

Parallel Power-of-Two FFTs on Hypercubes

Mahn-ling Woo and R. A. Renaut

Department of Mathematics

Arizona State University

Tempe, AZ 85287-1804

e mail: na.renaut @ na-net.ornl.gov FAX: 602-965-8119
na.woo @ na-net.ornl.gov

Abstract

Theoretical analysis of ordered power-of-two Fast Fourier transforms (PO2FFT) demonstrates that the most efficient algorithms for hypercube architectures may not always use nearest-neighbor communication. The ordered PO2FFT which uses nearest-neighbor (distance 1) communication, requires d interprocessor communications for $r/2 \geq d$ and $2d - \lfloor r/2 \rfloor$ interprocessor communications for $r/2 < d$, where $2^r = N$. In this paper we show that an ordered PO2FFT can be obtained with just d interprocessor communications for any r if the restriction that all communications are distance one is removed. This new algorithm uses $\lfloor r/2 \rfloor$ distance one and $d - \lfloor r/2 \rfloor$ distance two interprocessor communications. Packets of size $N/2^{d+1}$ are transmitted in both of the ordered PO2FFT algorithms. The time complexity of both the distance one and distance two algorithms is discussed.

1. Introduction

The Fast Fourier Transform (FFT) is one of the principal tools for scientific computing. Recently, the design of parallel machines and the decreasing cost of computer hardware have made parallelism not only feasible but also attractive. Thus, in this paper we address the interesting topic of the design of efficient parallel FFTs.

Our work is motivated by the solution of partial differential equations by the pseudospectral method [4]. In particular, we consider the power-of-two (PO2) FFT since this is the natural candidate for consideration on the hypercube due to its power-of-two topology. Furthermore, Swarztrauber et al. [9] have shown that Bluestein's FFT [1] for transforms of arbitrary length maps quite well onto the hypercube since it replaces the arbitrary length transform by an unordered PO2FFT. For transforms of length $N = 500$ it has been shown that Bluestein's FFT is superior to matrix multiplication for the discrete Fourier transform (DFT). The PO2FFT pair presented in [5] further improves the efficiency of Bluestein's FFT.

There are several criteria that must be satisfied when developing an efficient parallel algorithm. Reducing the transmission time is of major importance for minimizing execution time of the parallel algorithm. The crucial factors for reducing the transmission time are minimal transmissions, minimal transmission distances, minimal transmission packet sizes, and good data assignment. There are usually some tradeoffs among those requirements since the transmission time depends not only on the algorithms but also on the hardware. The algorithms we present here are designed to reduce the transmission time by considering all of these criteria.

The algorithms we describe here are all based on assigning the data in a wrapping around order across the cube. This method of data assignment was first presented by Woo [11]. For the unordered PO2FFT, the algorithm requires d nearest neighbor interprocessor communications on a d -dimensional cube and for each transmission the packet size is length 2^{r-d-1} where 2^r is the length of the transform. Renaut and Woo [5] compared this algorithm with Swarztrauber's [8] algorithm and Chamberlain's algorithm [2]. Each of these assign the data to the cube sequentially. The comparison demonstrated that the wraparound data assignment is superior.

Now, most of the previous research on the design of efficient PO2FFTs for hypercube architectures has assumed that nearest neighbor communication is essential. In this paper we extend the ideas used in the development of the unordered FFT to an ordered FFT and by consideration of two algorithms for an ordered PO2FFT we demonstrate that it may be, in fact, better to drop the restriction that all communication is between nearest-neighbors, distance one apart. Both of these algorithms assign data in wraparound fashion but for the first algorithm all communication is distance one while for the second some distance two communication is allowed. The distance one algorithm requires d communications for $r/2 \geq d$ and $2d - \lfloor r/2 \rfloor$ for $r/2 < d$. This result was also derived independently

by Tong et al. [10] but the specifics of the algorithm vary from that presented here. For the distance two algorithm there are a total of d communications in all cases but $d - \lfloor r/2 \rfloor$ of these are distance two while the remainder are distance one. Experimental evidence indicates, furthermore, that transmission times for data over distances one and two are very close on the Intel iPSC2 and iPSC2/860. Thus, we might conclude that the distance two FFT is more efficient than the distance one FFT.

In Section 2 the Cooley-Tukey FFT algorithm is briefly described and the index map as a means for introducing the recursion relation for the FFT is reviewed. Then in Section 3.1 the interleaved-sequence-to-processor map is introduced and general implementation considerations for the design of efficient parallel algorithms are discussed. As a means of introducing the ideas for parallel FFTs the unordered PO2FFT is reviewed in section 3.2. Then, in Sections 3.3 and 3.4 the distance one and distance two ordered PO2FFTs are introduced. In Section 4.1 expressions for the time complexity of all the preceding PO2FFTs are derived. Numerical experimentation on the Intel iPSC/860 is used in Section 4.2 to determine values for the parameters in these complexity expressions. We conclude that with the current iPSC/860 architecture using the release 3.3 operating system the distance-one FFT is still superior but that if the message-passing system is improved the distance-2 algorithm may be the optimal choice.

2. The Cooley-Tukey Algorithm

The FFT, which is an algorithm for computing the discrete Fourier transform of a series of length N in $O(N \log_2 N)$ operations instead of $O(N^2)$ operations as required by the direct method, was introduced by Cooley and Tukey in 1965 [3]. In the general case the length of the series is any arbitrary N which can be written as a product of primes. Here we consider the PO2FFT where N is taken to be a power of two. The discrete Fourier transform of the series $\{x_n : n = 0, \dots, N - 1\}$ is given by

