Yale University
Department of Computer Science

Systolic FFT Algorithms on Boolean Cube Networks[1]

S. Lennart Johnsson, Ching-Tien Ho,
Michel Jacquemin, and Alan Ruttenberg

YALEU/DCS/TR-619
March 1988

This work has been supported in part by the Office of Naval Research under Contracts N00014-84-K-0043 and N00014-86-K-0564. Approved for public release: distribution is unlimited.

[1]To appear in the Proceedings of *International Conference on Systolic Arrays* San Diego, CA, May, 1988.

Systolic FFT Algorithms on Boolean Cube Networks

Lennart Johnsson[†‡§] Ching-Tien Ho[‡] Michel Jacquemin[‡†] Alan Ruttenberg[¶†]

[†] Thinking Machines Corp., 245 First Street, Cambridge, MA 02142
[‡] Dept. of Computer Science, Yale Univ., New Haven, CT 06520
[§] Dept. of Electrical Engineering, Yale Univ., New Haven, CT 06520
[¶] Dept. of Computer Science, MIT, Cambridge, MA 02139

Abstract Systolic algorithms are typically devised for one- or two-dimensional arrays of processing elements. In this paper we describe a systolic Cooley-Tukey Fast Fourier Transform algorithm for Boolean n-cubes with a substantial amount of storage per cube node. In mapping a Cooley-Tukey type FFT to such a network the main concerns are effective use of the high connectivity/bandwidth of the Boolean n-cube, the effective use of the computational resources, the effective use of the storage bandwidth, if there is a storage hierarchy, and pipelines should the arithmetic units have such a feature. Another important consideration in a multiprocessor, distributed storage architecture is the allocation and access to coefficients, if they are precomputed. We describe FFT algorithms that use both the storage bandwidth and the communication system optimally, and which require storage of $P + nN$ coefficients for a transform on $P \geq N$ data elements. A complex to complex FFT on 16 million points is predicted to require about 1.5 seconds on a Connection Machine model CM-2.

1 Introduction

The Fast Fourier Transform (FFT) is one of the most important algorithms in signal processing and the solution of partial differential equations. This paper addresses the data mapping and control structures for the implementation of Cooley-Tukey type FFTs on Boolean cube networks. In addition to the locality of communication (or lack thereof) as defined by the topological properties of the Boolean cube, we also consider the effective use of a local storage hierarchy. Such a hierarchy occurs in multiprocessors in which each node has a cache, or a register set allowing multiple, concurrent accesses.

A large amount of work has been devoted to reducing the needs for storage bandwidth in register or cache based architectures by careful partitioning and rearrangement of loops in the Cooley-Tukey FFT, such that intermediate results can be kept in the registers, or the cache, and thereby the need for maim storage bandwidth reduced by locality of reference. Another concern that has received a significant attention is to avoid bank conflicts in architectures for which the storage cycle time is a multiple of the processor cycle time. The performance gain from proper attention to storage references can be quite significant. Algorithms for vector architectures are described in [17,16,2,14] and for a combined cache and vector architecture in [1]. The algorithm by Fornberg relies on fast compress and merge instructions. Swarztrauber [16] not only considers vector and cache architectures, but also shared memory and Boolean cube configured architectures. For a summary of current efforts for the computation of Fourier Transforms on a variety of architectures see [18].

We have implemented the algorithms described here on the Connection Machine. It has 64k processors, and each processor has 8k bytes of primary storage using 256 kbit memory chips for a total of 512 Mbytes of storage, and 2 Gbytes using 1 Mbit memory chips. The bandwidth to storage is approximately 50 Gbytes/sec at 8 MHz clock rate. The bandwidth for the emulation of butterfly networks has a peak capacity of about 16 Gbytes/sec for many small networks, and a peak of about 13 Gbytes/sec for a large butterfly network. The nominal peak floating-point capacity for three operand instructions that do not make use of the registers on the floating-point unit is about 5 Gflops/sec, and the nominal peak for multiply-add instructions that make use of the registers is 32 Gflops/sec. In practice, the measured rate for the first type of operations in high level languages is in the range of 1.6 to about 3 Gflops/sec, whereas for carefully coded kernels of the second type rates in excess of 20 Gflops have been achieved.

