Cooley-Tukey FFT on the Connection Machine

S. Lennart Johnsson, Robert L. Krawitz,
Roger Frye and Douglas MacDonald

YALEU/DCS/TR-750
November 1989

This work has been supported in part by the Air Force Office of Scientific Research grant number AFOSR-89-0382.

Cooley-Tukey FFT on the Connection Machine

S. Lennart Johnsson, Robert L. Krawitz, Roger Frye and Douglas MacDonald

Thinking Machines Corp.
245 First Street,
Cambridge, MA 02142

Abstract

We describe a radix-2 FFT implementation on the Connection Machine. The FFT implementation pipelines successive FFT stages to make full use of the communication capability of the network interconnecting processors, when there are multiple elements assigned to each processor. The number of twiddle factors stored in each processor is $\frac{P}{2N} + \log_2 N$ for an FFT on P complex points equally distributed among N processors. No communication of twiddle factors is required. The radix-2 FFT computations local to a processor has a peak performance of about 3 Gflops/s. The global FFT has a peak performance of about 1.7 Gflops/s.

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 describes the data mapping and control structures used in the implementation of a radix-2 FFT on the Connection Machine. Special attention is given to the effective use of the communication system, and to computation and storage of the twiddle factors.

The Connection Machine has up to 64k processors. 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. There are 16 processors on each processor chip, and two such chips share a floating-point unit. The processor chips are interconnected as a 12 dimensional Boolean cube. The floating-point units form an 11 dimensional cube, with two communication channels between each pair of processors. The FFT described below is developed for the Connection Machine with hardware floating-point processors.

The data allocation and scheduling of computations for the radix-2 FFT implementation on the Connection Machine pipelines the FFT stages as described in [19]. We first review the mapping of the radix-2 FFT to Boolean cube networks with particular emphasis on effective use of the communication system, and coefficient computation and storage [19]. Radix-4 and radix-8 FFT implementations on the Connection Machine are described in [9].

2 The Cooley-Tukey FFT

The Discrete Fourier Transform (DFT) is defined by

$$X(l) = \sum_{j=0}^{P-1} \omega_P^{lj} x(j), \quad \forall l \in [0, P-1], \quad \omega_P = e^{-\frac{2\pi i}{P}}.$$

The Cooley-Tukey Fast Fourier Transform [3] is obtained by a factoring of $P = P_0 P_1 \ldots P_{p-1}$ and using the properties of the "twiddle factors" ω_P. For instance, $\omega_P^{\frac{P}{2}} = -1$, $\omega_P^{k\frac{P}{4}} = (-i)^k$, and $\omega_P^{k\frac{P}{8}} = (\frac{(-1+i)}{\sqrt{2}})^k$. These properties are used in the derivation of radix-2, -4, and -8 FFTs. For the derivation of the radix-2 FFT we let $P = 2^p$ and $P_m = 2, \forall m \in [0, p-1]$. Then,

$$X(l) = \sum_{j=0}^{\frac{P}{2}-1} \omega_P^{lj} x(j) + \sum_{j=0}^{\frac{P}{2}-1} \omega_P^{l(j+\frac{P}{2})} x(j+\frac{P}{2}), \quad \forall l \in [0, P-1],$$

or with $l = 2l'$ for l even and $l = 2l'+1$ for l odd

$$X(2l') = \sum_{j=0}^{\frac{P}{2}-1} \omega_{\frac{P}{2}}^{l'j} (x(j) + x(j+\frac{P}{2})), \quad \forall l' \in [0, \frac{P}{2}-1],$$

$$X(2l'+1) = \sum_{j=0}^{\frac{P}{2}-1} \omega_{\frac{P}{2}}^{l'j} \omega_P^j (x(j) - x(j+\frac{P}{2})), \quad \forall l' \in [0, \frac{P}{2}-1].$$

The computation of a DFT on P data elements is replaced by the computation of $(x(j) + x(j+\frac{P}{2}))$ and $\omega_P^j (x(j) - x(j+\frac{P}{2}))$ for $j \in [0, \frac{P}{2}-1]$, plus the computation of two half-sized transforms. Repeating this procedure results in a *decimation-in-frequency* (DIF) FFT [21] after p steps. The number of complex multiplications is $\frac{1}{2}Pp$, and the number of complex additions is Pp, compared to $(P-1)^2$ complex multiplications and $P(P-1)$ complex additions for the DFT computed as a matrix-vector product. The computations in each step constitutes a "butterfly". All butterfly computations in the first step are performed on the highest order bit of the data index. The result of the FFT is obtained in bit-reversed order with the ordering implied in the derivation above, Figure 1.