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{1nk2\pi}{N}}$$

where $N = N_0 N_1 \dots N_{r-1}$ and r is the number of possible factorizations which yield the Cooley-Tukey FFT. For the PO2FFT $N_i = 2, i = 0, \dots, r - 1$. The FFT recursion can be defined relative to the index map $n = I(i_0, i_1, \dots, i_{r-1})$ introduced by Swarztrauber [8]

where the index function is defined by

$$I(i_0, i_1, \dots, i_{r-1}) = i_0 + 2i_1 + \dots + 2^{r-1}i_{r-1}.$$

The index function $k = I(k_{r-1}, \dots, k_0)$ is defined similarly. For the PO2FFT both k_j and i_j are either 0 or 1 where $j = 0, \dots, r - 1$. Then for the multidimensional arrays

$$\begin{aligned} x(i_0, i_1, \dots, i_{r-1}) &= x_n \text{ and} \\ X(k_{r-1}, k_{r-2}, \dots, k_0) &= X_k \end{aligned}$$

repeated factorization yields the recurrence

$$\begin{aligned} (2.1) \quad X^{(s)}(i_0, \dots, i_{r-s-1}, k_{r-s}, \dots, k_{r-1}) \\ = \sum_{i_{r-s}=0}^1 X^{(s-1)}(i_0, \dots, i_{r-s}, k_{r-s+1}, \dots, k_{r-1}) \\ \times \omega \sum_{i_{r-s-1}=0}^{r-1} 2^{r-1-i_{r-s-1}} k_{r-s} \end{aligned}$$

which is initialized with

$$X^0(i_0, \dots, i_{r-1}) = x(i_0, \dots, i_{r-1}).$$

Here $s = 1, \dots, r, N = 2$ and $\omega = e^{-\frac{2\pi}{N}}$. The output of the FFT is then given by the digit-reversed ordering

$$X_k = x(k_{r-1}, \dots, k_0) = X^{(r)}(k_0, \dots, k_{r-1}).$$

For example, in the case of $N = 8 = 2 * 2 * 2$ (2.1) yields

$$\begin{aligned} X^{(0)}(i_0, i_1, i_2) &= x(i_0, i_1, i_2) \\ X^{(1)}(i_0, i_1, k_2) &= \sum_{i_2=0}^1 X^{(0)}(i_0, i_1, i_2) \omega^{4i_2 k_2 + 2i_1 k_2 + i_0 k_2} \\ &= \omega^{(i_0 + 2i_1)k_2} \sum_{i_2=0}^1 X^{(0)}(i_0, i_1, i_2) \omega^{4i_2 k_2} \\ X^{(2)}(i_0, k_1, k_2) &= \sum_{i_1=0}^1 X^{(1)}(i_0, i_1, k_2) \omega^{4i_1 k_1 + 2i_0 k_1} \\ &= \omega^{2i_0 k_1} \sum_{i_1=0}^1 X^{(1)}(i_0, i_1, k_2) \omega^{4i_1 k_1} \\ X^{(3)}(k_0, k_1, k_2) &= \sum_{i_0=0}^1 X^{(2)}(i_0, k_1, k_2) \omega^{4i_0 k_0} \\ x_k &= x(k_2, k_1, k_0) = X^{(3)}(k_0, k_1, k_2). \end{aligned}$$

We observe the following properties of a PO2FFT of length 2^r :

- (1) r butterflies are required per FFT, where a butterfly is the process from $X^{(i)}$ to $X^{(i+1)}$ and $i = 0, \dots, r$.

- (2) After r butterflies have been processed, the resulting data id is the bit reverse of the original data id.
- (3) One switch stage, which can be performed before or after the butterflies, is needed to reorder the data ids.

Hence, a total of r butterflies and one switch stage are required per FFT. If the data is then transformed back out of the Fourier domain the switch stage can be eliminated if a forward-inverse FFT pair is designed for which the input of the inverse FFT is precisely the output of the forward FFT.

3. Parallel FFT algorithms

3.1 Implementation considerations

On the hypercube architecture, the major issue in speeding parallel algorithms is reducing the transmission cost. The cost depends on the number of data transmissions, the distance the message travels, and the length of the data transmission in bytes.

All the FFT algorithms we consider here satisfy the in-place property, i.e., data from two different processors are exchanged and combined with the result staying in the original two processors. This is a very important property for an efficient parallel algorithm since it is inefficient to combine data from two different processors and then put the result into a third processor. Thus, extra interprocessor communication results if the in-place property does not hold.

In order to satisfy the in-place property, however, we have to pay the price that the order of the data is scrambled. Thus, for ordered FFTs the output data must be reordered. This reordering is a nontrivial operation and can lead to extra expense if implemented naively.

The distribution of the data points is also very important since a naive distribution may also increase cost. All the algorithms we present here are based on the same idea of data assignment. Assume that each processor has its own name, me , and consider a transform of 2^r data points on a d -dimensional hypercube. Then the data are distributed such that the ids of the data in processor me are given by

$$\text{data id} = me + P \cdot j, \quad j = 0, \dots, 2^{r-d} - 1,$$

where $P = 2^d$ is the number of processors. This is commonly referred to as a wraparound data assignment. Then, in order to implement our PO2FFT algorithms the interleaved sequence-to-processor map as

introduced by Renaut and Woo [5] is used:

$$x_n = (i_d, \dots, i_{r-1} \mid i_0, \dots, i_{d-1}).$$

Here $n = i_{r-1} \dots i_0$ (binary) and the partition \mid has been introduced to separate the address on the left from the processor number on the right. With this notation the element x_n has address $i_{r-1} \dots i_d$ in processor $i_{d-1} \dots i_0$. For the PO2FFT algorithms the concept of the partial-exchange i -cycle is also needed [5].

Definition. The partial-exchange i -cycle is an index permutation of x_n in which the pivot is exchanged only with any other digit in the processor number.

In this definition the pivot is any element in the address. This means that if the digit i_j is inside the address, then no exchange occurs. If i_j is in the processor number, then the exchange is carried out. For the interleaved sequence-to-processor map this means that if j is greater than $d - 1$ there will be no exchange and otherwise there will be an exchange.

For an ordered FFT algorithm, the i -cycle which was introduced by Swarztrauber [8] will be used.

Definition. An i -cycle is an index-digit permutation of x_n in which the most significant digit of the address (called the pivot) is exchanged with any other digit, either in the address or the processor number.

These two definitions and the interleaved sequence-to-processor map will be used to develop PO2FFT algorithms. Note that both the partial-exchange i -cycle and the i -cycle have the same properties:

- (1) The transmission distance is equal to the number of different bits in the processor id (in binary form).
- (2) Any element in the address position could be the pivot
- (3) Any exchange in address only means that the data point sequence in each processor must be rearranged and no interprocessor communication is involved.

In most cases, the FFT is performed many times: for example, in the Fourier method for finding the solution of a partial differential equation. Recomputation of the powers of ω in each processor then becomes very desirable, especially when the ratio N/P becomes large where P is the number of processors. Chamberlain [2] demonstrated that the precomputation of the powers of ω does indeed reduce the time for the FFT over many iterations. Also, this precomputation recovers some of the load imbalance in the amount of work that each processor has to compute in each butterfly. But,

extra storage for the precomputation is needed. This drawback limits the size of problem we can solve. The trend, however, is for hypercubes with more powerful processors and larger local memories which should cope with these storage requirements. Thus, in general, we consider algorithms in which the powers of ω are precomputed.

