

"Self-sorting" means that the input to and output from the algorithm are naturally ordered. In the FFT algorithm as originally presented by Cooley and Tukey (1965), and in many of its descendants, the data had to be either shuffled before entry to the transform, or unscrambled on exit. This kind of data shuffling can be an expensive overhead on a vector computer. Self-sorting variants have in fact been known for a long time; Cochran et al (1967) attribute the idea to Stockham. For a detailed derivation of self-sorting and other versions of the complex FFT algorithm, see Temperton (1977, 1982a).

Finally, the package described here implements the FFT algorithm for the case in which the x_j's of Eqs (1) and (2) are real numbers. Conventionally, the FFT is regarded as an algorithm for complex numbers, and the real case is handled either by forming an artificial complex series of length N/2 to which a complex FFT (of length N/2) can be applied (Method 1); or a complex FFT of length N can be used to perform two real Fourier transforms of length N simultaneously (Method 2). The operation counts for these two methods are almost identical (see Table 2).

An alternative approach, first suggested by Bergland (1968) for the case $N = 2^P$ but subsequently rather neglected, uses the complex transform of length N more directly. If $c_{N-k} = c_k^*$ then there are only N real degrees of freedom in the input data for Eq (1); the same is true for the output data (all the x_j's are real). In fact it can be shown that the same is true at each stage of the algorithm; every number in the array is either real, or else accompanied by its complex conjugate somewhere else in the array (Temperton, 1982c). We can adapt the complex FFT of length N directly to the real case by "pruning out" redundant operations (i.e. on zero imaginary parts, or complex conjugates of operations already performed elsewhere). This leads to a rather more complicated algorithm, but also to a significantly reduced operation count, typically 25-30% less than the conventional approach. Some examples for N = 192 are given in Table 2.

The first two lines represent typical black-box FFT packages (assuming they rise above the restriction $N = 2^p$), while the last line represents the package described here.

Table 2. Real FFT: operation counts for N = 192

	real adds	real mults
Method 1, cplx $N/2 = 2^5 \times 3$	2236	1152
Method 2, cplx $N = 2^6 \times 3$	2239	1154
Method 1, cplx $N/2$ $4^2 \times 6$	2076	832
Method 2, cplx $N = 2 \times 4^2 \times 6$	2079	834
Real, $N = 3 \times 2^6$	1764	964
Real, $N = 3 \times 4^3$	1698	802
Real, $N = 4 \times 6 \times 8$	1654	694

A program to implement Eq (1) via the FFT algorithm will normally include provision for implementing the inverse transform, Eq (2), instead. With the conventional approaches this requires minimal extra programming. One disadvantage of the approach adopted here for the special real case is that separate programs are required for Eqs (1) and (2). Operation counts are the same for both forward and inverse transforms.

3. Vectorization

In order to appreciate the vectorization scheme for the FFT package on the Cyber 205, it is necessary to look at the structure of the self-sorting complex

FFT routine as it might be implemented on a scalar computer. There is one pass through the data for each factor of N, and two arrays are used alternately as input and output for successive passes. The indexing within each pass is controlled by the value of the current factor of N, and by the variable LA which takes the value 1 during the first pass, and on completion of each pass is multiplied by the current factor. Suppose for example that the factors of N are listed in the array IFAX(1) to IFAX(NFAX). The outermost structure is then as follows:

```
      COMPLEX A(N), C(N)
      INTEGER IFAX(NFAX)
      LA=1
      DO 1Ø I=1, NFAX
      IFAC = IFAX(I)
      CALL PASS(A,C, IFAC, N, LA)
      LA= LA*IFAC
      (now reverse roles of arrays A and C)
   1Ø CONTINUE
```

The subroutine PASS has a nested structure:

```
      SUBROUTINE PASS (A,C, IFAC, N, LA)
      COMPLEX A(N), C(N)
      M= N/IFAC
      I=Ø
      J=Ø
      JUMP = (IFAC-1)* LA
      DO 2Ø K=Ø, M-LA, LA
      DO 1Ø L=1, LA
      C (J) =Ω(K) * W (IFAC) * A (I)
      I = I + 1
      J = J + 1
   1Ø CONTINUE
      J = J + JUMP
   2Ø CONTINUE
      RETURN
      END
```

