A Sine Transform Algorithm for the Hypercube*

Clare Chu
87-825

March 1987

Department of Computer Science
Cornell University
Ithaca, New York 14853-7501

*The work that produced this paper was [partially] completed with the assistance of computing facilities of the Advanced Computing Facility at the Cornell Center for Theory and Simulation in Science and Engineering, which is supported by the National Science Foundation and New York State.

A Sine Transform Algorithm for the Hypercube

Clare Chu

Center for Applied Mathematics

Cornell University, Ithaca, NY

Dept. of Computer Science TR 87-825

March, 1987

Abstract: A new sine transform algorithm is presented where the pre- and post-processing steps are amenable to implementation on the hypercube parallel computer. Interprocessor communication is minimized at the expense of some redundant computations resulting in an algorithm with almost linear speedup against the conventional sequential algorithm. The transforms for both naturally ordered input and bit-reversed input can be processed, thereby avoiding the communication overhead needed to either run an autosort algorithm or to *unscramble* the results by performing a bit-reversed permutation involving about $O(d)$ parallel transmissions on hypercubes of dimension d.

The work that produced this paper was [partially] completed with the assistance of computing facilities of the Advanced Computing Facility at the Cornell Center for Theory and Simulation in Science and Engineering, which is supported by the National Science Foundation and New York State.

1. Introduction

The adaptation of numerical algorithms to new parallel architectures is a challenge involving considerations beyond the usual operations counts and stability analyses. Details concerning the placement of data, the synchronization of computation and the accessing of data, including communication costs for systems without shared memory, suddenly become of paramount importance. Computations should be as independent of each other as possible to maximize parallelism and in many cases, dependent computations have to be reformulated into as large extent as independent computations.

Since the hypercube system is a group of parallel processors with local memory only, communication between dependent portions of the computations must take place by message passing. An overhead is associated with this message passing since communication rates are usually much slower than that of computation. Hence message passing should be kept to a minimum and always associated with some computational need. Busy-work, such as permutation of data sequences among the processor should be avoided, if possible, as they contribute nothing to the speedup gained by parallel processing.

The implementation of symmetric FFTs on the hypercube is nowhere as straightforward as that of the complex FFT. While most symmetric FFT algorithms utilize the complex FFT as the core computation, they are devised by a series of pre- and post- processing steps designed to take advantage of their associated symmetries to minimize their operations count. In the conventional single processor setting the savings of both storage and computational effort is a reasonable goal to pursue.

An important detail in multiprocessor FFTs is the fact that the output of the Cooley-Tukey complex FFT algorithm is generated in bit-reversed order. In the single processor case this array can be unscrambled easily with index permutations. However in the case of the hypercube, the bit-reverse unscrambling takes d parallel transmission steps, where d is the dimension of the particular hypercube (Swarztrauber, 1986b). If the data is not unscrambled, post- processing operations for the sine transform require a significant amount of communication between processors. Since the complex FFT itself is extremely efficient and also efficiently parallelizable, it would be a shame if the cost of the pre- and post- processing operations

and communications were to outweigh the benefits of parallel computation.

Our new sine transform algorithm attempts to keep pre- and post- processing communications to a minimum at the expense of a relatively small number of redundant calculations. We believe that with the high overhead associated with communication in the hypercube, the total return of this algorithm will be superior to the savings of some computational effort at the expense of a severe tie-up in communications. In fact our algorithm has a higher speedup rate than that of a similarly implemented complex FFT without any pre- or post- processing.

In section 2 we present our sine transform algorithm with less pre- and post-processing and compare it to an existing vectorizable algorithm. We also discuss a possible implementation of this conventional algorithm and show why it is not practical for implementation on the hypercube. Section 3 discusses the topology of the hypercube and the relative merits of data mapping schemes. In section 4 we show how to implement our sine transform algorithm and its inverse on the hypercube. Finally we give implementation results on the Intel iPSC in section 5, and conclude with some remarks on two-dimensional transforms with applications to Fast Poisson Solvers.

2. The Sine Transform

The sine transform of a real sequence $y(j)$ for $j = 1, 2, \ldots, N-1$ is given by

$$b(n) = \frac{2}{N} \sum_{j=1}^{N-1} y(j) \sin nj\frac{\pi}{N}, \quad n = 1, \ldots, N-1. \tag{2.1}$$

It is also its only inverse except for a multiplicative constant

$$y(j) = \sum_{n=1}^{N-1} b(n) \sin jn\frac{\pi}{N}, \quad j = 1, \ldots, N-1. \tag{2.2}$$

The sine transform uses only sines as a complete set of functions in the interval $[0, 2\pi]$ and is derived from the imaginary portion of the complex transform $C(n)$ of the extended sequence $(0, y(1), y(2), \ldots, y(N-1), 0, -y(N-1), \ldots, -y(2), -y(1))$. $C(n)$ is pure imaginary and the sine series $b(n)$ is

$$b(n) = 2iC(n), \quad n = 1, 2, \ldots, N-1. \tag{2.3}$$

The sine transform can be calculated efficiently via the complex FFT by taking advantage of the symmetry in the extended sequence. Methods for transforming this and other symmetric sequences are discussed in G. D. Bergland (1968), E. O. Brigham (1974), Cooley *et al.* (1970), Dollimore (1973), Swarztrauber (1977), Swarztrauber (1982), and Swarztrauber (1986a). The vector properties of symmetric FFT algorithms with pre- and post- processing is discussed in Swarztrauber (1982). And a commercially available package from the National Center for Atmospheric Research contains 19 programs for transforming real and complex sequences, as well as sine, cosine and quarter wave transforms [Swarztrauber (1982)].

2.1. A Hypothetical Implementation of the Sine Transform on the Hypercube.

A vectorizable method for computing the sine transform is given by Cooley *et al.* (1970). It is the reverse of Dollimore's (1973) algorithm and calculates the inverse sine transform, which, as we noted before, is also the forward sine transform scaled differently. It is one of the classic pre- and post- processed algorithms and has as its core computation, an inverse complex FFT of length $N/2$ for a sine transform of length N.

There are several layers of pre- and post- processing which are required to get to this core computation and we shall outline them briefly and see how they would fit into the hypercube. Let $N = 2^k$, k some integer and $P = 2^d$, the number of processors.