The Connection Machine is a data parallel computer in that an operation is performed in one instruction time on all data elements to which a given operation is applied. In practice, there may not be a sufficient number of processors to support concurrent operation on all elements to which an instruction is applied. *Virtual processors* are introduced to provide the programmer with an arbitrary number of processors. Virtual processors time-share a physical processor. Each virtual processor occupies a unique part of the local storage of a processor. The allocation of virtual processors to physical processors is currently performed at compile time, and transparent to the user. The instruction time is a function of the number of virtual processors.

The 64k Connection Machine processors are distributed evenly among 4k chips interconnected as a 12 dimensional Boolean cube. Communication can take place on all ports concurrently. The details of the interconnection network on-chip are immaterial for the description here. Each floating-point unit is 32-bits wide, has one path to primary storage, a number of registers, and a multiplier and an adder that can be operated concurrently. The FFT algorithms we describe make efficient use of both the communication network and the floating-point unit.

In the remainder of the paper we first review the computational considerations of Cooley-Tukey FFT algorithms, then consider the mapping of such algorithms to Boolean cube networks. The allocation and use of the twiddle factors are considered last.

2 Computational Considerations of Cooley-Tukey FFT Algorithms

For a description of the Cooley-Tukey radix-2, -4, and -8 FFT algorithms see for instance [15] or [13]. We consider both decimation-in-frequency (DIF) and decimation-in-time (DIT) algorithms. There are $\log_2 P$ stages in a radix-2 algorithm on a data set of $P = 2^p$ complex elements. There is one complex multiplication for each butterfly operation, and two complex additions for a total of ten real operations. With one complex data point per processor, these operations can be split between a pair, but we will for the moment only consider the sequential complexity. The number of butterfly stages for a radix-4 algorithm is half of that for a radix-2 algorithm, and that of a radix-8 one third of that of a radix-2 algorithm. The number of complex multiplications is $p\frac{P}{2}$ for a radix-2 algorithm, and $\frac{p}{2}\frac{3P}{4}$ for a radix-4 algorithm. A radix-8 algorithm requires some internal complex multiplications, which the radix-2 and radix-4 algorithms do not. A careful analysis of the number of real arithmetic operations for the radix-2, -4, and -8 FFT algorithms yields the following ratio $\frac{3}{2} : \frac{9}{8} : 1$ for real multiplications, and $\frac{36}{31} : \frac{33}{31} : 1$ for real additions. The actual number of operations are given in Table 1. For the number of additions it is assumed that the radix-4 and radix-8 FFTs are factored internally so that the number of additions remains constant. The total number of arithmetic operations for the radix-8 algorithm is approximately 20% less than that of the radix-2 algorithm.

In an architecture with a higher bandwidth for the register, or cache, access the lower storage bandwidth requirement for a higher radix algorithm may be of greater importance than the reduced number of arithmetic operations. Assuming that all temporary results fit in the registers, or the cache, the storage bandwidth requirements of the radix-2, -4, and -8 algorithms compare as $3 : \frac{3}{2} : 1$ for the data. In general, a radix-Q algorithm reduces the need for storage bandwidth by a factor of $\log_2 Q$ compared to a radix-2 algorithm. This is indeed of the optimum order [3].

If the twiddle factors are read from storage and discarded immediately upon use, then the storage references for the twiddle factors compare as $\frac{12}{7} : \frac{9}{7} : 1$. However, if $R = 2^r$ registers (complex) are used for twiddle factors and all butterfly computations requiring those coefficients in a given rank computed successively, then r ranks can be computed with a single load of R twiddle factors, one rank with two loads, one with four, etc. The total number of storage references is $P - R$. The minimum number of registers needed for twiddle factors for a straightforward implementation of the radix-2, -4, or -8 FFT is 2, 6, and 14, respectively. If the twiddle factors are stored, then the storage references for data dominates. If there is a sufficient number of registers for radix-2, and radix-4 FFTs, but not radix-8, then the total number of storage references compare as $\frac{48}{23} : \frac{24}{23} : 1$, approximately. If only a radix-2 can be performed with a single load of twiddle factors for each rank, then the storage references compare as $\frac{48}{23} : \frac{33}{23} : 1$. Reuse of coefficients in registers requires that, for a DIF FFT, as the number of independent FFTs increases the same subset of all FFTs is processed before any other subset.

FFT	Arithmetic Operations				Storage References	
	Add	Mult	Total	Rel.	Data	Rel.
Radix-2	$3Pp$	$2Pp$	$5Pp$	$\frac{60}{47}$	$2Pp$	3
Radix-4	$2\frac{3}{4}Pp$	$\frac{3}{2}Pp$	$\frac{17}{4}Pp$	$\frac{51}{47}$	Pp	$\frac{3}{2}$
Radix-8	$2\frac{7}{12}Pp$	$\frac{4}{3}Pp$	$\frac{47}{12}Pp$	1	$\frac{2}{3}Pp$	1

Table 1: Comparison of arithmetic operations and storage references for radix-2, -4, and -8 FFTs.