In the inner loop, W(IFAC) represents the DFT matrix of order IFAC, Ω (K) is a diagonal matrix of complex numbers (fixed during the loop), $\underset{\sim}{A}$ (I) and $\underset{\sim}{C}$ (J) are vectors of length IFAC, of the form

$$\underset{\sim}{A}(I) = \begin{bmatrix} A(IA+I) \\ A(IB+I) \\ \vdots \end{bmatrix} , \quad \underset{\sim}{C}(J) = \begin{bmatrix} C(JA+J) \\ C(JB+J) \\ \vdots \end{bmatrix}$$

where IA, IB, JA, JB etc are base addresses fixed for the duration of the pass. In practice the inner loop would of course be expanded into real scalar arithmetic, using various tricks to speed up the multiplication by W(IFAC), and taking advantage of the fact that the first element of Ω(K) is always 1, and Ω(K) is the identity matrix for K=0.

The salient points are that the inner loop vectorizes naturally with an increment of 1 between successive elements of vectors* (I = I + 1, J = J + 1; note that the vectors we are considering now should not be confused with the mathematical vectors $\underset{\sim}{A}$ (I) and $\underset{\sim}{C}$ (J), used here only for compactness of notation), and that the vector length is LA. This scheme was referred to by Temperton (1979) as vectorization scheme A, and is illustrated schematically in Fig 1. The vector

* Assuming here that the real and imaginary parts of complex numbers have been stored in separate arrays.

length is unfortunately never very long; for example if N = 96 = 4 x 4 x 6 the vector length is 1, 4, 16 on successive passes.

On Cray-1 (though not on the Cyber 205), the elements of a vector do not have to be contiguously stored, but can be spaced at any constant increment. This permits the nested loop structure of subroutine PASS to be "turned inside out" (vectorization scheme B), giving instead a vector length of N/(IFAC*LA). The case N = 96 = 4 x 4 x 6 can then be implemented with vector lengths 24, 6, 16 on successive passes, by choosing the most appropriate structure at each stage. Even on the Cray-1, which reaches half maximum speed with vector lengths as short as 10 or thereabouts, this is some way from being optimal. On the Cyber 205, which requires vector lengths of order 100 to reach half maximum speed, this would be much too slow even if the data could be reorganized to give contiguous vectors with the "inside-out" loop structure.

The problem of vectorizing the FFT on the Cray-1 was studied in detail by Temperton (1979). The simplest solution stems from the fact that we usually need to perform many transforms simultaneously. It is then an easy matter to arrange the data so that the transforms are performed in parallel, with each vector containing one element from each transform, and the vector length equal to the number of transforms being performed simultaneously. The details of the indexing are transferred to outer loops, and become irrelevant from the standpoint of vectorization. This multiple vectorization scheme for Cray-1 is illustrated in Fig 2. As the <u>optimum</u> vector length is 64, and we can usually arrange for the number of simultaneous transforms to be of that order or even greater, the problem was considered solved for that machine.

On the Cyber 205 the situation is somewhat different. First, the requirement that vector elements be contiguous means that the data organization of Fig 2 must be transposed (see Fig 3; this reorganization would also work on Cray-1). Second, the efficiency continues to improve as vector lengths increase (up to a maximum of 64K-1), and vector lengths should preferably be at least of order several hundred, typically rather more than the number of transforms which can conveniently be performed in parallel. Fortunately, the transposed data organization as in Fig 3 allows us to apply the "direct product" of the first vectorization scheme (A) referred to above and the multiple vectorization scheme. The vector length becomes LA*LOT, where LOT is the number of transforms being performed together. The situation is illustrated in Fig 4. Again, this procedure could also be applied on Cray-1, but there would be little gain unless the number of transforms was much less than 64.

The preceding discussion has referred throughout to complex Fourier transforms, but the situation is almost exactly the same for the specialization to the real case.