- Enter $Y(1), \ldots, Y(N-1)$ real numbers into the hypercube in a U-Turn (see section 3).
- form $N/2 + 1$ terms of the conjugate even complex sequence

$$\begin{aligned} X(0) &= (2Y(1), 0) \\ X(j) &= (Y(2j+1) - Y(2j-1), Y(2j)), \quad j = 1, \ldots, N/2 - 1 \\ X(N/2) &= (-2Y(N-1), 0) \end{aligned}$$

  (requires only nearest neighbor communication).
- Overwrite $X() \leftarrow X^*()/N$.
- $\omega_0 = \exp\{2\pi i/N\}$.

- Form the auxilliary arrays

$$A_1(j) \leftarrow X(j) + X^*(N/2 - j)$$
$$A_2(j) \leftarrow [X(j) - X^*(N/2 - j)]\omega_0^j$$
$$j = 0, \ldots, N/4$$

(requires nearest neighbor communications if original sequence was mapped in a U-Turn).

- Form the complex array that the IFFT is to be done on

$$X(0) \leftarrow A_1(0) + iA_2(0)$$
$$X(j) \leftarrow A_1(j) + iA_2(j)$$
$$X(N/2 - j) \leftarrow A_1^*(j) + iA_2^*(j)$$
$$j = 1, \ldots, N/4 - 1$$
$$X(N/4) \leftarrow A_1(N/4) + iA_2(N/4)$$

(requires only nearest neighbor communications if original data was mapped in a U-Turn).

- Call complex IFFT subroutine on X() of length N/2. store results in X().
- Set

$$Y(2j) \leftarrow \text{Re } X(j)$$
$$Y(2j+1) \leftarrow \text{Im } X(j)$$
$$j = 0, \ldots, N/2 - 1$$

(local inside processor).

- Set

$$b(j) = \frac{1}{2}\{[Y(j) - Y(-j)] + \frac{1}{2\sin(\pi j/N)}[Y(j) + Y(-j)]\}$$
$$j = 1, 2, \ldots, N - 1$$

(This is the hard part since the data is scrambled up in a bit-reversed order and thus even with our U-Turn mapping the communications needed is more than just nearest neighbor).

- $b()$ is the sine transform of the original $Y()$.

Suppose that we were able to find $b()$ by doing a bunch of reflection communications, i.e. processor j with $2^k - j - 1$ for some k, we would still have our $b()$ in

bit-reversed order in 2-vectors, i.e. for $N = 32$.

0	31
1	30
16	15
17	14
8	23
9	22
24	7
25	6
4	27
5	26
20	11
21	10
12	19
13	18
28	3
29	2

Not a very pretty state of affairs for attempting yet another sine transform for the inverse, with its associated pre- and post- processing.

A way to get around this bit-reversed result is obviously to unscramble at the end of the complex FFT subroutine. As Swarztrauber (1986b) has shown, this involves an extra d exchanges where d is the dimension of the hypercube. Therefore if we assume that we were to unscramble after the FFT is done, we would get a total of approximately $3d + 3$ parallel transmissions just for the forward transform. Since we are primarily interested in algorithms applying the sine transform to fast Poisson solvers where the data in the transform space is not required to be unscrambled before being inverse transformed, we would not want to pay the price of unscrambling just to make the algorithm workable. Thus even though this sine transform algorithm is efficient for the single processor case, computing the $(N-1)$-point sine transform through an $N/2$-point complex FFT, it requires too many communication steps to make it attractive for use on the hypercube.

Finally note that we do not consider Dollimore's (1973) algorithm because it contains at the end, a recurrence formula which is not efficiently parallelized.

2.2. Development

In this section we present our new sine transform algorithm designed to be implementable on the hypercube with a simplified pre- and post- processing scheme. We shall in fact keep all of the post-processing local to a processor and require only one linear-shift communication (or *shift-vector* operation of McBryan and Van de Velde (1985)) for the pre-processing, i.e. every processor sends one real number to its preceeding node and receives one real number from its succeeding node. In the case that the original sequence is generated by a formula instead of entered in, we can circumvent this communication by having each processor generate one extra real number. This method results in the sine transform being laid out so that just two exchanges are required for pre-processing the inverse sine transform. This simplified procedure is accomplished by doing some redundant computations and computing a complex FFT of length N.

Since the sine transform of an $(N-1)$-point sequence is derived from the imaginary part of the complex transform of the extended odd $(2N)$-point sequence of real numbers, we first consider the use of an N-point complex FFT to find the DFT of this extended sequence.

Let $y(j)$, $j = 0, 1, \ldots, 2N-1$ be the real extended odd sequence for our original $(N-1)$-point sequence. We can split this sequence into the sequences

$$\begin{aligned} X_1(j) &= y(2j) \quad \text{odd} \\ X_2(j) &= y(2j+1) \quad \text{qtr. odd} \\ & j = 0, 1, \ldots, N-1 \end{aligned} \tag{2.2.1}$$

Swarztrauber (1986a) has shown that these two sequence also have symmetries and that

$$X_1(N-j) = y(2N-2j) = -y(2j) = -X_1(j)$$

making $X_1(j)$ an odd sequence (O-symmetric) of length N. Meanwhile $X_2(j)$ exhibits the following symmetry

$$X_2(N-j-1) = y(2N-2j-1) = -y(2j+1) = -X_2(j).$$

We define such a sequence $X(j)$ where $X(j) = -X(N-j-1)$ to be quarter-wave odd (QO-symmetric).

Now we form the complex sequence

$$X(j) = X_1(j) + iX_2(j) \tag{2.2.2}$$

and find the DFT $A(n) \leftrightarrow X(j)$ (letting $\leftrightarrow$ denote a transform pair). Using linearity we obtain

$$A(n) = A_1(n) + iA_2(n) \tag{2.2.3}$$

where $A_1(n) \leftrightarrow X_1(j)$ and $A_2(n) \leftrightarrow X_2(j)$.