An alternative way of deriving an FFT is to let $j = 2j'$ for j even and $j = 2j'+1$ for j odd. Then,

$$X(l) = \sum_{j'=0}^{\frac{P}{2}-1} \omega_{\frac{P}{2}}^{lj'} x(2j') + \sum_{j'=0}^{\frac{P}{2}-1} \omega_{\frac{P}{2}}^{lj'} \omega_P^l x(2j'+1), \quad \forall l \in [0, P-1]$$

or

$$X(l) = \sum_{j'=0}^{\frac{P}{2}-1} \omega_{\frac{P}{2}}^{lj'} x(2j') + \omega_P^l \sum_{j'=0}^{\frac{P}{2}} \omega_{\frac{P}{2}}^{lj'} x(2j'+1), \quad \forall l \in [0, \frac{P}{2} - 1]$$

$$X(l + \frac{P}{2}) = \sum_{j'=0}^{\frac{P}{2}-1} \omega_{\frac{P}{2}}^{lj'} x(2j') - \omega_P^l \sum_{j'=0}^{\frac{P}{2}} \omega_{\frac{P}{2}}^{lj'} x(2j'+1), \quad \forall l \in [0, \frac{P}{2} - 1]$$

Continuing this derivation leads to a *decimation-in-time* (DIT) FFT. The recursive derivation proceeds from the input towards the output for the decimation-in-frequency FFT, and from the output towards the input for the decimation-in-time FFT. In either case the butterfly computations proceed from the highest order bit towards the lowest order bit in the data index. However, the ordering made in the above derivation of the decimation-in-time FFT results in a requirement for a data input vector in bit-reversed order, but yields an output vector in normal order.

3 Boolean Cubes and Address Maps

The network interconnecting processor chips in the Connection Machine is a 12-dimensional Boolean cube. In a Boolean cube of $N = 2^n$ nodes, n bits are required for the encoding of the node addresses. 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 address field of the Connection Machine is divided into three parts: (off-chip|on-chip|memory). The off-chip field consists of 12 bits that encode the Connection Machine processor chips, the on-chip field encodes the 16 processors on each Connection Machine processor chip, and the lower order bits encode the memory addresses local to a processor. The lowest order off-chip bit encodes pairs of processor chips sharing a floating-point unit. On-chip communication is considerably faster than inter-chip communication. On-chip communication is a local memory reference. Off-chip communication is slower due to the limited bandwidth at the chip Boundary. The Non-Uniformity in access time impacts the optimum data allocation [12, 14].

3.1 Configuring the address space

The default data allocation scheme on the Connection Machine first determines how many data elements need to be stored in each processor for an equal number of elements