It is worth noting that Korn and Lambiotte (1979), addressing the problem of vectorizing multiple complex FFT's on vector computers (particularly the Cyber 205's ancestor, the CDC STAR-100), suggested a scheme similar to that adopted here. Later writers (Fornberg, 1981; Wang, 1980) have suggested a somewhat different approach in order to achieve a vector length of (N*LOT)/IFAC throughout. Their schemes essentially replace $\underset{\sim}{C}$ (J) by $\underset{\sim}{C}$ (I) in the skeleton Fortran routine PASS, and follow each call to PASS with an explicit permutation of the data using scatter/gather or merge/compress techniques. However, recent analysis (Temperton, 1982b) shows that this approach is slower than the present proposal on the Cyber 205; the longer vectors are outweighed by the time taken for data permutation, not to mention the need to create and store enormous arrays of trigonometric function values and bit strings.

There remain a few technical points concerning the details of vectorization in the Cyber 205 FFT package. In the real Fourier transform algorithm, real and imaginary parts of complex numbers are interleaved in such a way that they can frequently be combined into a single vector, thus further doubling the vector length. Also, triadic operations of the form $\underset{\sim}{a} = b * (\underset{\sim}{c} + \underset{\sim}{d})$ or

$\underset{\sim}{a} = \underset{\sim}{b} + c * \underset{\sim}{d}$ can be performed in almost the same time as a single vector addition or multiplication, using the vector "link" instruction (note that one of the operands must be a scalar). This is analogous to the (more flexible) operation of "chaining" on Cray-1. Since the FFT algorithm contains more additions than multiplications, the ideal situation would be for all multiplications to be linked with additions, so that the total time depended effectively only on the number of additions, with multiplications being implemented free of charge. With a little ingenuity, this ideal can be realized.

The conventional coding for factor 5 contains several pairs of statements of the form

$$\underset{\sim}{y} = a * \underset{\sim}{b} + c * \underset{\sim}{d}$$

$$\underset{\sim}{z} = \underset{\sim}{x} + \underset{\sim}{y}$$

Since a and c are prescribed constants, this can be rewritten in triadic form as

$$\underset{\sim}{y}' = \underset{\sim}{b} + (c/a) * \underset{\sim}{d}$$

$$\underset{\sim}{z} = \underset{\sim}{x} + a * \underset{\sim}{y}'$$

Throughout the FFT algorithm, complex numbers are added together or subtracted and then multiplied by a complex number of the form $e^{i\theta} = \cos\theta + i \sin\theta$, where $0 < \theta < \pi$. This requires a computation of the form:

(real part) $\underset{\sim}{w} = \underset{\sim}{a} + \underset{\sim}{b}$

(imag. part) $\underset{\sim}{x} = \underset{\sim}{c} + \underset{\sim}{d}$

(real part) $\underset{\sim}{y} = \cos\theta * \underset{\sim}{w} - \sin\theta * \underset{\sim}{x}$

(imag. part) $\underset{\sim}{z} = \sin\theta * \underset{\sim}{w} + \cos\theta * \underset{\sim}{x}$

This sequence can be rewritten in triadic form as:

$$\underset{\sim}{w}' = \cos\theta * (\underset{\sim}{a} + \underset{\sim}{b})$$

$$\underset{\sim}{x}' = \cos\theta * (\underset{\sim}{c} + \underset{\sim}{d})$$

$$\underset{\sim}{y} = \underset{\sim}{w}' - \tan\theta * \underset{\sim}{x}'$$

$$\underset{\sim}{z} = \tan\theta * \underset{\sim}{w}' + \underset{\sim}{x}'$$

The only snag is that tan θ is infinite for $\theta = \pi/2$, and in fact to minimize rounding error the above sequence should be reorganized using sin θ and cot θ for $\pi/4 < \theta < 3\pi/4$. The FFT package implemented on the Cyber 205 includes both options in each case, and the appropriate path is taken depending on the value of θ.

4. Timings

The real FFT package for the Cyber 205 was first written in straightforward vector Fortran, and subsequently translated into "special call" format using Q8 calls for all vector instructions and descriptor manipulations. This gave some speed advantage at short vector lengths, but the main reason was to permit the use of 32-bit arithmetic before a suitable compiler became available. The timing information presented here refers to the special call version, and to the forward transform, Eq (1). The inverse transform, Eq (2), runs slightly slower due to some multiplications which could not readily be linked with additions in this case.