Since $X_1(j)$ is real odd and $X_2(j)$ is real quarter odd, their transforms have the following properties (see Swarztrauber, 1986a):

$$\begin{aligned}
A_1(n) &= -A_1^*(n) \\
A_1(n) &= A_1^*(-n) \\
A_2(n) &= -\exp\{in2\pi/N\}A_2^*(n) \\
A_2(n) &= A_2^*(-n) \\
A_2(n) &= \exp\{in\pi/N\}\tilde{A}_2(n) \\
\tilde{A}_2(n) &= -\tilde{A}_2^*(n)
\end{aligned}$$

where $\tilde{A}_2(n)$ is pure imaginary. Thus equation (2.2.3) becomes

$$A(n) = A_1(n) + i\exp\{in\pi/N\}\tilde{A}_2(n) \tag{2.2.4}$$

and taking the conjugate of $A(n)$ gives

$$\begin{aligned}
A^*(n) &= A_1^*(n) - i\exp\{-in\pi/N\}\tilde{A}_2^*(n) \\
&= -A_1(n) + i\exp\{-in\pi/N\}\tilde{A}_2(n) \quad n = 0,\ldots,N/2-1
\end{aligned} \tag{2.2.5}$$

Solving (2.2.4) and (2.2.5) for $A_1(n)$ and $\tilde{A}_2(n)$ gives

$$\begin{aligned}
A_1(n) &= i\mathrm{Im}[A(n)] - i\tan\{n\pi/N\}\mathrm{Re}[A(n)] \\
\tilde{A}_2(n) &= \mathrm{Re}[A(n)]/\left(i\cos\{n\pi/N\}\right) \\
&\qquad n = 0,\ldots,N/2-1
\end{aligned} \tag{2.2.6}$$

Finally

$$\begin{aligned}
A(N/2) &= A_1(N/2) + i\exp\{i\pi/2\}\tilde{A}_2(N/2) \\
&= -\tilde{A}_2(N/2)
\end{aligned}$$

And

$$A_1(N/2) = 0$$
$$\tilde{A}_2(N/2) = -A(N/2) = -i\mathrm{Im}[A(N/2)]$$

The splitting equation for odd sequences gives

$$C(n) = \frac{1}{2}[A_1(n) + \tilde{A}_2(n)] \quad n = 1, \ldots, N/2$$
$$C^*(N-n) = \frac{1}{2}[A_1(n) - \tilde{A}_2(n)] \quad n = 1, \ldots, N/2 - 1 \tag{2.2.7}$$

Since $y(j)$ is odd, $C(n) = -C^*(n)$ and thus

$$C(N-n) = \frac{1}{2}[\tilde{A}_2(n) - A_1(n)] \quad n = 1, \ldots, N/2 - 1$$

Since we really want $b(n) = 2iC(n)$ we multiply the $A_1(n)$ and $\tilde{A}_2(n)$ by i, so that we can use real arrays for $A_1(n)$ and $\tilde{A}_2(n)$ in our program.

$$A_1(n) = -\mathrm{Im}[A(n)] + \tan\{n\pi/N\}\mathrm{Re}[A(n)]$$
$$\tilde{A}_2(n) = \mathrm{Re}[A(n)]/\cos\{n\pi/N\}$$
$$\tilde{A}_2(N/2) = \mathrm{Im}[A(N/2)] \tag{2.2.8}$$

Therefore we have the following equations for $b(n)$:

$$b(n) = A_1(n) + \tilde{A}_2(n) \quad n = 1, \ldots, N/2$$
$$b(N-n) = \tilde{A}_2(n) - A_1(n) \quad n = 1, \ldots, N/2 - 1 \tag{2.2.9}$$

and we never had to calculate $C(n)$ at all. In fact none of our arrays have to be longer than N. The $y(j)$'s were extended only for illustration purposes. $A_1(n)$ and $\tilde{A}_2(n)$ are only of length $N/2 + 1$ and are real.

Summarizing:

(1) Set

$$X_1(j) = Y(2j)$$
$$X_2(j) = Y(2j+1) \quad j = 0, 1, \ldots, N-1$$

And let $X(j) = X_1(j) + iX_2(j)$.

(2) Find the N-point DFT $A(n) \leftrightarrow X(j)$.

(3) Set

$$A_1(n) = -\mathrm{Im}[A(n)] + \tan\{n\pi/N\}\mathrm{Re}[A(n)]$$
$$A_2(n) = \mathrm{Re}[A(n)]/\cos\{n\pi/N\} \quad n = 0, \ldots, N/2-1$$
$$A_1(N/2) = 0$$
$$A_2(N/2) = \mathrm{Im}[A(N/2)]$$

(4) Find

$$b(n) = A_1(n) + A_2(n) \quad n = 1, \ldots, N/2$$
$$b(N-n) = A_2(n) - A_1(n) \quad n = 1, \ldots, N/2-1$$

This last step, requires $b(N-n)$ to be calculated by values in the nth position in $A_1()$ and $A_2()$. Since this would require additional communication in a distributed environment, we require that the $b(n)$'s all be computed from values in the nth position of the auxilliary arrays A_1 and A_2. Therefore we may want to do some redundant computations instead of taking full advantage of the symmetries. Since $X_1(j)$ and $X_2(j)$ are real, their transforms satisfy

$$A_2(-n) = A_1^*(n)$$
$$A_2(-n) = A_2^*(n)$$

and the pure imaginary sequence $\tilde{A}_2(N-n) = \tilde{A}_2^*(n)$. These arrays have all been multiplied by i so we get, for the real version,

$$A_1(-n) = -A_1(n)$$
$$\tilde{A}_2(-n) = -\tilde{A}_2(n)$$

Instead of exploiting this symmetry, we proceed to figure out the rest of the $A_1(-n)$ and $A_2(-n)$ directly in step (3). Step (3) is now changed to be,

(3)

$$A_1(n) = -\mathrm{Im}[A(n)] + \tan\{n\pi/N\}\mathrm{Re}[A(n)]$$
$$n = 0, \ldots, N/2-1, N/2+1, \ldots, N-1$$
$$A_2(n) = \mathrm{Re}[A(n)]/\cos\{n\pi/N\}$$
$$n = 0, \ldots, N/2-1$$
$$A_2(n) = -\mathrm{Re}[A(n)]/\cos\{n\pi/N\}$$
$$n = N/2+1, \ldots, N-1$$
$$A_1(N/2) = 0$$
$$A_2(N/2) = \mathrm{Im}[A(N/2)]$$

And step (4) now becomes

$$b(n) = A_1(n) + A_2(n) \quad n = 1, \ldots, N/2$$
$$b(n) = A_1(n) - A_2(n) \quad n = N/2 + 1, \ldots, N - 1$$

Notice that the $\tilde{A}_2(n)$ is different for $n < N/2$ and $n > N/2$. This is because what we really want is

$$\tilde{A}_2(-n) = \mathrm{Re}[A(-n)]/\cos\{-n\pi/N\}$$

but we are computing

$$\tilde{A}_2(N-n) = \mathrm{Re}[A(N-n)]/\cos\{(N-n)\pi/N\}$$

Since

$$\cos\{(N-n)\pi/N\} = -\cos\{-n\pi/N\}$$

$$\tilde{A}_2(N-n) = -\tilde{A}_2(-n)$$

and thus

$$\tilde{A}_2(n) = -\mathrm{Re}[A(n)]/\cos\{n\pi/N\} \quad \text{for } n = N/2 + 1, \ldots, N - 1$$

Now everything after the complex FFT in step (2) is entirely local to a processor with the introduction of N extra multiplications and $N/2$ extra additions. However these extra operations are not noticable in the parallel setting since the processors are working independently. If the redundant computations were not introduced, some of the processors would be sitting idle anyhow. Therefore we have achieved load balancing as well as saved $O(d)$ communication steps for a total of $2d - 1$ exchanges and one linear-shift. Notice the $2d - 1$ exchanges are solely used for the computation of the complex FFT on a hypercube mapped on a BRGC, therefore the pre-processing resulted in only one (optional) communication.