3.2. The unordered FFT algorithm

Many parallel FFT algorithms have been developed by using binary divide-and-conquer strategies. Chamberlain [2] showed that a PO2FFT requires d interprocessor communications with packets of length N/P . Swarztrauber [8] presented an unordered PO2FFT for which $d + 1$ interprocessor communications with packets of length $N/(2P)$ are required. Obviously, for Swarztrauber's parallel FFT (PFFT) algorithm, we pay the cost of one more interprocessor communication at the gain of halving the packet size. It is hard to decide which algorithm should be better since the efficiency depends on the number of processors used, the start-up time per interprocessor communication, and the per-byte transmission cost.

Woo [11], however, has developed a more efficient unordered PO2FFT on the Intel iPSC1 which uses only d interprocessor communications with packets of length $N/2P$. This algorithm is clearly an improvement on both of these algorithms in that it minimizes the packet size and the number of transmissions. Numerical and theoretical comparisons were shown by Renaut and Woo [5]. The algorithm differs from Swarztrauber's and Chamberlain's algorithms in the way that the data points are distributed to the processors: the interleaved pattern is used instead of the natural order distribution. As opposed to Swarztrauber's [8] map any element in the address can be used to be the pivot. In this way, $r - d$ different unordered FFT algorithms which have different sets of output data point sequences in each processor are obtained. In fact, Swarztrauber could also use any element in the address of the processor for the pivot.

Furthermore, the FFT algorithms defined with respect to the partial-exchange i -cycle have different sets of output data points in each processor compared with those obtained from the i -cycle. Thus, in total we have $2(r - d)$ different FFT algorithms, all of which require only d interprocessor communications and data lengths $N/2P$ per transmission.

The question naturally arises as to which algorithm we should choose from this set of $2(r - d)$ algorithms. We require not only an efficient algorithm but also one that is portable for a cube of any dimension. With

this requirement it is simplest to choose the pivot to be either the leftmost or the rightmost digit in the address. In this way no ambiguity will occur as to which pivot to choose for $r \gg d$. Thus, four choices for the PO2FFT are left.

Examples of the four PO2FFT algorithms for $d = 6$ and $r = 2$ are illustrated in Tables 1 to 4. Since the interprocessor communication occurs only when there is a change in the processor id, each of these four FFTs requires the same number of communications. In the above example each one performs two interprocessor communications per FFT. However, the partial-exchange i -cycle will be preferred since the i -cycle requires reordering of the indices in the address. It is not important whether we use the leftmost or rightmost digit of the address to be the pivot.

Thus an unordered PO2FFT of length 2^r can be obtained with d nearest-neighbor communications of length $2^r/2^{d+1} = 2^{r-d-1}$. What is the best we can achieve with an ordered PO2FFT?

Table 1. Unordered FFT for 64 data points, 4 processors and pivot element i using the i -cycle

$$\begin{aligned}
 x(i_2i_3i_4i_5 | i_0i_1) &= X^{(0)}(i_2i_3i_4i_5 | i_0i_1) \\
 &X^{(1)}(k_5i_3i_4i_2 | i_0i_1) \\
 &X^{(2)}(k_4i_3k_5i_2 | i_0i_1) \\
 &X^{(3)}(k_3k_4k_5i_2 | i_0i_1) \\
 &X^{(4)}(k_2k_4k_5k_3 | i_0i_1) \\
 &X^{(5)}(k_1k_4k_5k_3 | i_0k_2) \\
 &X^{(6)}(k_0k_4k_5k_3 | k_1k_2) \\
 &\text{bit reversed} \\
 &x(k_5k_1k_0k_2 | k_4k_3)
 \end{aligned}$$

Table 2. Unordered FFT for 64 data points, 4 processors and pivot element i_2 using the partial exchange i -cycle

$$\begin{aligned}
 x(i_2i_3i_4i_5 | i_0i_1) &= X^{(0)}(i_2i_3i_4i_5 | i_0i_1) \\
 &X^{(1)}(i_2i_3i_4k_5 | i_0i_1) \\
 &X^{(2)}(i_2i_3k_4k_5 | i_0i_1) \\
 &X^{(3)}(i_2k_3k_4k_5 | i_0i_1) \\
 &X^{(4)}(k_2k_3k_4k_5 | i_0i_1) \\
 &X^{(5)}(k_1k_3k_4k_5 | i_0k_2) \\
 &X^{(6)}(k_0k_3k_4k_5 | k_1k_2) \\
 &\text{bit reversed} \\
 &x(k_5k_2k_1k_0 | k_4k_3)
 \end{aligned}$$

Table 3. Unordered FFT for 64 data points, 4 processors and pivot element i_5 using the i -cycle

$$\begin{aligned}
x(i_2i_3i_4i_5 | i_0i_1) &= X^{(0)}(i_2i_3i_4i_5 | i_0i_1) \\
&X^{(1)}(i_2i_3i_4k_5 | i_0i_1) \\
&X^{(2)}(i_2i_3k_5k_4 | i_0i_1) \\
&X^{(3)}(i_2k_4k_5k_3 | i_0i_1) \\
&X^{(4)}(k_3k_4k_5k_2 | i_0i_1) \\
&X^{(5)}(k_3k_4k_5k_1 | i_0k_2) \\
&X^{(6)}(k_3k_4k_5k_0 | k_1k_2) \\
&\text{bit reversed} \\
&x(k_2k_1k_0k_5 | k_4k_3)
\end{aligned}$$

Table 4. Unordered FFT for 64 data points, 4 processors and pivot element i_5 using the partial-exchange i -cycle

$$\begin{aligned}
x(i_2i_3i_4i_5 | i_0i_1) &= X^{(0)}(i_2i_3i_4i_5 | i_0i_1) \\
&X^{(1)}(i_2i_3i_4k_5 | i_0i_1) \\
&X^{(2)}(i_2i_3k_4k_5 | i_0i_1) \\
&X^{(3)}(i_2k_3k_4k_5 | i_0i_1) \\
&X^{(4)}(k_2k_3k_4k_5 | i_0i_1) \\
&X^{(5)}(k_2k_3k_4k_1 | i_0k_5) \\
&X^{(6)}(k_2k_3k_4k_0 | k_1k_5) \\
&\text{bit reversed} \\
&x(k_3k_2k_1k_5 | k_4k_0)
\end{aligned}$$

3.3 The Distance-1 Ordered PO2FFT algorithm

The distance-1 ordered PO2FFT we present here is also based on the interleaved sequence-to-processor map. For this algorithm, d interprocessor communications are required for $\frac{r}{2} \geq d$ and $2d - \lceil \frac{r}{2} \rceil$ interprocessor communications are required for $\frac{r}{2} < d$. All the transmissions involve only neighboring nodes.