3 Boolean Cubes and Address Maps

In a Boolean cube of $N = 2^n$ nodes, n bits are required for the encoding of the node addresses. In a Boolean cube every node $u = (u_{n-1}u_{n-2}\ldots u_m \ldots u_0)$ is connected to nodes $v = (u_{n-1}u_{n-2}\ldots \overline{u}_m \ldots u_0), \forall m \in [0, n-1]$. Every node has n neighbors. The distance between a pair of nodes u and v is $Hamming(u,v) = \sum_{m=0}^{n-1}(u_m \oplus v_m)$. The maximum distance between any pair of nodes is n, the number of nodes at distance m is $\binom{n}{m}$, and the average distance is $\frac{1}{2}n$. The number of edge-disjoint paths between a pair of nodes u and v is n. Of these paths $Hamming(u,v)$ are of length $Hamming(u,v)$ and $n - Hamming(u,v)$ are of length $Hamming(u,v)+2$.

In our model each processor has a local storage of $S = 2^s$ words, and the total address space requires $s+n$ bits assuming word addressing only. For an FFT on $P = 2^p$ complex data elements it is clearly required that $p \leq s+n-1$, if the "twiddle factors" are computed and there are a few registers in addition to the local storage. If the twiddle factors are precomputed and stored it is necessary that $p \leq s+n-2$. For the computation of an FFT of a size that only requires a fraction of the total storage, several options exist for the allocation of the data, assuming the choice can be made entirely with respect to the FFT computation. If $N = 2^n$ processors are used, $N \leq P$, then several elements are allocated per processor. Of the p bits required for the encoding of the data elements a subset of n bits is used for *real processor* addresses, and a subset of $p-n$ bits for *virtual processor* addresses, or local storage addresses. The bit-fields can be chosen in many ways. We consider *cyclic* and *consecutive* assignment [5]. The two forms are illustrated below.

Cyclic assignment:

$$(\underbrace{x_{p-1}x_{p-2}\ldots x_n}_{vp}\underbrace{x_{n-1}x_{n-2}\ldots x_0}_{rp}).$$

Consecutive assignment:

$$(\underbrace{x_{p-1}x_{p-2}\dots x_{p-n}}_{rp}\underbrace{x_{p-n-1}x_{p-n-2}\dots x_0}_{vp}).$$

In the cyclic assignment all data elements with the same n low order bits reside in the same processor. In the consecutive assignment the elements in a processor have the same n high order bits.

4 Cooley-Tukey FFTs on Boolean Cubes

4.1 Mapping Butterfly Networks to Boolean Cubes

A radix-2 butterfly network for P inputs and outputs has $P(p+1)$ nodes. These can be uniquely encoded with a total of $p + \lceil \log_2(p+1) \rceil$ bits. Let the address be partitioned as follows $(y_{p-1}y_{p-2}\dots y_0|z_{t-1}z_{t-2}\dots z_0)$, where $t = \lceil \log_2(p+1) \rceil$. Then, the butterfly network is defined by connecting node $(y|z)$ to the nodes $(y \oplus 2^z|z+1)$ and $(y|z+1)$, $z \in [0, p-1]$, where $\oplus$ denotes the bit-wise exclusive-or operation. For the computation of the radix-2 FFT the last t bits can also be interpreted as time. The network utilization defined as the fraction of the total number of nodes that are active at any given time is $\frac{1}{t}$. During step z communication between ranks z and $z+1$ takes place, and the complex multiplications in rank z for decimation-in-time FFT and rank $z+1$ for decimation-in-frequency FFT.

The butterfly and CCC networks are closely related. If the last column of the butterfly network is identified with the first, and node $(y|z)$ connected to nodes $(y|(z-1) \bmod p)$, $(y|(z+1) \bmod p)$ and $(y \oplus 2^z|z)$ then the CCC network is obtained. The mapping of FFTs to linear arrays are described in detail in [12,8], and to Boolean cubes and other networks in [11,6,16,18].