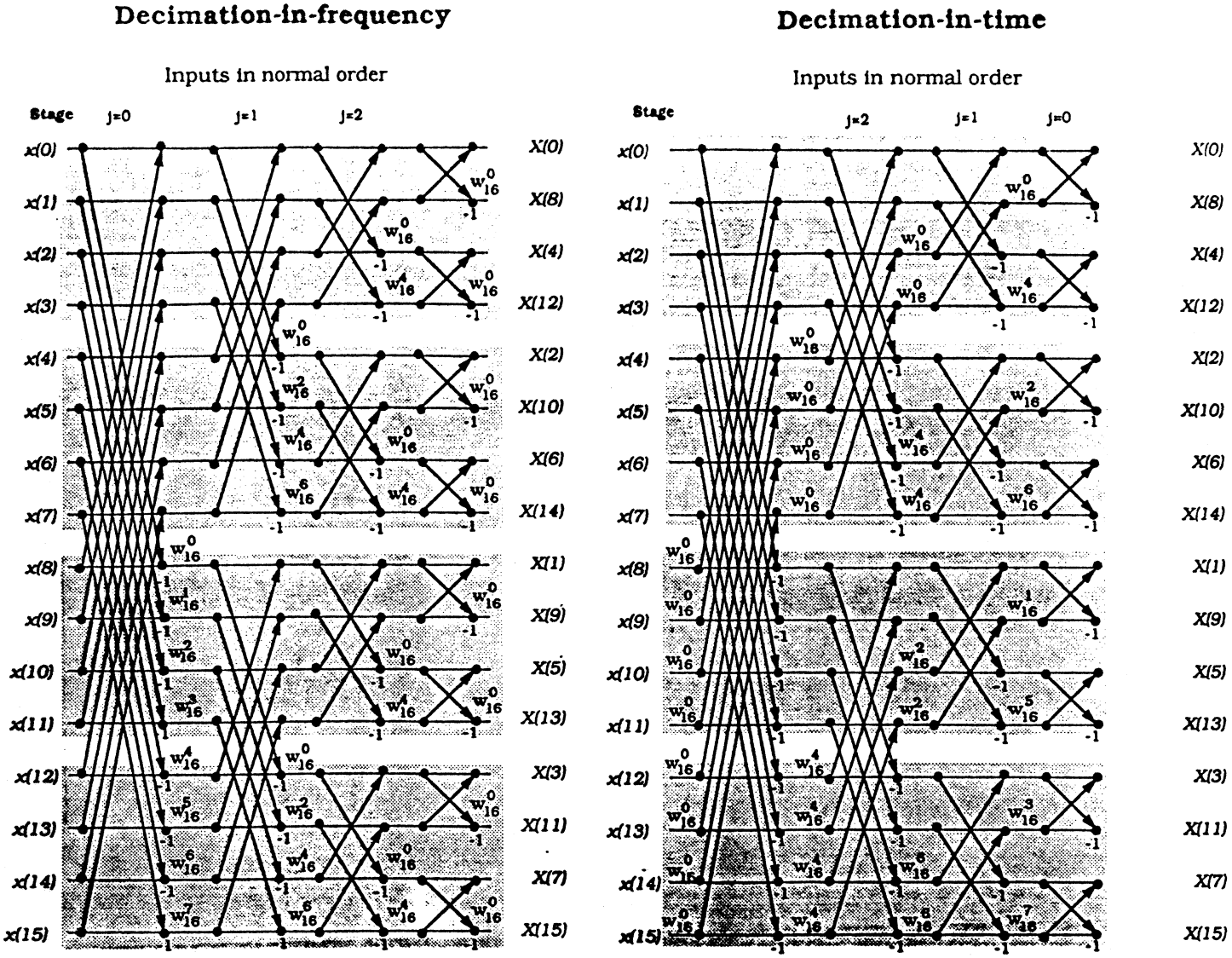

Figure 1: Decimation-in-Frequency and Decimation-in-Time FFT.

per processor, then stores that many successive elements in each processor, *consecutive storage* [12]. With the n highest order bits encoding the processors and the lower order bits encoding memory addresses in each processor, the consecutive assignment can be illustrated as follows.

$$\textit{Consecutive}\ \text{assignment:}(\underbrace{x_m x_{m-1} \ldots x_{m-n+1}}_{rp}\ \underbrace{x_{m-n} x_{m-n-1} \ldots x_0}_{vp})$$

The rp-field encodes *real processor* addresses and the vp-field encodes local *memory* addresses. For a data set of $M = 2^m$ complex points $m+1$ address bits are required, n of which are processor address bits with the data set distributed evenly across processors. There are $m - n + 1$ local storage address bits. In *cyclic* assignment the lowest order address bits determine the real processor address.

$$\textit{Cyclic}\ \text{assignment:}(\underbrace{x_m x_{m-1} \ldots x_n}_{vp}\ \underbrace{x_{n-1} x_{n-2} \ldots x_0}_{rp}).$$

All data elements in a processor have the same n low order bits. In the consecutive assignment the elements in a processor have the same n high order bits. The cyclic allocation scheme currently is not supported on the Connection Machine system. However, it offers a performance advantage in some situations, as described below.

The Connection Machine languages encode each axis of a multi-dimensional array separately. $\lceil log_2 P \rceil$ address bits are assigned to the encoding of the elements along an axis of length P thereby extending it to the nearest greater power of two, if P is not a power of two. The consecutive allocation scheme is used for each axis. The default encoding of the axes in the total address space attempts to configure each part of the address space (off-chip, on-chip, and memory) to conform with the array. To the extent possible, all axes have a segment of each address field, and the ratio of the lengths of segments for different axes is the same as that of the length of the axes.

In CM-Fortran compiler directives allow a user to change the default axes encoding by specifying an axis as *serial*, which implies that the axis is allocated to a single processor. *Lisp provides *in-processor arrays*. In PARIS (PARallel Instruction Set), the Connection Machine native language, a user has full control over what dimensions of the address space an axis occupies. But, only consecutive allocation of data to processors is supported.