In all cases, we apply the partial-exchange i -cycle during the first $r - d$ butterflies which introduces no interprocessor communications. For the following d butterflies of the FFT, the algorithm depends on the number of processors and the number of data points. The algorithm for generating the distance-1 ordered PO2FFT with the interleaved sequence-to-processor map is shown in Table 5. As with the unordered PO2FFT, it is not important whether we choose the leftmost or rightmost digit as the pivot. The rightmost digit is used here. In Table 5, processing the $(r - idx)$ th butterfly by the i -cycle means that the pivot element is

Table 5. The Distance-1 ordered PO2FFT algorithm with the interleaved sequence-to-processor map

```

* Assume there are  $2^d$  processors and  $2^r$  data points
* Case 1  $r/2 \geq d$ 
*
IF  $r/2 \geq d$  THEN
  FOR  $idx = r-1, -1, 0$ 
    IF  $idx \geq d$  THEN
      Process the  $(r-idx)$ th butterfly by partial-
      exchange  $i$ -cycle (idx)
    ELSE
      Exchange the pivot element with  $k_{r-idx-1}$  (idx a)
      Process the  $(r-idx)$ th butterfly by  $i$ -cycle (idx)
    ENDIF
  ENDFOR
  Reorder the sequence  $k_j$  in the address into the descending
  order ( No transmission needed here ) (idx d)
ENDIF
* Case 2  $(r+1)/2 = d$ 
*
IF  $(r+1)/2 = d$  THEN
  FOR  $idx = r-1, -1, 0$ 
    IF  $idx \geq d$  THEN
      Process the  $(r-idx)$ th butterfly by partial-
      exchange  $i$ -cycle (idx)
    ELSEIF  $idx = d-1$  THEN
      Exchange  $k_{r-1}$  with  $i_0$  (idxi b)
      Process the  $(r-idx)$ th butterfly by  $i$ -cycle (idx)
    ELSEIF  $idx > 0$  THEN
      Exchange the pivot element with  $k_{r-idx-1}$  (idx a)
      Process the  $(r-idx)$ th butterfly by  $i$ -cycle (idx)
    ELSE
      Exchange the pivot element with  $k_{d-1}$  (idx a)
      Process the  $r$ th butterfly by  $i$ -cycle (idx)
    ENDIF
  ENDFOR
  Reorder the sequence  $k_j$  in the address into the descending
  order ( No transmission needed here ) (idx d)
ENDIF
* Case 3  $(r+1)/2 < d$ 
*
IF  $(r+1)/2 < d$  THEN
  CRPOINT =  $\left\lfloor \frac{2d-r}{2} \right\rfloor + r - d$ 
  IF  $idx \geq d$  THEN
    Process the  $(r-idx)$ th butterfly by partial-
    exchange  $i$ -cycle (idx)
  ELSEIF  $idx \geq \text{CRPOINT}$  THEN
    Exchange the pivot element with  $k_p$  where  $p$  is
    the largest index in the address position (idx a)
    Exchange the pivot element with  $i_{r-idx-\text{CRPOINT}}$  (idx b)
    Process the  $(r-idx)$ th butterfly by  $i$ -cycle (idx)
  ELSE
    Exchange the pivot element with  $k_{r-idx-1}$  (idx a)
    Process the  $(r-idx)$ th butterfly by  $i$ -cycle (idx)
  ENDIF
  ENDFOR
  Reorder the sequence  $k_j$  in the address into the descending
  order ( No transmission needed here ) (idx d)
ENDIF

```

exchanged with the digit i_{idx} and then i_{idx} is replaced by k_{idx} which is exactly the $(r - idx)$ th butterfly in the FFT. Processing the $(r - idx)$ th butterfly by the partial-exchange i -cycle means i_{idx} is replaced by k_{idx} and neither interprocessor communication nor indexing reordering is needed. The notation (idx) indicates that the idx th butterfly is processing which is exactly the superscript of the step $X^{(idx)}$. Further, the notation $X^{(idxa)}$ indicates that one butterfly is process-

ing, index rearrangement occurs and no interprocessor communication is needed. $X^{(idxb)}$ means one interprocessor communication but neither i -cycle nor partial exchange i -cycle and $X^{(idxc)}$ is a distance two communication while one butterfly is processing. $X^{(idxd)}$ indicates index rearrangement, no interprocessor communication and no butterfly computation. The * after $X^{(idx)}$ indicates an interprocessor communication occurs. Examples for $d = 3, 4, 5, 6$ and $r = 7$ are given in Tables 6 to 9.

Table 6. Distance-1 ordered PO2FFT for 128 data points ($r = 7$), and 8 processors ($d = 3$)

$$\begin{aligned}
x(i_3i_4i_5i_6 | i_0i_1i_2) &= X^{(0)}(i_3i_4i_5i_6 | i_0i_1i_2) \\
&X^{(1)}(i_3i_4i_5k_6 | i_0i_1i_2) \\
&X^{(2)}(i_3i_4k_5k_6 | i_0i_1i_2) \\
&X^{(3)}(i_3k_4k_5k_6 | i_0i_1i_2) \\
&X^{(4)}(k_3k_4k_5k_6 | i_0i_1i_2) \\
&X^{(5a)}(k_3k_6k_5k_4 | i_0i_1i_2) \\
&X^{(5)}(k_3k_6k_5k_2 | i_0i_1k_4)* \\
&X^{(6a)}(k_3k_6k_5 | i_0i_1k_4) \\
&X^{(6)}(k_3k_6k_2k_1 | i_0k_5k_4)* \\
&X^{(7a)}(k_3k_1k_2k_6 | i_0k_5k_4) \\
&X^{(7)}(k_3k_1k_2k_0 | k_6k_5k_4)* \\
&X^{(7d)}(k_3k_2k_1k_0 | k_6k_5k_4) \\
&\text{bit reversed} \\
&x(k_3k_4k_5k_6 | k_0k_1k_2)
\end{aligned}$$