By identifying all nodes with the same y value and different z values in the butterfly network node y becomes connected to nodes $y \oplus 2^z$, $\forall z \in [0, p-1]$, which defines a Boolean p-cube. All nodes participate in every step in computing an FFT on P elements on a p-cube. In step z all processors communicate in dimension z. For $N = 2^n$ processors and $p > n$ there are 2^{p-n} *virtual processors* per *real processor*. If virtual processors correspond to high order bits of the data index, i.e., cyclic assignment is used, then regardless of whether a decimation-in-frequency or decimation-in-time FFT is used the first $p-n$ ranks of butterfly computations are local to a processor. The last n ranks require interprocessor communication. For consecutive assignment the first n steps require interprocessor communication, and the last $p-n$ steps are local to a processor. If the data is allocated in a bit-reversed order, then the order of the interprocessor communication and the local reference phases are reversed.

4.2 Algorithm Selection and Scheduling for Maximum Efficiency

With no virtual processors all the data is engaged in every step of the algorithm. The best load balance that can be achieved is obtained by splitting each butterfly operation between the two processors holding data for a butterfly computation, and the parallel arithmetic complexity becomes $5 \log_2 N$ for real arithmetic operations, ignoring lower order terms. The sequential complexity is $5N \log_2 N$, and the speed-up is N with respect to arithmetic complexity. The communication complexity is $3 \log_2 N$ element exchanges in sequence.

With a large number of virtual processors the looping order becomes important. Virtual processors as well as the butterfly stages are mapped to time. With the loop over virtual processors being an inner loop with respect to the loop over butterfly stages only $\frac{1}{n}$ of the total communication bandwidth is used, as in the case with one virtual processor. If the mapping of the data to the processors is such that the first $p - n$ stages are local to a processor, for instance by using a cyclic mapping of the data to the processors, then the computation for the last n stages consists of 2^{p-n} independent FFTs, each with one data point per processor. However, exchanging the loop order does not resolve the problem with a poor communication efficiency. We consider the following ideas for its improvement.

- Perform an FFT on the local data set followed by pipelining of successive FFT computations defined by virtual processors.
- Perform an FFT computation on the local data set followed by a data reallocation to allow for a new FFT on the same size data set, or a smaller set if $p - n > n$. With 2^{p-n} local elements this procedure is equivalent to computing the FFT by radix-2^{p-n} FFTs.
- Perform a set of radix-Q_{vp} FFT locally, $Q_{vp} \leq 2^{p-n}$, then perform a data reallocation requiring interprocessor communication for the computation of radix-Q_c FFTs, $Q_c \leq 2^n$, and pipeline these computations and reallocations.
- Divide the *virtual processors* into n sets and perform a reallocation of sets such that the first rank of each set requires communication in a distinct dimension, the same for the second and all following ranks. One such reallocation is obtained by m shuffle operations on *virtual processor* set m, with sets labeled consecutively from zero.

4.2.1 Pipelining successive FFT computations for virtual processors

For pipelining of computations we consider cyclic assignment of data elements to processors and a DIF FFT. Then, after the first $p - n$ ranks of butterfly computations there are 2^{p-n} independent FFTs of size 2^n to be computed. This fact was used in [4] for devising sorting algorithms on Boolean cubes. Each FFT has one element per processor, and each only requires one communication in each of the n cube dimensions. The computations of the 2^{p-n} FFTs can be pipelined, such that a total of $n + 2^{p-n} - 1$ communications are required with concurrent communication on all ports. The communication efficiency, measured as (the sum of the communication resources used over time)/((total number of available communication resources)*(time)), for the stages requiring communication is $\frac{2^{p-n}}{n+2^{p-n}-1}$, if $2^{p-n} > n$. The efficiency is approximately one for $2^{p-n} \gg n$. The total communication efficiency measured as (the sum of the communication resources used over time)/((available communication resources)*(total time)) is $\frac{2^{p-n}}{n+2^{p-n}-1+(p-n)2^{p-n}}$, which for $2^{p-n} \gg n$ is approximately equal to $\frac{1}{p-n+1}$. The reason for this low efficiency is simply that most of the computations are local to a processor.

4.2.2 Pipelining successive FFT computations with recursive reallocation of the data set

An alternative to this straightforward pipelined algorithm is obtained by recursively partitioning the set of 2^{p-n} FFTs, such that one half of the FFTs are computed in a half-sized Boolean cube, and the other half of the FFTs in the other half. The number of elements per processor of a given FFT doubles for every step. The communication time for this pipelined, recursive partitioning FFT is $2^{p-n-1} + n - 1$, which is approximately half of that of the straightforward pipelined algorithm. The recursive partitioning technique was used in [7,9] for *Balanced Cyclic Reduction* on Boolean cubes.