2.3. The Inverse Sine Transform of a Sequence in Bit-Reversed Order

Even though the sine transform is its own inverse (scaled by a constant) our output from the forward sine transform is in bit-reversed order. Therefore we require an inverse sine transform which takes data originally in bit-reversed order and finishes with it in the natural order.

An inverse complex FFT of a transformed sequence in bit-reversed order gives back the original sequence. Recall that the sine transform is actually the first half of the imaginary part of a $2N$-point complex transform of a $2N$-point real odd sequence. To get this real odd sequence back, we should devise the $2N$-point inverse FFT of this complex transform (which is pure imaginary) (and in bit-reversed order). We should try to do this in a similar way as we did the forward transform, by using arrays only of length N and not actually computing the whole inverse transform of length $2N$.

Let $b(n)$, $n = 1, \ldots, N-1$ be the sine series of $y(j)$, $j = 1, \ldots, N-1$. Let E_{n-1} be an exchange matrix of order $n-1$ [see section 4.2 for definitions].

The Inverse FFT of

$$\begin{bmatrix} (0,0) \\ (0, -\frac{1}{2}b(n)) \\ (0,0) \\ -E_{n-1}(0, -\frac{1}{2}b(n)) \end{bmatrix}$$

is

$$\begin{bmatrix} (0,0) \\ (y(j),0) \\ (0,0) \\ -E_{n-1}(y(j),0) \end{bmatrix}$$

and the Inverse FFT of

$$\begin{bmatrix} (0,0) \\ (\frac{1}{2}b(n),0) \\ (0,0) \\ -E_{n-1}(\frac{1}{2}b(n),0) \end{bmatrix}$$

is

$$\begin{bmatrix} (0,0) \\ (0,y(j)) \\ (0,0) \\ -E_{n-1}(0,y(j)) \end{bmatrix}$$

Suppose we have $b(n)$ in bit reversed order, then we need to use an IFFT algorithm where data is input in bit-reversed order. We shall also use real numbers

instead of pure imaginary numbers whenever possible. First we overwrite

$$b(n) \leftarrow -\frac{1}{2}b(n)$$

Define $\bar{b}'(n)$ as the extended sequence of $b(n)$ of length $2N$ and in bit reverse scrambled order. Let $A_1(n)$ be a real array consisting of the first half of $\bar{b}'(n)$ (i.e. $n = 0, \ldots, N-1$); and $A_2(n)$ be a real array consisting of the second half of $\bar{b}'(n)$ $n = N, \ldots, 2N-1$.
Example $N = 8$:

$$A_1(n) = \begin{bmatrix} 0 \\ 0 \\ b_4 \\ -b_4 \\ b_2 \\ -b_6 \\ b_6 \\ -b_2 \end{bmatrix}$$

and

$$A_2(n) = \begin{bmatrix} b_1 \\ -b_7 \\ b_5 \\ -b_3 \\ b_3 \\ -b_5 \\ b_7 \\ -b_1 \end{bmatrix}$$

Now form

$$A(n) = A_1(n) + iA_2(n) \tag{2.3.1}$$

Find the IFFT (bit-reverse input version) $X(k) \leftrightarrow A(n)$. By linearity we have

$$X(k) = X_1(k) + iX_2(k) \tag{2.3.2}$$

Proceeding as we did for the forward transform, we observe the following properties. $A_1(n)$ is real and in its bit-reversed order is odd, so its inverse transform satisfies

$$X_1(k) = X_1^*(-k)$$
$$X_1(k) = -X_1^*(k)$$

$A_2(n)$ is real and in its bit-reversed order is quarter-wave odd, so its inverse transform satisfies

$$\begin{aligned} X_2(k) &= -\exp\{-ik2\pi/N\}X_2^*(k) \\ X_2(k) &= X_2^*(-k) \\ X_2(k) &= \exp\{-ik\pi/N\}\tilde{X}_2(k) \\ \tilde{X}_2(k) &= -\tilde{X}_2^*(k) \end{aligned}$$

We thus have

$$\begin{aligned} X(k) &= X_1(k) + iX_2(k) \\ &= X_1(k) + i\exp\{-ik\pi/N\}\tilde{X}_2(k) \end{aligned} \tag{2.3.3}$$

and taking conjugates,

$$\begin{aligned} X^*(k) &= X_1^*(k) - i\exp\{ik\pi/N\}\tilde{X}_2^*(k) \\ &= -X_1(k) + i\exp\{ik\pi/N\}\tilde{X}_2(k) \end{aligned} \tag{2.3.4}$$

So

$$\begin{aligned} \tilde{X}_2(k) &= \mathrm{Re}[X(k)]/(i\cos\{k\pi/N\}) \\ & \quad k < N/2 \\ \tilde{X}_2(k) &= -\mathrm{Re}[X(k)]/(i\cos\{k\pi/N\}) \\ & \quad k > N/2 \end{aligned} \tag{2.3.5}$$

Solving for $X_1(k)$ gives

$$X_1(k) = i\mathrm{Im}[X(k)] + i\tan\{k\pi/N\}\mathrm{Re}[X(k)] \tag{2.3.6}$$

Since we want to work with real quantities, we multiply each of these expressions by i and get

$$\begin{aligned} X_1(k) &= -\mathrm{Im}[X(k)] - \tan\{k\pi/N\}\mathrm{Re}[X(k)] \\ & \quad k < N/2 \quad k > N/2 \\ X_1(N/2) &= 0 \\ \tilde{X}_2(k) &= \mathrm{Re}[X(k)]/(\cos\{k\pi/N\}) \\ & \quad k < N/2 \\ \tilde{X}_2(N/2) &= -\mathrm{Im}[X(N/2)] \\ \tilde{X}_2(k) &= -\mathrm{Re}[X(k)]/(\cos\{k\pi/N\}) \\ & \quad k > N/2 \end{aligned}$$

Notice that we have introduced redundant computations since strictly speaking $X_1(k)$ and $X_2(k)$ only need to be found for $k < N/2$. But we want to eliminate the communication for the next step where the answers are obtained directly by

$$\begin{aligned} Y(k) &= X_1(k) + \tilde{X}_2(k) \quad k = 1, \ldots, N/2 \\ Y(k) &= X_1(k) - \tilde{X}_2(k) \quad k = N/2 + 1, \ldots, N-1 \end{aligned} \tag{2.3.7}$$

recalling that $X_2(k) = \exp\{-ik\pi/N\}\tilde{X}_2(k)$.