Comparative figures are given for an FFT package written at ECMWF and run on the Cray-1 computer; the mathematical aspects of the FFT algorithm used are the same as for the Cyber 205, while vectorization was by the simple multiple scheme as discussed in the previous section. It should be noted that the Cray-1 package was written in CAL (Cray Assembly Language), and runs about twice as fast as a corresponding vectorized Fortran version. 32-bit arithmetic is not available on Cray-1.

Table 3 presents the time per real transform (in microseconds) for three values of N, and for four values of LOT, the number of transforms being performed together. Results for N = 180, 192 and 200 are shown to demonstrate that the choice of N is not as critical as is sometimes thought for efficiency of the FFT. Values of LOT = 16, 64, 256 and 1024 are shown to demonstrate the increasing efficiency of long vectors on the Cyber 205, in contrast to the Cray-1 where there is no advantage in increasing the vector length beyond 64.

Notice that 32-bit arithmetic is asymptotically twice as fast as 64-bit arithmetic on the Cyber 205, but because the start-up times for each vector operation (and other overheads) remain the same, many transforms must be performed together before this ratio is approached.

Some timing results (not shown) were also obtained on the Cyber 205 with the vector link instruction suppressed. If many transforms are performed together, the times are then about 50% longer, as expected from the ratio of multiplications to additions in the algorithm. For fewer simultaneous transforms, the effects of suppressing the link instruction are again less marked.

Table 3. Time per transform in μs

N	LOT	Cyber 205	Cyber 205	Cray-1
		(64-bit)	(32-bit)	(64-bit)
	16	87	78	36
180	64	35	26	26
(5×6^2)	256	22	13	26
	1024	18	10	26
	16	81	72	36
192	64	33	24	25
$(4 \times 6 \times 8)$	256	21	12	25
	1024	18	9	25
	16	98	88	41
200	64	40	30	30
$(5^2 \times 8)$	256	25	15	30
	1024	21	11	30

Comparing times on the Cyber 205 with those on Cray-1, at LOT = 16 the Cray-1 is markedly faster. At LOT = 64, 32-bit arithmetic on the Cyber matches 64-bit arithmetic on the Cray. By LOT = 256, 64-bit arithmetic on the Cyber beats that on the Cray, while at LOT = 1024 32-bit arithmetic on the Cyber is almost three times as fast as 64-bit arithmetic on the Cray.

The performance of supercomputers such as the Cray-1 and Cyber 205 is often measured in <u>megaflops</u>, or millions of floating-point instructions per second. For a computation in which there are as many multiplications as additions (and assuming on the Cyber 205 that they can all be linked), the machine architectures

Table 4. Megaflop rates for FFT packages

N	LOT	Cyber 205 (64-bit)	Cyber 205 (32-bit)	Cray-1 (64-bit)
	16	32	35	74
180	64	80	106	104
(5×6^2)	256	129	214	104
	1024	152	277	104
	16	32	35	71
192	64	78	105	100
$(4 \times 6 \times 8)$	256	124	207	100
	1024	145	274	100
	16	34	37	79
200	64	83	111	109
$(5^2 \times 8)$	256	132	220	109
	1024	155	293	109

impose maximum limits on the Cray-1 of about 145 megaflops, and on the (two-pipe) Cyber 205 of 200 megaflops for 64-bit arithmetic, or 400 megaflops for 32-bit arithmetic. For computations in which the mix of additions and multiplications is in the ratio 2:1, the corresponding limits are about 110, 150 and 300 megaflops respectively. Table 4 shows that these limits are very nearly achieved in the FFT packages, but that many transforms must be performed together on the Cyber 205 before maximum performance is approached.

5. Documentation of the Cyber 205 FFT package

This package performs multiple real transforms, defined by the following formulae:

a. Spectral to gridpoint transform, ISIGN = + 1:

$$x_j = \sum_{k=0}^{N-1} c_k \exp(2ijk\pi/N)$$

where $c_{N-k} = c_k^*$ (complex conjugate). c_0 and $c_{N/2}$ are real.

b. Gridpoint to spectral transform, ISIGN = -1:

$$c_k = \frac{1}{N} \sum_{j=0}^{N-1} x_j \exp(-2ijk\pi/N)$$

N must be factorizable in the form $N = 2^p 3^q 5^r$.