Table 7. Distance-1 ordered PO2FFT for 128 data points ($r = 7$), and 16 processors ($d = 4$)

$$\begin{aligned}
x(i_4i_5i_6 | i_0i_1i_2i_3) &= X^{(0)}(i_4i_5i_6 | i_0i_1i_2i_3) \\
&X^{(1)}(i_4i_5k_6 | i_0i_1i_2i_3) \\
&X^{(2)}(i_4k_5k_6 | i_0i_1i_2i_3) \\
&X^{(3)}(k_4k_5k_6 | i_0i_1i_2i_3) \\
&X^{(4b)}(k_4k_5i_0 | k_6i_1i_2i_3)* \\
&X^{(4)}(k_4k_5k_3 | k_6i_1i_2i_0)* \\
&X^{(5a)}(k_3k_5k_4 | k_6i_1i_2i_0) \\
&X^{(5)}(k_3k_5k_2 | k_6i_1k_4i_0)* \\
&X^{(6a)}(k_3k_2k_5 | k_6i_1k_4i_0) \\
&X^{(6)}(k_3k_2k_1 | k_6k_5k_4i_0)* \\
&X^{(7a)}(k_1k_2k_3 | k_6k_5k_4i_0) \\
&X^{(7)}(k_1k_2k_0 | k_6k_5k_4k_3)* \\
&X^{(7d)}(k_2k_1k_0 | k_6k_5k_4k_3) \\
&\text{bit reversed} \\
&x(k_4k_5k_6 | k_0k_1k_2k_3)
\end{aligned}$$

Table 8. Distance-1 ordered PO2FFT for 128 data points ($r = 7$), and 32 processors ($d = 5$)

$$\begin{aligned}
x(i_5i_6 | i_0i_1i_2i_3i_4) &= X^{(0)}(i_5i_6 | i_0i_1i_2i_3i_4) \\
&X^{(1)}(i_5k_6 | i_0i_1i_2i_3i_4) \\
&X^{(2)}(k_5k_6 | i_0i_1i_2i_3i_4) \\
&X^{(3b)}(k_5i_0 | k_6i_1i_2i_3i_4)* \\
&X^{(3)}(k_5k_4 | k_6i_1i_2i_3i_0)* \\
&X^{(4a)}(k_4k_5 | k_6i_1i_2i_3i_0) \\
&X^{(4b)}(k_4i_1 | k_6k_5i_2i_3i_0)* \\
&X^{(4)}(k_4k_3 | k_6k_5i_2i_1i_0) \\
&X^{(5a)}(k_3k_4 | k_6k_5i_2i_1i_0) \\
&X^{(5)}(k_3k_2 | k_6k_5k_4i_1i_0)* \\
&X^{(6a)}(k_2k_3 | k_6k_5k_4i_1i_0) \\
&X^{(6)}(k_2k_1 | k_6k_5k_4k_3i_0)* \\
&X^{(7a)}(k_1k_2 | k_6k_5k_4k_3i_0) \\
&X^{(7)}(k_1k_0 | k_6k_5k_4k_3k_2)* \\
&\text{bit reversed} \\
&x(k_5k_6 | k_0k_1k_2k_3k_4)
\end{aligned}$$

Table 9. Distance-1 ordered PO2FFT for 128 data points ($r = 7$), and 64 processors ($d = 6$)

$$\begin{aligned}
x(i_6 | i_0i_1i_2i_3i_4i_5) &= X^{(0)}(i_6 | i_0i_1i_2i_3i_4i_5) \\
&X^{(1)}(k_6 | i_0i_1i_2i_3i_4i_5) \\
&X^{(1b)}(i_0 | k_6i_1i_2i_3i_4i_5)* \\
&X^{(2)}(k_5 | k_6i_1i_2i_3i_4i_0)* \\
&X^{(3b)}(i_1 | k_6k_5i_2i_3i_4i_0)* \\
&X^{(3)}(k_4 | k_6k_5i_2i_3i_1i_0)* \\
&X^{(4b)}(i_2 | k_6k_5k_4i_3i_1i_0)* \\
&X^{(4)}(k_3 | k_6k_5k_4i_2i_1i_0)* \\
&X^{(5)}(k_2 | k_6k_5k_4k_3i_1i_0)* \\
&X^{(6)}(k_1 | k_6k_5k_4k_3k_2i_0)* \\
&X^{(7)}(k_0 | k_6k_5k_4k_3k_2k_1)* \\
&\text{bit reversed} \\
&x(k_6 | k_0k_1k_2k_3k_4k_5)
\end{aligned}$$

3.4 The distance-2 Ordered PO2FFT algorithm

The interleaved sequence-to-processor map is also used to implement a distance-2 ordered PO2FFT. For this FFT $d - \lfloor \frac{r}{2} \rfloor$ communications are over a distance

Table 10. The Distance-2 ordered PO2FFT algorithm with the interleaved sequence-to-processor map

```

* Assume there are  $2^d$  processors and  $2^r$  data points
* Case 1  $r/2 \geq d$ 
*
IF  $r/2 \geq d$  THEN
  FOR  $idx = r-1, -1, 0$ 
    IF  $idx \geq d$  THEN
      Process the  $(r-idx)$ th butterfly by partial-
      exchange  $i$ -cycle (idx)
    ELSE
      Exchange the pivot element with  $k_{r-idx-1}$  (idx a)
      Process the  $(r-idx)$ th butterfly by  $i$ -cycle (idx)
    ENDIF
  ENDFOR
  Reorder the sequence  $k_j$  in the address into the descending
  order ( No transmission needed here ) (idx d)
ENDIF
* Case 2  $(r+1)/2 = d$ 
*
IF  $(r+1)/2 = d$  THEN
  FOR  $idx = r-1, -1, 0$ 
    IF  $idx \geq d$  THEN
      Process the  $(r-idx)$ th butterfly by partial-
      exchange  $i$ -cycle (idx)
    ELSEIF  $idx = d-1$  THEN
      Replace  $k_{r-1}$  by  $i_0$  and then replace  $i_0$  by  $i_{idx}$ 
      within 1 step in order to process the  $(r-idx)$ th
      butterfly (distance-2 communication needed here) (idx c)
    ELSEIF  $idx > 0$  THEN
      Exchange the pivot element with  $k_{r-idx-1}$  (idx a)
      Process the  $(r-idx)$ th butterfly by  $i$ -cycle (idx)
    ELSE
      Exchange the pivot element with  $k_{d-1}$  (idx a)
      Process the  $r$ th butterfly by  $i$ -cycle (idx)
    ENDFOR
    Reorder the sequence  $k_j$  in the address into the descending
    order ( No transmission needed here ) (idx d)
  ENDFOR
ENDIF
* Case 3  $(r+1)/2 < d$ 
*
IF  $(r+1)/2 < d$  THEN
  CRPOINT =  $\lfloor \frac{2d-r}{2} \rfloor + r - d$ 
  IF  $idx \geq d$  THEN
    Process the  $(r-idx)$ th butterfly by partial-
    exchange  $i$ -cycle (idx)
  ELSEIF  $idx \geq$  CRPOINT THEN
    Exchange the pivot element with  $k_p$  where  $p$  is
    the largest index in the address position (idx a)
    Replace the pivot element by  $i_{r-idx-CRPOINT}$  and
    then replace  $i_{r-idx-CRPOINT}$  by  $i_{idx}$  within 1
    step to process the  $(r-idx)$ th butterfly
    (distance-2 communication) (idx c)
  ELSE
    Exchange the pivot element with  $k_{r-idx-1}$  (idx a)
    Process the  $(r-idx)$ th butterfly by  $i$ -cycle (idx)
  ENDFOR
  Reorder the sequence  $k_j$  in the address into the descending
  order ( No transmission needed here ) (idx d)
ENDIF