The recursive partitioning strategy for computing FFT for 2 *virtual processors*, $p - n = 1$, is illustrated in Table 2. Steps $p-n$ through p of the algorithm can be illustrated in terms of the address space as follows

Initial allocation: $(\underbrace{x_n}_{vp}\underbrace{x_{n-1}x_{n-2}\dots x_0}_{rp}) \qquad (p-1=n).$

Step 1: $(\underbrace{x_n}_{rp}\underbrace{x_{n-1}}_{vp}\underbrace{x_{n-2}\dots x_0}_{rp}).$

Step 2: $(\underbrace{x_nx_{n-1}}_{rp}\underbrace{x_{n-2}}_{vp}\underbrace{x_{n-3}\dots x_0}_{rp})$

$\vdots \qquad\qquad \vdots$

Step $p-1$: $(\underbrace{x_nx_{n-1}\dots x_1}_{rp}\underbrace{x_0}_{vp}).$

The dimension representing the virtual processors is successively moved to the lowest order bit position. In general, for $p-n > 1$ several exchange algorithms are possible. For instance, if the real processor dimension on which a butterfly operation is to take place is moved to the highest order virtual processor position, then the data permutation is

Proc. id	P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7
after	0	1	2	3	4	5	6	7
step $vp-1$	8	9	10	11	12	13	14	15
after	0	1	2	3	8	9	10	11
step vp	4	5	6	7	12	13	14	15
after	0	1	4	5	8	9	12	13
step $vp+1$	2	3	6	7	10	11	14	15
after	0	2	4	6	8	10	12	14
step $vp+2$	1	3	5	7	9	11	13	15

Table 2: The data distribution for the recursive partitioning, radix-2, FFT for two *virtual processors*.

equivalent to conversion from cyclic to consecutive storage [5]. In any step only one dimension is used, and successive steps can be pipelined. The numbers in Tables 2 and 3 are with respect to the input, not the output. The frequency components are in bit-reversed order with respect to the input ordering. Note, that the bit-reversal operation applies to the entire data set.

Initial allocation: $(\underbrace{x_{p-1}x_{p-2}\cdots x_n}_{vp}\underbrace{x_{n-1}\cdots x_0}_{rp})$.

Step $p-n$: $(\underbrace{x_{p-1}}_{rp}\underbrace{x_{p-2}x_{p-3}\cdots x_n x_{n-1}}_{vp}\underbrace{x_{n-2}\cdots x_0}_{rp})$.

Step $p-n+1$: $(\underbrace{x_{p-1}x_{p-2}}_{rp}\underbrace{x_{p-3}x_{p-4}\cdots x_{n-2}}_{vp}\underbrace{x_{n-3}\cdots x_0}_{rp})$.

$\vdots$ $\qquad$ $\vdots$

Step $p-1$: $(\underbrace{x_{p-1}x_{p-2}\cdots x_{p-n}}_{rp}\underbrace{x_{p-n-1}x_{p-n-2}\cdots x_0}_{vp})$.

4.2.3 High radix FFTs for many virtual processors

Computing the FFT as a sequence of radix-2^{p-n} FFTs requires a data reallocation of the form *all-to-all personalized communication* [10] in cubes of dimension $p-n$. There exist 2^{2n-p} distinct such cubes in a n-cube, assuming $2n-p>0$. The total number of reallocations that are required is $\lceil\frac{n}{p-n}\rceil$, each requiring a total of 2^{p-n-1} element transfer times, with concurrent communication on all $p-n$ ports.

The ideas of pipelining and high radix FFTs can be combined. After an FFT is performed on the local data set radix-Q_c FFTs involving communication are performed, and the different FFTs pipelined with respect to communication, assuming $Q_c < 2^n$.

The total number of element transfers in sequence for this algorithm is $(2^{p-n-q_c} + \lceil \frac{n}{q_c} \rceil - 1)2^{q_c-1}$, where $Q_c = 2^{q_c}$. This expression specializes to the pipelined, recursive partitioning radix-2 algorithm for $q_c = 1$, and the non-pipelined radix-2^{p-n} algorithm for $q_c = p - n$. The value of the expression is minimized for $q_c = 2$ if $q_c < n$. However, the improvement of such a radix-4 algorithm over a radix-2 algorithm is insignificant (one element transfer).