The Fourier coefficients $c_k = a_k + ib_k$ are stored in the order a_0, a_1, b_1, a_2, b_2, ..., a_{m-1}, b_{m-1}, a_m where m = N/2. (b_0 and b_m are zero and not stored).

Suppose the number of real transforms of length N to be performed together is LOT. Then the following arrays are required:

```
DIMENSION A(LOT*N), B(LOT*N), IFAX(1Ø), TRIGS(N)
```

Before any transforms are performed, the following setup routine should be called:

```
CALL SETFIL (TRIGS,IFAX,N)
```

SETFIL factorizes N and puts the factors into the array IFAX. The number of factors, NFAX, is put into IFAX(1), and the factors follow in IFAX(2) to IFAX(NFAX+1). An array of trigonometric function values required by the FFT package is set up in TRIGS. The call to SETFIL need only be made once for a given value of N.

The input data for the transforms is stored in A. The data vectors should be stored as rows of a columnwise matrix A(LOT,N), e.g. for ISIGN = + 1 successive storage locations should contain:

A(1,1) = a_o for transform 1

A(2,1) = a_o for transform 2

.

.

.

A(LOT,1) = a_o for transform LOT

A(1,2) = a_1 for transform 1

A(2,2) = a_1 for transform 2

.

.

.

etc

The transform routine is invoked by:

```
CALL FFT77 (A, B, TRIGS, IFAX, N, LOT, ISIGN)
```

The output from the routine is stored in the same format as was the input data. If NFAX is even, then the output is in the array A; if NFAX is odd, then the output is in the array B. In either case the input data is destroyed.

Warning: there is no check for illegally long vectors, which may be generated if (LOT*N) $\geqslant$ 128K.

The package exists as a set of source decks on the UPDATE PL CDEXB in pool P11. For the 64-bit version the required source decks are SETFIL, FFT77, RPASSQ and QPASSQ. If the input and output data are 32-bit numbers, then replace the last two routines by RPASSQH and QPASSQH. (N.B. The array TRIGS is still assumed to contain 64-bit numbers).

RPASSQ and QPASSQ are written in "special call" format; alternative versions using straightforward vector Fortran (slower but more easily understood) exist as RPASSM and QPASSM.

References

G. D. Bergland (1968): "A Fast Fourier Transform algorithm for real-valued series", Comm. ACM 11, 703-710.

W. Bourke (1972): "An efficient, one-level, primitive-equation spectral model", Mon. Wea. Rev. 100, 683-689.

D. M. Burridge and C. Temperton (1979): "A fast Poisson-solver for large grids", J. Comp. Phys. 30, 145-148.

W. T. Cochran et al (1967): "What is the Fast Fourier Transform?", IEEE Trans. Audio and Electroacoustics, AU-15, 45-55.

J. W. Cooley and J. W. Tukey (1965): "An algorithm for the machine calculation of complex Fourier series", Math. Comp, 19, 297-301.

B. Fornberg (1981): "A vector implementation of the Fast Fourier Transform algorithm", Math. Comp. 36, 189-191.

J. L. Holloway, M. J. Spelman and S. Manabe (1973): "Latitude-longitude grid suitable for numerical time integration of a global atmospheric model", Mon. Wea. Rev. 101, 69-78.

D. G. Korn and J. J. Lambiotte (1979): "Computing the Fast Fourier Transform on a vector computer", Math. Comp. 33, 977-992.

S. A. Orszag (1971): "Numerical simulation of incompressible flows within simple boundaries, I: Galerkin (spectral) representations", Studies in Applied Mathematics 50, 293-327.

P. N. Swarztrauber (1974): "The direct solution of the discrete Poisson equation on the surface of a sphere", J. Comp. Phys. 15, 46-54.

C. Temperton (1977): "Mixed-radix Fast Fourier Transforms without reordering", ECMWF Technical Report No 3.

C. Temperton (1979): "Fast Fourier Transforms and Poisson-solvers on Cray-1", in Supercomputers, Infotech State of the Art Report, Infotech International Ltd., Maidenhead, U.K.