If an array has fewer elements than the number of real processors in the configuration the array is extended such that there is one element per real processor. In CM-Fortran a new first axis is added to the array with a length equal to the number of instances of the specified array that matches the number of real processors. In *Lisp the axis length is extended.

Algorithm	Arithmetic Op's/ Element	Arithmetic Op's/ Elem. communic.
Matrix-vector mpy	2	$\sqrt{M}$
Matrix-matrix mpy	$\frac{2}{3}\sqrt{N}$	$\sqrt{M}$
Relaxation 5-point stencil	1.6/iter.	$\sqrt{M}$
Radix-2 FFT	$1.25 \log_2 N$	$1.25 \log_2(M/2)$

Table 1: The effect of a local storage of size M in computations on N elements.

3.2 Encoding of array axes

Since local memory references are significantly faster than inter-chip communication in most cases because of technological limitations, it is of interest to preserve locality of reference in the index space when mapped to the machine address space. Many algorithms for solving partial differential equations make extensive use of local references along the different axes of array data structures. But, the same data set may also be used for FFT computations, as for instance in the solution of Poisson's equation. The potential performance benefit from preserving locality is illustrated in Table 1 for some frequently occurring computations.

In the binary encoding successive integers may differ in an arbitrary number of bits. For instance, 63 and 64 differs in 6 bits, and hence are at a *Hamming* distance of 6 in the Boolean cube. A *Gray* code by definition has the property that successive integers differ in precisely one bit. The most frequently used Gray code for the embedding of arrays in Boolean cubes is a *binary-reflected* Gray code [12, 20, 22]. This Gray code is periodic. The code preserves adjacency for any loop (periodic one-dimensional lattice) of even length, and for loops of odd length one edge in the loop is mapped into a path of length two [12]. For the embedding of multi-dimensional arrays each axis may be encoded by the *binary-reflected* Gray code. The embedding of an $N_1 \times N_2 \times \ldots \times N_d$ array requires $\sum_{i=1}^{d} \lceil \log_2 N_i \rceil$ bits. The *expansion*, i.e., the ratio between the consumed address space, and the actual array size, is $2^{\sum_{i=1}^{d} \lceil \log_2 N_i \rceil} / \Pi_{i=1}^{d}$, which may be as high as $\sim 2^d$ [5, 6]. The expansion can be reduced by allowing some successive array indices to be encoded at a Hamming distance of two. The *dilation* is the maximum Hamming distance between any pair of adjacent array indices. Every two-dimensional array can be embedded with minimum expansion and dilation 2 [1]. Minimum expansion dilation 2 embeddings for a large class of two-dimensional arrays are given in [6], which also provides a technique for reducing the expansion of higher dimensional arrays. Minimal expansion dilation 7 embeddings are possible for all three dimensional arrays [2]. Embeddings with dilation 2 for many three dimensional arrays are given in [8].

In the programming systems on the Connection Machine data sets configured as arrays have their axes indices by default encoded in a binary-reflected Gray code for the off-chip

segment of the address field of each axes. In the other languages the Gray code encoding is invoked by configuring the Connection Machine as a lattice of the appropriate number of dimensions.

4 Radix-2 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 nodes can be uniquely encoded with a total of $p + \lceil \log_2(p+1) \rceil$ bits. With node addresses $(y_{p-1}y_{p-2}\ldots y_0|z_{t-1}z_{t-2}\ldots z_0)$, the butterfly network is defined by connecting node $(y|z)$ to nodes $(y \oplus 2^{p-1-z}|z+1)$ and $(y|z+1)$, $z \in [0, p-1]$, where $t = \lceil \log_2(p+1) \rceil$ and $\oplus$ denotes the bit-wise exclusive-or operation. For the computation of the radix-2 FFT the last t bits can 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 the communication is between ranks z and $z+1$. Complex multiplications are made in rank z for decimation-in-time FFT and rank $z+1$ for decimation-in-frequency FFT.

By identifying all nodes with the same y value and different z values 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. Only $\frac{1}{p}th$ of the total communications bandwidth of the p-cube is used. The full arithmetic power, instead of only half, can be used by splitting the butterfly computations between the pair of nodes storing the data (done on the CM-1). The parallel arithmetic complexity for computing an FFT on $P = 2^p$ complex elements on a Boolean n-cube is $5\lceil \frac{P}{N} \rceil \log_2 P$ real arithmetic operations with splitting of complex multiplications, ignoring lower order terms. The speed-up of the arithmetic time is $\min(P, N)$ The communication complexity is $3\lceil \frac{P}{N} \rceil \log_2 N$ element exchanges in sequence.