The recursive partitioning strategy can also be applied to higher radix FFTs. Table 3 shows the global steps for $p-n = 2$ and $n = 4$. The first step is an *all-to-all personalized communication* within each subcube of dimension 2 with respect to the 2 highest order real processor dimensions. For instance, processors 0, 4, 8 and 12 are in the same subcube. For the radix-4 algorithm, two bits are involved in every step. In general, for a radix-Q_c algorithm, q_c bits are involved. Successive steps involve consecutive blocks of q_c dimensions, and the steps can be pipelined. With the same strategy as in the radix-2 case for multiple virtual processors, the data permutation is again the same as in conversion from cyclic to consecutive assignment. Note, however, that it is only guaranteed that consecutive sets are in consecutive processors, but the order within a processor is not necessarily sequential. (It is sequential if the total number of elements is divisible by the number of *virtual processors*.)

$$\begin{array}{ll}
\text{Initial allocation:} & (\underbrace{x_{p-1}x_{p-2}\dots x_n}_{vp}\underbrace{x_{n-1}\dots x_0}_{rp}). \\
\text{Step } \frac{p-n}{q_c}: & (\underbrace{x_{p-1}x_{p-2}\dots x_{p-q_c}}_{rp}\underbrace{x_{p-q_c-1}x_{p-q_c-2}\dots x_n x_{n-1}\dots x_{n-q_c}}_{vp}\underbrace{x_{n-q_c-1}\dots x_0}_{rp}). \\
\text{Step } \frac{p-n}{q_c}+1: & (\underbrace{x_{p-1}x_{p-2}\dots x_{p-2q_c}}_{rp}\underbrace{x_{p-2q_c-1}x_{p-2q_c-2}\dots x_{n-2q_c}}_{vp}\underbrace{x_{n-2q_c-1}\dots x_0}_{rp}). \\
\vdots & \vdots \\
\text{Step } \frac{p}{q_c}-1: & (\underbrace{x_{p-1}x_{p-2}\dots x_{p-n}}_{rp}\underbrace{x_{p-n-1}x_{p-n-2}\dots x_0}_{vp}).
\end{array}$$

We conclude that with respect to communication efficiency a pipelined radix-2, or radix-4, algorithm shall be used, if $p - n < n$. Otherwise, a radix-Q_c algorithm with $Q_c \geq n$ shall be chosen. In such a case, the pipeline filling time of $n - 1$ is avoided. For a large number of *virtual processors* this time is only a small improvement.

4.2.4 Concurrent communication by reallocation through shuffle operations on sets of FFT data

A fourth alternative is to divide the *virtual processors* into n sets and perform a reallocation by cyclic rotation of the addresses m steps for set m with the sets numbered consecutively from 0. After such a reallocation the communication required for any

Proc. id	P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	P_{11}	P_{12}	P_{13}	P_{14}	P_{15}
after	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
step	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$\frac{vp}{2}-1$	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
after	0	1	2	3	16	17	18	19	32	33	34	35	48	49	50	51
step	4	5	6	7	20	21	22	23	36	37	38	39	52	53	54	55
$\frac{vp}{2}$	8	9	10	11	24	25	26	27	40	41	42	43	56	57	58	59
	12	13	14	15	28	29	30	31	44	45	46	47	60	61	62	63
after	0	4	8	12	16	20	24	28	32	36	40	44	48	52	56	60
step	1	5	9	13	17	21	25	29	33	37	41	45	49	53	57	61
$\frac{vp}{2}+1$	2	6	10	14	18	22	26	30	34	38	42	46	50	54	58	62
	3	7	11	15	19	23	27	31	35	39	43	47	51	55	59	63

Table 3: The data distribution for the recursive partitioning, radix-4 FFT for 16 processors and 4 virtual processors.

rank is in different dimensions for the different sets of *virtual processors*. The communication system is fully utilized. A final reallocation is required after the completion of the computations to realign the data. A one step rotation is a shuffle operation; $sh(u) = (u_{n-2}u_{n-3}\dots u_0u_{n-1})$, where $u = (u_{n-1}u_{n-2}\dots u_0)$. A rotation of j steps is defined by $sh^j = sh \circ sh^{j-1}$. The communication time required for the two sets of shuffle operations and the FFT computation is at least $2 \cdot 2^{p-n-1}\frac{n-1}{n} + 2^{p-n-1}$. This bound is obtained by dividing the local data set into n subsets, and noticing that the average communication distance is $\frac{n}{2}$. An FFT based on shuffle operations is clearly inferior to the pipelined, or high radix algorithms.