```

two and $\lfloor \frac{r}{2} \rfloor$ are over a distance one. The total number of transmissions is thus again only d . Of course, if the cost for communication over distance two is twice that for distance one, this algorithm would not represent a saving since the total bandwidth would then be $2(d - \lfloor \frac{r}{2} \rfloor) + \lfloor \frac{r}{2} \rfloor = 2d - \lfloor \frac{r}{2} \rfloor$. In section 4.2 it is

shown, however, that the distance two cost is less than twice the distance one cost for some parallel machines such as the Intel iPSC2 and iPSC/860. Furthermore, the packet length is again $N/2P$. This algorithm, therefore, is optimal in achieving the minimal number of transmissions at the cost of losing the nearest-neighbor transmission property.

The distance-2 PO2FFT algorithm is shown in Table 10. Again, the partial-exchange i -cycle is applied during the first $r - d$ butterflies and no interprocessor communication is involved. For $\frac{r}{2} \geq d$ the algorithm is exactly the same as the distance-1 PO2FFT. The distance-2 transmission occurs when $idx \geq$ CRPOINT and the idx th butterfly is being processed, where $CRPOINT = \lfloor \frac{2d-r}{2} \rfloor + r - d$.

Examples of the Distance-2 algorithm for $d = 4, 5, 6$ and $r = 7$ are given in Tables 11 to 13. For the $d = 3$ and $r = 7$ case the algorithm is exactly the same as Table 6. In all cases, a total of d interprocessor communications of length $N/2P$ are required. The number of distance-2 butterflies equals 1 for the case of $d = 4$ and $r = 7$, 2 for the case of $d = 5$ and $r = 7$, and 3 for the case of $d = 6$ and $r = 8$. This agrees with the conclusion that only $d - \lfloor \frac{r}{2} \rfloor$ transmission involving distance 2 are required. Notice that exactly half of the nodes need to send messages twice to different processors (one is distance 2, the other is distance 1) at the same butterfly step. A signal flow graph of the distance-2 PO2FFT and also the algorithm for the case of $r = 4, d = 3$ is shown in Appendix A. This clarifies how the interleaved sequence-to-processor map, partial-exchange i -cycle and i -cycle relate to the PO2FFT algorithm.

Table 11. Distance-2 ordered PO2FFT for 128 data points ($r = 7$), and 16 processors ($d = 4$)

$X^{(0)}(i_4 i_5 i_6 i_0 i_1 i_2 i_3)$	$X^{(6a)}(k_3 k_2 k_5 k_6 i_1 k_4 i_0) *$
$X^{(1)}(i_4 i_5 k_6 i_0 i_1 i_2 i_3)$	$X^{(6)}(k_3 k_2 k_1 k_6 k_5 k_4 i_0) *$
$X^{(2)}(i_4 k_5 k_6 i_0 i_1 i_2 i_3)$	$X^{(7a)}(k_1 k_2 k_3 k_6 k_5 k_4 i_0)$
$X^{(3)}(k_4 k_5 k_6 i_0 i_1 i_2 i_3)$	$X^{(7)}(k_1 k_2 k_0 k_6 k_5 k_4 k_3) *$
$X^{(4b)}(k_4 k_5 k_3 k_6 i_1 i_2 i_0) **$	$X^{(7d)}(k_2 k_1 k_0 k_6 k_5 k_4 k_3)$
$X^{(5a)}(k_3 k_5 k_4 k_6 i_1 i_2 i_0)$	bit reversed
$X^{(5)}(k_3 k_5 k_2 k_6 i_1 k_4 i_0) *$	$x(k_4 k_5 k_6 k_0 k_1 k_2 k_3)$

Table 12. Distance-2 ordered PO2FFT for 128 data points ($r = 7$), and 32 processors ($d = 5$)

$$\begin{aligned}
& X^{(0)}(i_5 i_6 \mid i_0 i_1 i_2 i_3 i_4) \quad X^{(5)}(k_3 k_2 \mid k_6 k_5 k_4 i_1 i_0) * \\
& X^{(1)}(i_5 k_6 \mid i_0 i_1 i_2 i_3 i_4) \quad X^{(6a)}(k_2 k_3 \mid k_6 k_5 k_4 i_1 i_0) \\
& X^{(2)}(k_5 k_6 \mid i_0 i_1 i_2 i_3 i_4) \quad X^{(6)}(k_2 k_1 \mid k_6 k_5 k_4 k_3 i_0) * \\
& X^{(3)}(k_5 k_4 \mid k_6 i_1 i_2 i_3 i_0) * * \quad X^{(7a)}(k_1 k_2 \mid k_6 k_5 k_4 k_3 i_0) \\
& X^{(4a)}(k_4 k_5 \mid k_6 i_1 i_2 i_3 i_0) \quad X^{(7)}(k_1 k_0 \mid k_6 k_5 k_4 k_3 k_2) * \\
& X^{(4)}(k_4 k_3 \mid k_6 k_5 i_2 i_1 i_0) * * \quad \text{bit reversed} \\
& X^{(5a)}(k_3 k_4 \mid k_6 k_5 i_2 i_1 i_0) \quad x(k_5 k_6 \mid k_0 k_1 k_2 k_3 k_4)
\end{aligned}$$