With P complex elements distributed evenly over $N = N < P$ processors there are $\frac{P}{N}$ elements per *real processor*. If the cyclic data assignment is used, then the first $p-n$ ranks of butterfly computations are local to a processor. The last n ranks require inter-processor communication. For consecutive assignment the first n steps require inter-processor 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 inter-processor communication and the local reference phases are reversed.

The embedding defined above is the binary encoding of array indices. Every index is directly identified by an address in the address space. For arrays embedded by a binary-reflected Gray code array elements that differ by a power of two greater than zero are at a distance of two, i.e., $Hamming(G(i), G(i+2^j)) = 2, j \neq 0$ [11]. Even though the elements to be used in a butterfly computation are at a Hamming distance of two it is still possible

to perform an FFT with $\min(p,n)$ nearest neighbor communications [17].

If there is only one element per processor, then every element is either involved in a computation or a communication. With multiple elements per processor the communication efficiency can be increased from $\frac{1}{\min(p,n)}$ to $\frac{\min(p,n)}{n}$ [19], which for $p > n$ is one.

4.2 Maximizing the communication efficiency

We first consider one-dimensional arrays encoded in binary code. For each of the n inter-processor communication stages increased utilization of the communication system can be achieved by

- pipelining the computations for successive butterfly stages
- radix-2^r FFTs locally and data reallocation in r-dimensional subcubes by *all-to-all broadcasting* [18] between successive radix-2^r computations.
- using n address maps. The set of inter-processor dimensions used for different address maps use every inter-processor dimension for every butterfly stage.

The radix-2 FFT implemented on the Connection Machine makes use of pipelining. Two different pipelined algorithms are presented below. They differ by a factor of two in arithmetic load balance and communication efficiency. High radix FFTs are discussed in [9].

4.2.1 Pipelining successive butterfly stages

With N processors performing $\frac{N}{2}$ butterfly computations concurrently, $\frac{P}{N}$ butterfly computations must be performed sequentially in each stage. In the first n butterfly stages the lowest order $p-n$ bits are identical for all data elements that interact in any butterfly computation. The first n butterfly stages can be viewed as consisting of $\frac{P}{N}$ independent FFTs, each of size N and identified by the lowest order $p-n$ bits [10]. In the consecutive data allocation scheme each of these FFTs has one complex data element per processor. Every FFT performs communication in processor dimensions $n-1, n-2, \ldots, 0$. By pipelining the communications for the different FFT computations all communication channels required for the FFT are used if $n < \frac{P}{N}$, except for the pipeline start-up and shut-down phases. Figure 2 illustrates the independent FFTs during the inter-processor communication phase. After the n butterfly stages with inter-processor communication, the remaining $p-n$ stages are entirely local. The high order n bits identify N different FFTs of size $\frac{P}{N}$ each.

The number of complex data element transfers in sequence for the pipelined FFT is $n + \frac{P}{N} - 1$. The communication efficiency, measured as (the sum of the communication