4.3 Allocation of Twiddle Factors

All twiddle factors $\omega_P^j, j \in [0, \frac{P}{2}-1]$ are used in the first rank of a radix-2 DIF FFT, or the first two ranks of a radix-4 DIF FFT, or three ranks of a radix-8 DIF FFT. As the computation proceeds from the input to the output, the number of distinct twiddle factors needed decreases. For the radix-2 DIF FFT the exponent of the twiddle factors needed for the butterfly on elements $x(j)$ and $x(j+\frac{P}{2})$, where $j = (j_{p-1}j_{p-2}\dots j_0)$, is $(j_{p-2}j_{p-3}\dots j_0)$. For the second rank the exponent of the twiddle factor ω_P is $(j_{p-3}j_{p-4}\dots j_0 0)$ for the pairs in locations j and $j+\frac{P}{4}$. In general, for an *in-place* DIF radix-2 FFT algorithm [13] the twiddle factor required for the computation of a butterfly on the elements in position j and $j+\frac{P}{2^{q+1}}$, i.e., for butterflies on bit $p-q-1$ is $\omega_P^{j_{p-2-q}j_{p-q-3}\dots j_0 00\dots 0}$, where the number of trailing 0's is q, $q \in [0, p-1]$ is the rank index. The exponent is simply the address of the lower storage location of the two involved in a butterfly computation shifted left as many steps as the index of the rank being computed.

If the DIF FFT is computed on a Boolean p-cube, then node $P-1$ requires $p-1$ distinct twiddle factors. If the FFT instead is computed on a n-cube with $n < p$, then there are $p-n$ ranks computed locally. The number of twiddle factors required for these ranks are $\sum_{m=1}^{p-n} 2^{p-n-m} = 2^{p-n} - 1$. After these first $p-n$ ranks are computed the interprocessor communication takes place. The twiddle factors needed are $\omega_P^{j_{p-2-q}j_{p-q-3}\dots j_0 00\dots 0}$, where the number of trailing 0's is q, $q \geq p-n$, and $p-2-q \leq n-2$. With a *cyclic* assignment of the data to processors, all twiddle factors needed for all elements in a processor are the same. Hence, for cyclic data assignment and a radix-2 DIF FFT computed on a Boolean n-cube, $n < p$, the maximum number of distinct twiddle factors needed in a processor is $2^{p-n} + n - 2$.

For a DIT radix-2 FFT, the exponent of the twiddle factors can also be computed from the addresses of the elements of an *in-place* algorithm. The twiddle factors for rank q are $\omega_P^{j_{p-q}j_{p-q+1}\dots j_{p-1}00\dots 0}$, where the number of trailing 0's is $n-1-q$, $q \in [1, p-1]$. Note, that the address is bit-reversed and shifted for the proper exponent. For the radix-2 DIT FFT the twiddle factors in computing rank 0 are all ω_P^0. In computing the DIT FFT on a Boolean n-cube it is convenient to use a *consecutive* allocation of data elements to processors. In such a case all elements assigned to a processor have the same n high order bits, and all butterfly computations for the elements assigned to a processor require the same twiddle factor for any rank requiring interprocessor communication.

5 Summary

We have presented FFT algorithms that make effective use of the hardware floating-point capability of the Connection Machine, and for which the use of the storage bandwidth and the communication system are of optimum order. For FFTs local to a processor, i.e., on *virtual processors*, a radix-R FFT should be used where R is determined by the number of available registers on the floating-point processor. For inter-chip FFTs a radix-4 pipelined algorithm yields minimum communication time with a very small margin over a radix-2 algorithm, if the number of dimensions required for virtual processors is smaller than the number required for real processors. Otherwise, a reallocation of the type *all-to-all personalized communication* between two sets of local FFTs offers the lowest communication time by a small margin over the radix-2 pipelined algorithm.

With a *cyclic* data assignment for a DIF FFT the first $p-n$ ranks of butterfly computations for an FFT on P elements on N processors are local and the last $\log_2 N$ ranks require interprocessor communication. All butterfly computations for a given rank and chip pair require the same twiddle factors. The exponent of the twiddle factor is obtained from the global address left shifted a number of steps equal to the rank.