Table 13. Distance-2 ordered PO2FFT for 128 data points ($r = 7$), and 64 processors ($d = 6$)

$$\begin{aligned}
& X^{(0)}(i_6 \mid i_0 i_1 i_2 i_3 i_4 i_5) \\
& X^{(1)}(k_6 \mid i_0 i_1 i_2 i_3 i_4 i_5) \\
& X^{(2c)}(k_5 \mid k_6 i_1 i_2 i_3 i_4 i_0) * * \\
& X^{(3c)}(k_4 \mid k_6 k_5 i_2 i_3 i_1 i_0) * * \\
& X^{(4c)}(k_3 \mid k_6 k_5 k_4 i_2 i_1 i_0) * * \\
& X^{(5)}(k_2 \mid k_6 k_5 k_4 k_3 i_1 i_0) * \\
& X^{(6)}(k_1 \mid k_6 k_5 k_4 k_3 k_2 i_0) \\
& X^{(7)}(k_0 \mid k_6 k_5 k_4 k_3 k_2 k_1) * \\
& \quad \text{bit reversed} \\
& \quad x(k_6 \mid k_0 k_1 k_2 k_3 k_4 k_5)
\end{aligned}$$

4. Comparisons and Conclusions

4.1 Theoretical Analysis of Both Unordered and Ordered FFTs

The precomputation of the powers of ω is assumed for all the PO2 FFT. $N = 2^r$ trigonometric function evaluations are required for this computation in all cases. Thus, the complexity of the precomputation is the same for all the PO2PFFTs and is not included in our calculation of the computational complexity. A complex multiplication and two complex additions are needed at each stage of the PO2FFT, i.e., ten computational cycles are needed. In total $5N \log N/P$ interprocessor communications are required per FFT. The computational complexity is the same for all the PO2PFFT algorithms we present here. They differ in the communication costs. Thus the relative efficiency of these algorithms is determined by the communication costs.

Communication costs on the hypercube can be modelled by an expression of the form $a + bl$, where a is the startup cost (in flops), b is the per-byte transmission cost (in flops), and l is the packet length in bytes.

As discussed in Section 3.2, the unordered PO2FFT requires d interprocessor communications with packets of size $N/2P$. The data is stored in complex form and thus eight bytes are needed per data point. Therefore, the total execution time for the unordered FFT without preprocessing is given by

$$T_u(N, P) = \frac{5N \log_2 N}{P} + d \left(a + \frac{b1 * N}{2P} * 8 \right)$$

where $b1$ is per-byte transmission cost for the neighboring-node communication.

For the ordered PO2FFT, the communication costs can again be modeled by the form $a + bl$, but for distance-2 transmission the form $a + b2l$ is used where $b2$ is the per-byte transmission cost for the distance-2 transmission. The communication cost of the node for distance 2 transmission can be evaluated as follows: $\max\{a + \frac{b2 * N}{2P} * 8, a + \delta + \frac{b1 * N}{2P} * 8\}$, where δ is the time delay before the distance one isend or csend is initiated due to starting the distance two transmission first. Since the transmission times, $b1$, $b2$ and the value δ depend on the hardware, it is hard to decide which factors dominate. Also, when some nodes are executing the distance-2 transmission, sending the hop-2 transmission first followed by a hop-1 transmission, the remaining nodes are performing a hop-1 transmission at the same time. Thus, the execution times for both ordered FFTs are given by

$$\begin{aligned}
& T_{d_1(N, P)} \\
& = \begin{cases} \frac{1}{P}(5N \log_2 N) + d * \left(a + \frac{b1 * N}{2P} * 8 \right) \frac{r}{2} \geq d \\ \frac{1}{P}(5N \log_2 N) \\ \quad + (2d - \lfloor \frac{r}{2} \rfloor) \left(a + \frac{b1 * N}{2P} * 8 \right) \frac{r}{2} < d \end{cases} \\
& T_{d_2(N, P)} \\
& = \begin{cases} \frac{1}{P}(5N \log_2 N) + d * \left(a + \frac{b1 * N}{2P} * 8 \right) \frac{r}{2} \geq d \\ \left\{ \frac{1}{P}(5N \log_2 N) + \lfloor \frac{r}{2} \rfloor * \left(a + \frac{b1 * N}{2P} * 8 \right) \right. \\ \quad \left. + (d - \lfloor \frac{r}{2} \rfloor) * \max \left\{ a + \frac{b1 * N}{2P} * 8, \right. \right. \\ \quad \left. \left. \max \left\{ a + \frac{b2 * N}{2P} * 8, a + \delta + \frac{b1 * N}{2P} * 8 \right\} \right\} \right\}, \frac{r}{2} < d \end{cases}
\end{aligned}$$

Let TS1 be the total transmission time for 1 hop per interprocessor transmission and TS2 be the worst

transmission time for the node which sends two messages at the same time, a message in 2 hops followed by a message in 1 hop. The difference of the time complexity is then given by

$$T_{d_1(N,P)} - T_{d_2(N,P)} = \left(d - \lfloor \frac{r}{2} \rfloor \right) * \{ 2 * TS1 - TS2 \}$$

where

$$TS1 = a + \frac{b1 * N}{2P} * 8,$$

$$TS2 = \max \left\{ a + \frac{b2 * N}{2P} * 8, a + \delta + \frac{b1 * N}{2P} * 8 \right\}$$

and the condition $TS1 \leq TS2$ is assumed. Let $F = 2 * TS1 - TS2$. When $F = 0$, it implies those two algorithms have the same execution time which happens when $TS1 = 2 * TS2$. If $F > 0$, then the distance-1 algorithm is slower, otherwise the distance-2 algorithm is slower. F is evaluated for the iPSC/860 in the next section.