3. Complex FFT Data Mapping Schemes

The heart of any symmetric FFT algorithm is of course the radix-two complex FFT. Throughout this paper we have assumed the existence of a subroutine which takes in a sequence of complex numbers and outputs either its discrete Fourier transform or its inverse Fourier transform. Now we shall examine how this is done on the hypercube and how we will map the complex sequence into the hypercube to best utilize its architecture subject to the constraints of the way symmetric sequences pre-process and post-process. In Chu (1987), various complex FFT implementations were explored with an eye towards load balancing, communication minimization, and storage considerations. Two methods were proposed which yielded better timings than the conventional set-up of the complex FFT in one single column [see Chamberlain (1986)]. One was called the two-track method and the second one the U-Turn method. For example, if we had a complex vector of length 16 and a two-dimensional hypercube, we would lay out our data as follows:

	Two-Track		U-Turn	
p(00)	0	8	0	15
	1	9	1	14
p(01)	2	10	2	13
	3	11	3	12
p(11)	4	12	4	11
	5	13	5	10
p(10)	6	14	6	9
	7	15	7	8

As was shown in Chu (1987), these two layouts provided complex FFT implementations which were both load balanced and efficient, with almost linear speedup.

A second issue which must be addressed is that the natural mapping of data into the hypercube, that is, with subvector x_j mapped in processor j, is not well suited for general numerical computations, although it is optimal for computing the FFT butterfly. Chamberlain (1986) and Chan (1986) have both pointed out the desirability of the use of a Binary Reflected Gray Code mapping of the nodes of the hypercube so that nearest neighbor properties would be preserved. Another property that the BRGC possesses is that processors that are reflections of each other are also at a Hamming distance of one. This is especially important for sequences where the pre-processing leads one to map the complex FFT computations in a U-Turn. Therefore in what follows, we assume that all the computations are done on a hypercubes whose nodes are mapped in the order of the BRGC. [See Chamberlain (1986), Johnsson (1985), and Saad and Schultz (1985) on the BRGC and the hypercube].

4. Implementation on the Hypercube

Our sine transform algorithm has the property that if the real sequence $y()$ were input into the hypercube in the natural order, its sine transform would appear in bit-reversed order in a U-Turn. However in most application, it does not matter that the transformed sequence is scrambled since it is simply scaled and then inverse transformed back to the original domain. Thus the important property is that we are also able to inverse transform this bit-reversed U-Turned sequence back to the original natural order. To see how this all works, let us walk our way through an example.

4.1. Example.

At the start we have for $N = 16$, the following set-up for the sine transform of y_i, $i = 1, \ldots, N-1$.

000	0	y_1
001	y_2	y_3
011	y_4	y_5
010	y_6	y_7
110	y_8	y_9
111	y_{10}	y_{11}
101	y_{12}	y_{13}
100	y_{14}	y_{15}

These are pre-processed according to the formula in step (1) for the forward sine transform to get.

000	X_0	X_{15}
001	X_1	X_{14}
011	X_2	X_{13}
010	X_3	X_{12}
110	X_4	X_{11}
111	X_5	X_{10}
101	X_6	X_9
100	X_7	X_8

Now the forward FFT (Step (2)) is done on X_i and we get the bit-reversed result A_i.

000	A_0	A_{15}
001	A_8	A_7
011	A_4	A_{11}
010	A_{12}	A_3
110	A_2	A_{13}
111	A_{10}	A_5
101	A_6	A_9
100	A_{14}	A_1

After local post-processing steps (3) and (4), we obtain:

000	b_0	b_{15}
001	b_8	b_7
011	b_4	b_{11}
010	b_{12}	b_3
110	b_2	b_{13}
111	b_{10}	b_5
101	b_6	b_9
100	b_{14}	b_1

To get the Inverse Sine Transform of these b_i's, we replace $b_i \leftarrow -\frac{1}{2}b_i$. Now we get the array A_i by the following rules.

A_0	$0 + ib_1$
A_8	$0 - ib_{15}$
A_4	$b_8 + ib_9$
A_{12}	$-b_8 - ib_7$
A_2	$b_4 + ib_5$
A_{10}	$-b_{12} - ib_{11}$
A_6	$b_{12} + ib_{13}$
A_{14}	$-b_4 - ib_3$
A_1	$b_2 + ib_3$
A_9	$-b_{14} - ib_{13}$
A_5	$b_{10} + ib_{11}$
A_{13}	$-b_6 - ib_5$
A_3	$b_6 + ib_7$
A_{11}	$-b_{10} - ib_9$
A_7	$b_{14} + ib_{15}$
A_{15}	$-b_2 - ib_1$

The A_i are positioned in the hypercube as follows:

000	A_0	A_8
001	A_4	A_{12}
011	A_2	A_{10}
010	A_6	A_{14}
110	A_1	A_9
111	A_5	A_{13}
101	A_3	A_{11}
100	A_7	A_{15}

The Inverse FFT (with bit-reversed input) is now performed on A_i and we get X_i.

000	X_0	X_1
001	X_2	X_3
011	X_4	X_5
010	X_6	X_7
110	X_8	X_9
111	X_{10}	X_{11}
101	X_{12}	X_{13}
100	X_{14}	X_{15}

Local post-processing gives us back the original array y_i in the original order.

000	0	y_1
001	y_2	y_3
011	y_4	y_5
010	y_6	y_7
110	y_8	y_9
111	y_{10}	y_{11}
101	y_{12}	y_{13}
100	y_{14}	y_{15}

The setup for the forward transform is to order the hypercube as a BRGC and put the data into the processors in the natural order by partitioning a vector **y** of length N into subvectors y_i of length N/P and placing y_i in processor $G_k(i)$, where G_k denotes the Binary Reflected Gray Code of dimension k.