For interprocessor communication only the processor address is needed. For DIT FFTs a *consecutive* data assignment yields similar properties, except that the first n steps require interprocessor communication, and the last $p-n$ steps are local. The exponent of the twiddle factors are computed from the global address bit-reversed, left shifted, but with a decreasing shift with the rank. For the first n steps the processor address suffices for the computation of the exponent, and all butterfly computations between a pair of processors for a given rank require the same twiddle factor. The total number of twiddle factors stored is $\frac{P}{2N} + nN$ for $\frac{P}{N}$ elements stored in each of N procesors. No communication of twiddle factors is needed. The expected execution time for an FFT on 16 million complex data elements is about 1.5 seconds.

The optimal number of processors is equal to the number of data elements, or the maximum available should the number of elements exceed the number of processors. The only exception is very small FFTs.

References

[1] David H. Bailey. A high-performance fast Fourier transform for the Cray-2. *The Journal of Supercomputing*, 1(1):43–60, 1987.

[2] Bengt Fornberg. A vector implementation of the fast Fourier transform. *Mathematics of Computation*, 36:189–191, 1981.

[3] Kung HT Hong J.W. I/O complexity: the red-blue pebble game. In *Proc. of the 13th ACM Symposium on the Theory of Computation*, pages 326–333, ACM, 1981.

[4] S. Lennart Johnsson. Combining parallel and sequential sorting on a Boolean n-cube. In *1984 International Conference on Parallel Processing*, pages 444–448, IEEE Computer Society, 1984. Presented at the 1984 Conf. on Vector and Parallel Processors in Computational Science II.

[5] S. Lennart Johnsson. Communication efficient basic linear algebra computations on hypercube architectures. *J. Parallel Distributed Comput.*, 4(2):133–172, April 1987. (Tech. Rep. YALEU/DCS/RR-361, Yale Univ., New Haven, CT, January 1985).

[6] S. Lennart Johnsson. The FFT and fast Poisson solvers on parallel architectures. In *Fast Fourier Transforms for Vector and Parallel Computers*, page , The Mathematical Sciences Institute, Cornell Univ., 1987. YALEU/DCS/RR-582, March 1987.

[7] S. Lennart Johnsson. Solving tridiagonal systems on ensemble architectures. *SIAM J. Sci. Statist. Comput.*, 8(3):354–392, May 1987. (Tech. Rep. YALEU/DCS/RR-436, Yale Univ., New Haven, CT, November 1985).

[8] S. Lennart Johnsson and Danny Cohen. *A Mathematical Approach to Computational Networks for the Discrete Fourier Transform.* Technical Report, USC/Information Sciences Institute, 1983. in preparation.

[9] S. Lennart Johnsson and Ching-Tien Ho. *Multiple tridiagonal systems, the alternating direction method, and Boolean cube configured multiprocessors.* Technical Report YALEU/DCS/RR-532, Dept. of Computer Science, Yale Univ., New Haven, CT, June 1987.

[10] S. Lennart Johnsson and Ching-Tien Ho. *Spanning graphs for optimum broadcasting and personalized communication in hypercubes.* Technical Report YALEU/DCS/RR-500, Dept. of Computer Science, Yale Univ., New Haven, CT, November 1986. To appear in IEEE Trans. Computers.

[11] S. Lennart Johnsson and Peggy Li. *Solutionset for AMA/CS 146.* Technical Report 5085:DF:83, California Institute of Technology, May 1983.

[12] S. Lennart Johnsson, Uri Weiser, Danny Cohen, and Al Davis. Towards a formal treatment of VLSI arrays. In *Proceedings of the Second Caltech Conference on VLSI*, pages 375 – 398, Caltech Computer Science Dept., January 1981.

[13] Alan V. Oppenheimer and Ronald W. Schafer. *Digital Signal Processing.* Prentice-Hall, Englewood Cliffs. NJ, 1975.

[14] W.P. Petersen. Vector fortran for numerical problems on CRAY-1. *Communications of the ACM*, 26(11):1008–1021, November 1983.

[15] L.R. Rabiner and B. Gold. *Theory and Application of Digital Signal Processing.* Prentice-Hall, Englewood Cliffs. NJ, 1975.

[16] Paul N. Swarztrauber. Multiprocessor FFTs. *Parallel Computing*, 5:197–210, 1987.

[17] Paul N. Swarztrauber. Vectorizing the FFTs. In G. Rodrique, editor, *Parallel Computations*, pages 490–501, Academic Press, 1982.

[18] Charles van Loan, editor. *Fast Fourier Transforms for Vector and Parallel Computers Workshop*, Mathematical Sciences Institute, Cornell Univ., March 1987.