C. Temperton (1982a): "Self-sorting mixed-radix Fast Fourier Transforms", in preparation.

C. Temperton (1982b): "Fast Fourier Transforms on the Cyber 205", in preparation.

C. Temperton (1982c): "Self-sorting mixed-radix real FFT's", in preparation.

H. H. Wang (1980): "On vectorizing the Fast Fourier Transform", BIT 20, 233-243.

D. L. Williamson (1976): "Linear stability of finite-difference approximations on a uniform latitude-longitude grid with Fourier filtering", Mon. Wea. Rev. 104, 31-41.

Figures

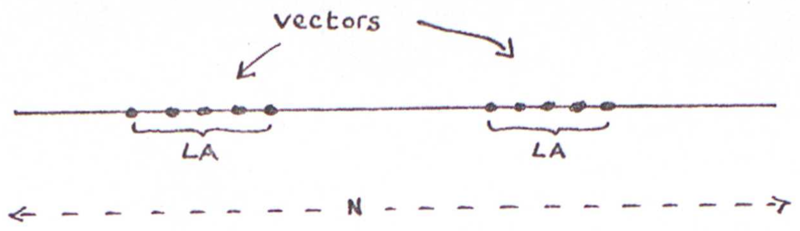

Fig 1: Vectorization scheme A for a single transform.

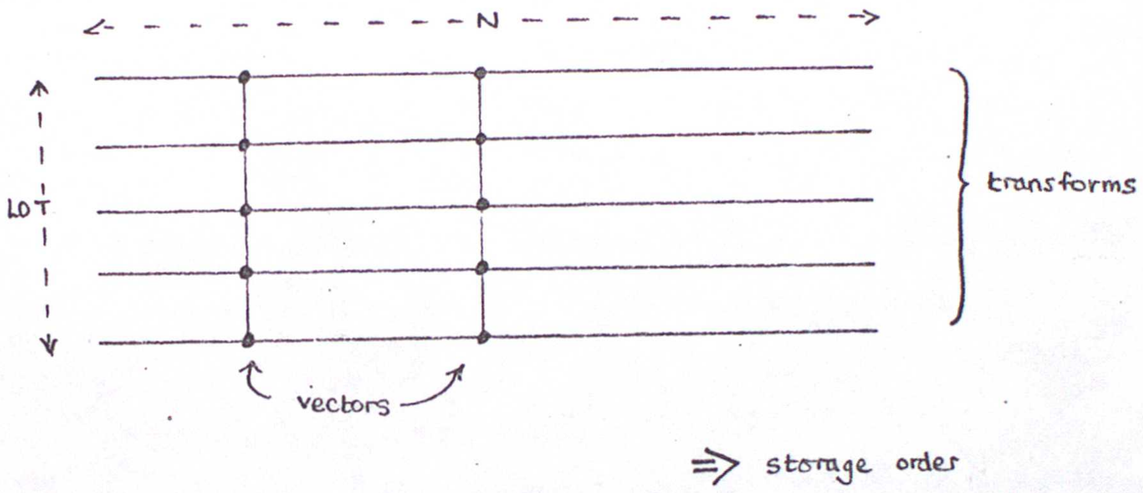

Fig 2: Multiple vectorization on Cray-1.