Step (1) requires each processor $G_k(i)$ $i > 0$ to send $y_i(1)$, its first element to $G_k(i-1)$. This is the linear-shift communication which is not necessary if the processors were to generate the $y()$'s directly. Step (2) finds the usual complex FFT computation done on data mapped in a U-Turn. The final steps (3) and (4) are entirely processor local and the sine series $b()$ ends up in bit-reversed order mapped in a U-Turn. The whole procedure is summarized: Let $Y(j)$ be real, $j = 1, \ldots, N-1$.

(1) Set

$$\begin{aligned} X(0) &= iY(1) \\ X(j) &= Y(2j) + iY(2j+1) \quad j = 1, \ldots, N/2-1 \\ X(N-j) &= -Y(2j) - iY(2j-1) \quad j = 1, \ldots, N/2-1 \\ X(N/2) &= -iY(N-1) \end{aligned}$$

This step requires only one linear-shift communication.

(2) Find the N-point DFT $A(n) \leftrightarrow X(j)$ by using the complex FFT algorithm where input data is mapped in a U-Turn [see Chu (1987)]. The answers in $A()$ appear in bit-reversed order.

(3) Set

$$A_1(n) = -\mathrm{Im}[A(n)] + \tan\{n'\pi/N\}\mathrm{Re}[A(n)]$$
$$n = 0, 2, \ldots, N-1$$
$$A_2(n) = (-1)^n \mathrm{Re}[A(n)]/\cos\{n'\pi/N\}$$
$$n = 0, 2, \ldots, N-1$$
$$A_1(1) = 0$$
$$A_2(1) = \mathrm{Im}[A(N/2)]$$

where n' = bit-reversed(n).

(4) Find

$$b(n) = A_1(n) + (-1)^n A_2(n)$$
$$n = 0, 2, \ldots, N-1$$
$$b(1) = A_1(n) + A_2(n)$$

Steps (3) and (4) are local to a processor.

The total amount of communication for the forward transform is $2d + 1$ exchanges for the complex FFT and 1 linear-shift communication for the pre-processing.

4.2. Set Up for the Inverse Sine Transform

Recall that the data from the forward transform was left in bit-reversed order and mapped into the hypercube in a U-Turn. Since we would like to eventually get the original sequence $y()$ back in the natural order, we would require that the inverse transform proceed on a two-track data layout. Some fancy foot-work is required to do the pre-processing required for the inverse transform.

First we give two lemmas concerning the binary bit-reversed sequence to enhance our facility in understanding and manipulating such a permutation.

Let B_k be the binary code for integers $0, 1, \ldots, 2^k - 1$, and R_k be its bit-reversed code. (^denotes concatenation) (and the superscript R denotes bit reversal)

Lemma 1.

$$R_k = \begin{bmatrix} 2 \cdot R_{k-1} \\ 2 \cdot R_{k-1} + 1 \end{bmatrix} = \begin{bmatrix} R_{k-1}\hat{}0 \\ R_{k-1}\hat{}1 \end{bmatrix}$$

Proof.

$$B_k = \begin{bmatrix} 0\hat{}B_{k-1} \\ 1\hat{}B_{k-1} \end{bmatrix}$$

Reversing B_k gives

$$B_k^R = R_k = \begin{bmatrix} B_{k-1}^R\hat{}0 \\ B_{k-1}^R\hat{}1 \end{bmatrix} = \begin{bmatrix} R_{k-1}\hat{}0 \\ R_{k-1}\hat{}1 \end{bmatrix}$$

Lemma 2.

$$[R_k(2j)] = R_{k-1}(j) \quad j = 0, \ldots, 2^{k-1} - 1$$
$$[R_k(2j+1)] = R_{k-1}(j) + 2^{k-1} = 1\hat{}R_{k-1}(j)$$

Proof.
(By Induction) trivial in case $k = 0$ and $k = 1$. Suppose true for $l < k$

$$R_k(2j) = \begin{bmatrix} R_{k-1}(2j)\hat{}0 \\ R_{k-1}(2j)\hat{}1 \end{bmatrix} = \begin{bmatrix} R_{k-2}(j)\hat{}0 \\ R_{k-2}(j)\hat{}1 \end{bmatrix} = R_{k-1}(j)$$

and

$$R_k(2j+1) = \begin{bmatrix} R_{k-1}(2j+1)\hat{}0 \\ R_{k-1}(2j+1)\hat{}1 \end{bmatrix} = \begin{bmatrix} 1\hat{}R_{k-2}\hat{}0 \\ 1\hat{}R_{k-2}\hat{}1 \end{bmatrix} = 1\hat{}R_{k-1}(j)$$

Next we define a few matrices.

Let E_n be the exchange matrix of order n, i.e.

$$E_n = [e_{n-1}, e_{n-2}, \ldots, e_1, e_0]$$

Example:

$$E_4 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Let T_n be a reflection matrix defined as

$$T_n = \begin{bmatrix} 1 & 0 \\ 0 & E_{n-1} \end{bmatrix}$$

Example:

$$T_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

Let P_n be the bit-reversed permutation matrix, i.e. $P_n B_k = R_k$ where $(n = 2^k)$.
Example:

$$P_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Let S_n be the "sort" matrix for the even-odd sorting of n numbers (n even), i.e.

$$S_n = [e_1, e_3, \ldots, e_{n-1}, e_2, e_4, \ldots, e_n]$$

Example:

$$S_8^T \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} = \begin{bmatrix} x_0 \\ x_2 \\ x_4 \\ x_6 \\ x_1 \\ x_3 \\ x_5 \\ x_7 \end{bmatrix}$$

Its inverse is often referred to as the "perfect shuffle".

And finally define F_n to be a direct sum of smaller exchange matrices,

$$F_{2^k} = \begin{bmatrix} E_1 & & & & & \\ & E_1 & & & & \\ & & E_2 & & & \\ & & & E_4 & & \\ & & & & \ddots & \\ & & & & & E_{2^{k-1}} \end{bmatrix}$$

or

$$F_n = \begin{bmatrix} F_{n/2} & 0 \\ 0 & E_{n/2} \end{bmatrix}$$

Example:

$$F_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Next a few more lemmas concerning these matrices and the binary code B_k and the bit-reversed code R_k (in what follows $\mathbf{2^k}$ denotes the vector, all of whose entries are 2^k; and $\mathbf{1}$ denotes the vector of ones):

Lemma 3.

$E_n B_k = (\mathbf{2^k} - B_k - \mathbf{1}) \bmod \mathbf{2^k}$.

Proof.

$B_k = (0, 1, \ldots, 2^k - 1)$ and $E_n B_k = (2^k - 1, \ldots, 1, 0)$.