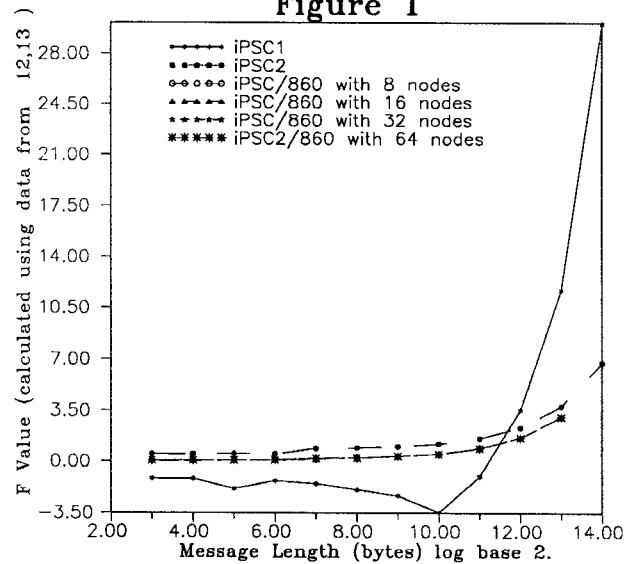
4.2 Comparisons on Intel iPSC1, iPSC2 and iPSC2/860

In order to compare these two PO2FFT algorithms, it is necessary to see how close the communication time for 1 hop is to the time for multiple hops. The Intel technical report [13] shows the communication times over 1 hop and 5 hops, rcv not pending, on the iPSC1 Release 3.2 and iPSC2 Release 1.0. To examine F , however, the time for 2 hops is needed. But in this case it is sufficient to assume that the time for 2 hops is not slower than that for 5 hops as this will give a worst case evaluation of F . Note of course that the evaluation of F from this data, which assumes no contention and $\delta = 0$, may be optimistic but at least allows some means for comparison with the newer architecture of the iPSC/860. Thus, in order to compare similar data, communication times for 1 hop and 2 hops on the iPSC/860 were found for cubes of dimension 3, 4, 5 and 6. These results are presented in [12] and were obtained by averaging results over several runs of the same program. Examination of these results shows that the iPSC/860 is indeed much faster than the iPSC2 and that, for a given message length, the time is virtually independent of cube dimension. In Figure 1 graphs of F calculated from the data in [12] and [13] are given. The distance-2 algorithm should be most efficient when $F > 0$ which is true for the iPSC/860 and iPSC2 and also for the iPSC1 except when the message length is less than $4K$.

For a more realistic determination of F , values of $TS1$ and $TS2$ are needed. The timing tests for $TS1$ and

$TS2$ were arranged so that the communication patterns of the distance-1 and distance-2FFT's, respectively, were simulated. As for the results in Figure 1 the data was obtained by averaging results over many runs of the same program. These tests were performed on the iPSC/860 with the latest release, Release 3.3. In each case the transmission of messages was achieved using `csend` and `crecv`. Although `csend` and `crecv` are used for transmission with blocking until completion we have found that this pair is faster than the `isend` and `irecv` pair (transmission without waiting for completion) for small sized messages [11]. In initial tests we evaluated $TS2$ by simulating one node alone performing a distance-two followed by distance-one transmission. Evaluation of F from these results indicated that $F > 0$ for the iPSC/860. The results with the complete simulation of the FFT, however, give $F < 0$. Thus contention causes the distance-two FFT to be less efficient than the distance-one FFT for the current iPSC architecture.

Figure 1



4.3 Conclusions and Future Work

Our numerical and theoretical results indicate that a parallel PO2FFT algorithm for a hypercube architecture can be designed with just d communications on a cube of size d for any messages of any length provided that the restriction that all messages occur over distance 1 is removed. In particular, an algorithm which allows some distance 2 transmission is described here. If the contention problem of multiple hop transmission can be solved by hardware, parallel PO2FFT algorithms using larger transmission distances may be useful. For example, the ordered PO2FFT can be ob-

tained by using $d-1$ neighboring communications and 1 communication over distance d . In [7] the communication performance of the CM2 using the NEWS communication facility is evaluated. These results are encouraging since they suggest that $F > 0$ for the CM2. Note that here F is not calculated from simulation of the distance-one and distance-two FFTs and so we cannot be sure that F is accurately predicted. We consider, however, that the new algorithm does have potential for SIMD architectures.

In future work we will evaluate the relative efficiencies of the distance-2 PO2FFT, the distance-1 PO2FFT, and the distance 2 PO2FFT described by Chamberlain [2] using messages of size N/P for a SIMD architecture such as the Connection Machine (CM).

For SIMD or MIMD architectures, it is also worth considering whether in the case $r/2 < d$ it might be more efficient to do the FFT on a cube of smaller dimension \tilde{d} , for which $r/2 \geq \tilde{d}$. In this way the FFT can be implemented as distance 1, but note that in most practical situations for which data are stored on the cube of dimension d this would introduce communication to put the data on the smaller cube.

Acknowledgments. We are grateful for the comments of an unknown referee which caused us to reevaluate the numerical results of Section 4.2. This work was supported by an NSF grant ASC 8812147. The experimental results were obtained by remote access to the iPSC/860 at NASA Ames Research Center.

References

- [1] L.I. Bluestein, IEEE Trans. Audio Electroacoust., AU-18 (1970) 451-455.
- [2] R.M. Chamberlain, Parallel Computing, 6 (1988) 225-233.
- [3] J.W. Cooley and J.W. Tukey, Math. Comput. 19 (1965), 297-301.
- [4] R.A. Renaut and M.L. Woo, SIAM Frontiers in Applied Mathematics, accepted (1990).
- [5] R.A. Renaut and M.L. Woo, Parallel FFT Pairs on a Hypercube, ASU Technical Report, (1991).
- [6] Y. Saad and M.H. Schultz, Data communication on hypercubes, Research report, Yale University, YALE U/DCS/PR-428, (1985).
- [7] R. Stevens and P. McDonough, Instruction Tunings and Message Passing Performance of the CM2. Argonne National Laboratory (1989).

- [8] P.N. Swarztrauber, Parallel Computing, 5 (1987), 197-210.
- [9] P.N. Swarztrauber, R.A. Sweet, W.L. Briggs, V.E. Henson, and J. Otto, Bluestein's FFT for Arbitrary N on the Hypercube, National Center for Atmospheric Research, Colorado, (1990).
- [10] C. Tong and P.N. Swarztrauber, Ordered Fast Fourier Transforms on a Massively Parallel Hypercube Multiprocessor, (December 1989).
- [11] M.L. Woo, Parallel Pseudospectral Methods on Hypercubes, Master's Thesis, Arizona State University, (1989).
- [12] M. Woo, R.A. Renaut and R. Casey, A Note on Programming the Intel iPSC/860, ASU Technical Report, 1991.
- [13] Performance Comparison, iPSC/2 Release 1.0 versus iPSC/1 Release 3.2, Intel Scientific Report (1988).

Appendix: The Distance-2 PO2FFT algorithm for $r = 4$ and $d = 3$