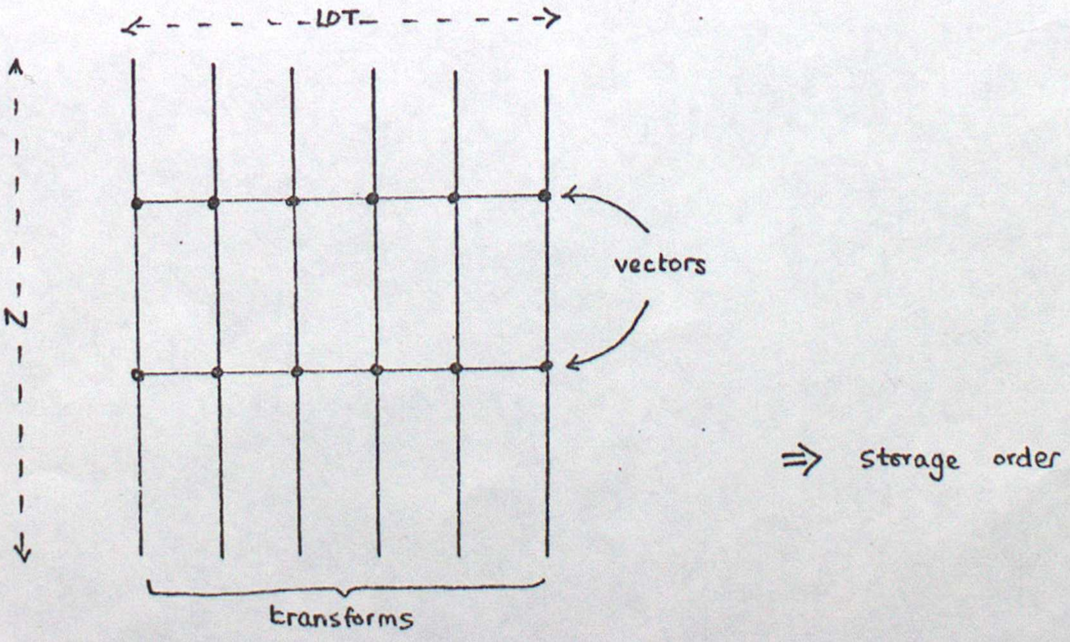

Fig 3: Multiple vectorization on Cyber 205 (or Cray-1)

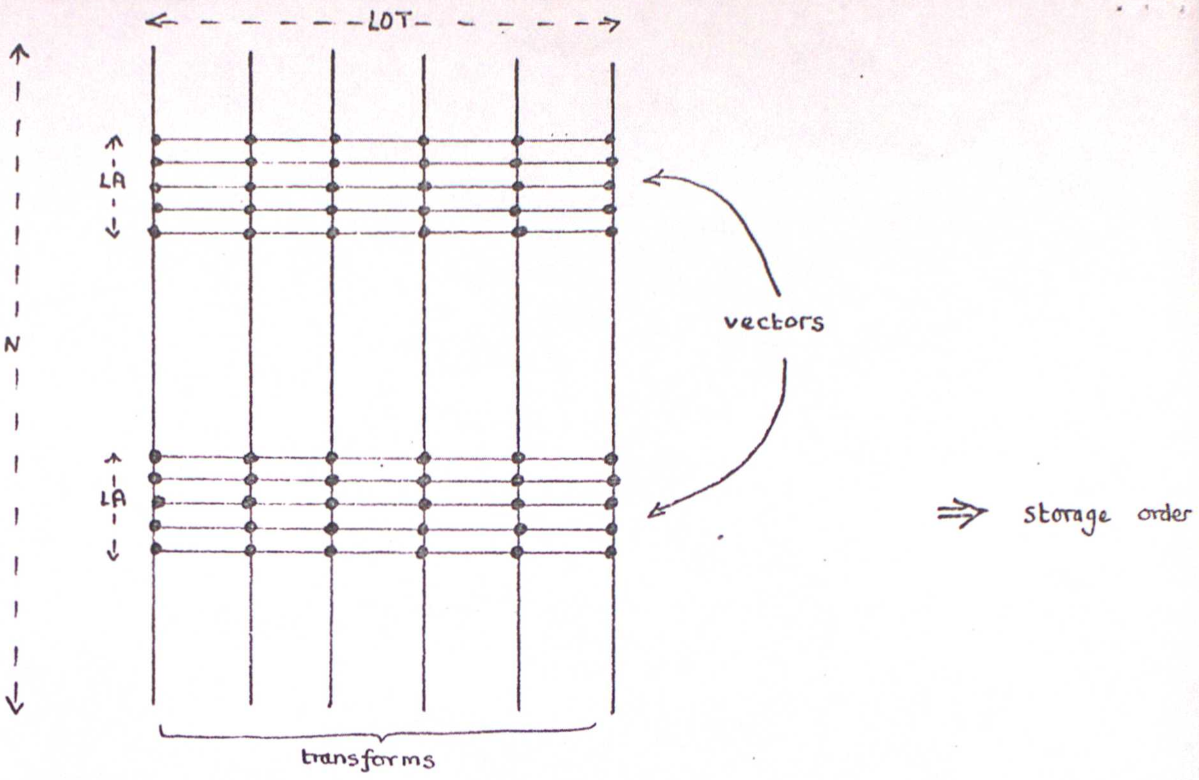

Fig 4: "Direct product" of vectorization scheme A and multiple vectorization on Cyber 205 (or Cray-1).