Lemma 4.

$E_n R_k = (\mathbf{2^k} - R_k - \mathbf{1}) \bmod \mathbf{2^k}$.

Proof.

Since $P_n B_k = R_k$, we have $E_n R_k = E_n P_n B_k = P_n E_n B_k = P_n(\mathbf{2^k} - B_k - \mathbf{1}) \bmod \mathbf{2^k} = (\mathbf{2^k} - R_k - \mathbf{1}) \bmod \mathbf{2^k}$. ($P_n E_n = E_n P_n$ since E_n essentially flips all the bits of all the elements of a sequence, i.e. 1's become 0's and 0's become 1's. Thus E_n commutes with any permutation matrix.)

Lemma 5.

$F_n R_k = (\mathbf{2^k} - R_k) \bmod \mathbf{2^k}$.

Proof.

Observe that $F_n = F_{n/2} \oplus E_{n/2}$ and $F_2 = I_2$ so it is trivially true for $k = 1$. By induction, suppose we have $F_{n/2} R_{k-1} = (\mathbf{2^{k-1}} - R_{k-1}) \bmod \mathbf{2^{k-1}}$, then

$$\begin{aligned} F_n R_k &= \begin{bmatrix} F_{n/2} 2 \cdot R_{k-1} \\ E_{n/2}(2 \cdot R_{k-1} + \mathbf{1}) \end{bmatrix} \\ &= \begin{bmatrix} (\mathbf{2^k} - 2 \cdot R_{k-1}) \bmod \mathbf{2^k} \\ (\mathbf{2^k} - (2 \cdot R_{k-1} + \mathbf{1})) \bmod \mathbf{2^k} \end{bmatrix} \\ &= (\mathbf{2^k} - R_k) \bmod \mathbf{2^k} \end{aligned}$$

using Lemma 1 and Lemma 4.

Lemma 6.

$T_n B_k = \mathbf{2^k} - B_k$

Proof.

Since $B_k = (0, 1, \ldots, 2^k - 1)$ and $T_n B_k = (0, 2^k - 1, \ldots, 1)$ from which the result follows.

Lemma 7.

$P_n(T_n B_k) = F_n R_k$

Proof.

Using Lemma 5 and Lemma 6, we see that $P_n(\mathbf{2^k} - B_k) = \mathbf{2^k} - R_k = F_n R_k$.

Corollary

$$P_n T_n = F_n P_n$$

Lemma 8.

$$P_n = S_n^T \begin{bmatrix} P_{n/2} & 0 \\ 0 & P_{n/2} \end{bmatrix}$$

Proof.

Lemma 2.

Lemma 9.

Let $b()$ be a sine series of real numbers $(0, b_1, b_2, \ldots, b_{n-1})$. Define $\bar{b}()$ to be its odd extension $(0, b_1, b_2, \ldots, b_{n-1}, 0, -b_{n-1}, \ldots, -b_2, -b_1)$. Let $\bar{b}'() = P_{2n}\bar{b}()$. Then the even points of $\bar{b}'() = P_n b()$ and the odd points of $\bar{b}'() = -P_n(T_n b())$.

Proof.

Applying Lemma 8 to

$$\bar{b}() = \begin{bmatrix} b() \\ -T_n b() \end{bmatrix}.$$

Now we shall use the fact that the development of the inverse transform was based on the odd extension of the sine transform in bit-reversed order and recall that the pre-processing consists of adding the first half of $\bar{b}'()$ to i times the second half of $\bar{b}'()$, i.e.

$$b(2j) + ib(2j + 1)$$

and

$$-b(2j) - ib(2j - 1)$$

Since the $b()$ were laid out in bit-reversed order in a U-Turn, (example $N = 8$):

$$\begin{array}{ll} b_0 & b_7 \\ b_4 & b_3 \\ b_2 & b_5 \\ b_6 & b_1 \end{array}$$

Lemma 9 tells us that these are the even points of $\bar{b}'()$ and gives us $b(2j)+ib(2j+1)$ as the even points in our complex sequence. This is a combination of the first column $B^{(1)}$ consisting of the $b(2j)$ with $E_{N/2}B^{(2)}$, the second column consistingof $b(2j+1)$ turned upside down.

The odd points of $\bar{b}'()$ are really $-P_n(T_n b())$ and Lemma 7 tells us that they are also represented in columns $B^{(1)}$ and $B^{(2)}$. In fact since $-P_n T_n b() = F_n P_n b()$ this gives us $-B^{(1)} - F_{N/2}B^{(2)}$ for the combination of the odd points of $\bar{b}'()$ which become the odd points of our complex sequence. Define $A^{(1)}$ to be the first column of the complex numbers to be inverse transformed in the above tables and $A^{(2)}$ to be the second column. Define $B^{(j)}$ to be the real numbers of the sine transform in column j of the table. Now it is easy to describe the pre-processing for the inverse transform.

$$A^{(1)} \leftarrow B^{(1)} + iE_{N/2}B^{(2)}$$
$$A^{(2)} \leftarrow -B^{(1)} - iF_{N/2}B^{(2)}$$

where N is the length of the transform.

This pre-processing requires 2 exchanges, all among processors which are at a Hamming distance of one if the BRGC mapping is used. Furthermore, this 2 remains constant and does not increase with the dimension of the cube.

The rest of the inverse sine transform proceeds similarly to the forward transform and is summarized below:

- Pre-process.

  $$A^{(1)} \leftarrow B^{(1)} + iE_{N/2}B^{(2)}$$
  $$A^{(2)} \leftarrow -B^{(1)} - iF_{N/2}B^{(2)}$$

  and set

  $$A(2j) \leftarrow A^{(1)}(j)$$
  $$A(2j+1) \leftarrow A^{(2)}(j)$$

  (requires two exchanges).

- Find the Inverse FFT $X(k) \leftrightarrow A(n)$, by using the complex inverse FFT algorithm on Two-Tracks [see Chu (1987)]. Since $A()$ was originally in bit-reversed orer, $X()$ appears in natural order.

- Find the real auxilliary arrays $X_1()$ and $\tilde{X}_2()$.

$$
\begin{aligned}
X_1(k) &= -\mathrm{Im}[X(k)] - \tan\{k\pi/N\}\mathrm{Re}[X(k)] \\
&\qquad k < N/2 \quad k > N/2 \\
X_1(N/2) &= 0 \\
\tilde{X}_2(k) &= \mathrm{Re}[X(k)]/(\cos\{k\pi/N\}) \\
&\qquad k < N/2 \\
\tilde{X}_2(N/2) &= -\mathrm{Im}[X(N/2)] \\
\tilde{X}_2(k) &= -\mathrm{Re}[X(k)]/(\cos\{k\pi/N\}) \\
&\qquad k > N/2
\end{aligned}
$$

- Obtain the answer $Y()$.

$$
\begin{aligned}
Y(k) &= X_1(k) + \tilde{X}_2(k) \quad k = 1, \ldots, N/2 \\
Y(k) &= X_1(k) - \tilde{X}_2(k) \quad k = N/2+1, \ldots, N-1
\end{aligned} \tag{2.3.7}
$$

As before, the last two steps are completely processor local. Thus the inverse transform requires a total of $2d+3$ exchanges with $2d+1$ of them used for the complex inverse FFT.

5. Implementation Results on the Intel iPSC

The complete procedure was implemented on a four-dimensional Intel iPSC hypercube with extended memory in the nodes. Since our goal is to compute a forward sine transform, scale it and then do an inverse sine transform, our timings are for both the forward and inverse transform done back to back. We have also implemented the Cooley *et al.* algorithm on a single node and obtained a time of 121685 milliseconds for a sine and inverse transform of length 16384.

Time (milliseconds) for Sine and Inverse Transform

$N = 16384$

# proc	time	speedup	speedup*
1	127515	1.00	0.95
2	67515–67535	1.89	1.80
4	36870–36915	3.45	3.30
8	19315–19375	6.58	6.28
16	10430–10535	12.10	11.55

(* speedup against conventional algorithm)

The results show almost linear speedup and reflect the efficiency with which the complex FFT can be parallelized. The speedup* against the sequential algorithm is also very good. Recall that the Cooley *et al.* sequential algorithm does a complex FFT of length $N/2$ or (8192 for our example). Therefore it should be roughly twice as fast as our algorithm in the sequential case. But our algorithm using only one processor is only about 4% slower than the conventional algorithm. So we see that the layers of pre- and post- processing are quite wasteful even in the totally sequential case.

One may wonder how this method would compare to doing a $2N$-point complex transform of the extended odd sequence (the only other feasible method on the hypercube). Results for the $N = 32768$ sequence are given below.

Time (milliseconds) for Complex Forward and Inverse Transform

$N = 32768$

# proc	time	speedup
1	197840	1.00
2	97785–97825	2.02
4	52450–52605	3.76
8	28800–28865	6.85
16	15055–15585	12.69

Comparing results shows that our method is faster than the direct computation of the $2N$-point sequence by 45%, 43%, 49% and 48% for hypercubes of dimension 1, 2, 3, and 4 respectively. Sequentially, it is also 55% faster.

6. Conclusion and Discussion.

The sine transform is used extensively in the Fourier method for the solution of boundary value problems in partial differential equations with Dirichlet-Dirichlet boundary conditions (Swarztrauber, 1977). In fact, the tensor product method for finding the solution to the Poisson equation with Dirichlet boundary conditions involves a two-dimensional sine transform [see Lynch, Rice and Thomas (1964)]. While in the single processor case, one would choose to use the matrix decomposition method [Buzbee (1973)], involving a fast transform in one direction and tridiagonal solves in the other, with the hypercube, it might be interesting to substitute the tridiagonal solves with fast transforms that are more efficiently parallelizable than tridiagonal solvers usually are. Sine and cosine transforms are also used extensively in image processing which may require transforms in two or more dimensions. Therefore fast efficiently implemented algorithms for vector computers and parallel systems are important and should be considered a basic building block in many other more complicated algorithms.

7. Acknowledgements

The author wishes to acknowledge the helpful comments and discussions with Professor Charles Van Loan and his interest in this project. The author also wishes to thank Ashley Doerschug and Michael North of the Theory Center for their help with the Intel iPSC hypercube.

8. Bibliography

Bergland, G. D. 1968. "A fast Fourier transform for real-valued series," *Comm. ACM* **11**, 703–710.

Brigham, E. O. 1974. *The Fast Fourier Transform,* Prentice-Hall, Inc. Englewood Cliffs, New Jersey.

Buzbee B. 1973. "A fast Poisson solver amenable to parallel computation", *IEEE Trans. Comput. C-22,* 793–796.

Chamberlain, R. M. 1986. "Gray Codes, Fast Fourier Transforms and Hypercubes," Tech. Report 864502-1, Chr. Michelsen Institute. Bergen, Norway.

Chan, Tony F. 1986. "On Gray Code Mappings for Mesh-FFTs on Binary N-Cubes," Tech. Report 86.17, RIACS, NASA Ames Research Center.

Chu, Clare. 1987. "Load Balanced FFT Implementations on the Intel iPSC," Dept. of Computer Science Tech. Report. (TR 87-824) Cornell University, Ithaca, New York.

Cooley, J. W., Lewis, P. A. and Welch, P. D. 1970. "The Fast Fourier Transform Algorithm: Programming Considerations in the Calculation of Sine, Cosine and Laplace Transforms," *J. Sound Vib.* **12**(3), 315–337.

Dollimore, J. 1973. "Some algorithms for use with the fast Fourier transform," *J. Inst. Math. App.* **12**, 115–117.

Johnsson, S. Lennart. 1985. "Communication Efficient Basic Linear Algebra Computations on Hypercube Architectures," Research Report YALEU/DCS/RR-361.

Lynch R. E., Rice, J. R. and Thomas, D. H. 1964. "Direct solution of partial difference equations by tensor product methods", *Numerische Mathematik 6,* 185–199.

McBryan, Oliver A. and Van de Velde, Eric F. 1985. "Hypercube Algorithms and Implementations," Presented to the 2nd SIAM Conference on Parallel Computing, Norfolk, Nov. 18–21, 1985.

Saad, Youcef and Schultz, Martin H. 1985. "Topological Properties of Hypercubes," Research Report YALEU/DCS/RR-389.

Swarztrauber, Paul N. 1977. "The methods of cyclic reduction, Fourier analysis and the FACR algorithm for the discrete solution of Poisson's equation on a rectangle," *SIAM Rev.*, V. 19, pp. 490–501.

———. 1982. "Vectorizing the FFTs," in *Parallel Computations* (G. Rodrigue, ed.), Academic Press, New York, 1982, pp. 51–83.

———. 1986a. "Symmetric FFTs," *Mathematics of Computation,* Volume 47, Number 175, 323–346.

———. 1986b. "Multiprocessor FFTs," preprint from National Center for Atmospheric Research. Boulder, Colorado. (to appear in *Parallel Computing*)