Decimation-in-time

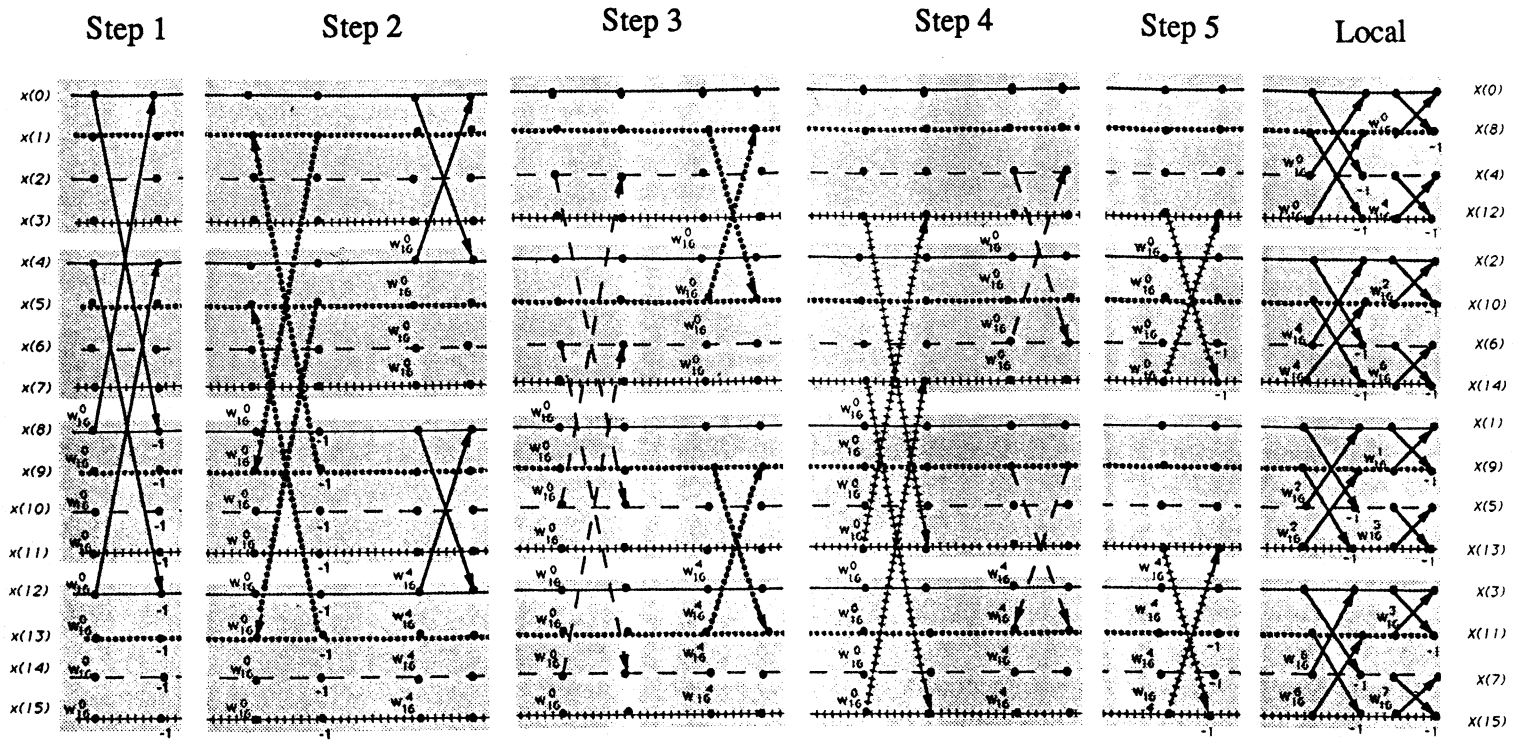

Figure 2: Partitioning of a one-dimensional FFT with multiple elements per processor

resources used over time)/((total number of available communication resources)*(time)), for the stages requiring communication is $\frac{\frac{P}{N}}{n+\frac{P}{N}-1}$, $p \geq n$. The efficiency is approximately one for $\frac{P}{N} \gg n$.

Pipelining can also be applied in the cyclic allocation scheme. The first $p-n$ stages consist of N independent FFTs of size $\frac{P}{N}$ each. The latter stages consist of $\frac{P}{N}$ independent FFTs, each of size N with one complex data element per processor.

4.2.2 Improved communication efficiency with full arithmetic load balance.

With a single exchange of complex data points the complex addition/subtraction can be performed concurrently in the two processors exchanging data, but sharing the complex multiplication requires additional communication. By recursively partitioning the set of $\frac{P}{N}$ FFTs during the n inter-processor communication stages, 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 perfect arithmetic load balance is achieved. The number of complex element transfers in sequence for this pipelined, recursive partitioning FFT is $\frac{P}{2N}+n-1$ [19], which is approximately half of the number of element transfers of the straightforward pipelined algorithm. The recursive partitioning doubles the number of elements per processor of a given FFT in every step, and reduces the number of FFTs serviced by a processor. The recursive partitioning technique was also used in [13, 15] for *Balanced Cyclic Reduction* on Boolean cube networks.

The recursive partitioning strategy for computing a radix-2 FFT with two complex data elements per real processor, $p-n=1$, and *cyclic* allocation of elements to processors is illustrated in Table 2 and below

$$
\begin{array}{ll}
\text{Initial allocation:} & (\underbrace{x_n}_{vp}\,\underbrace{x_{n-1}x_{n-2}\ldots x_0}_{rp}) \qquad (p-1=n). \\
\text{Step 1:} & (\underbrace{\overline{x_{n-1}}}_{vp}\,\underbrace{\overline{x_n}x_{n-2}\ldots x_0}_{rp}). \\
\text{Step 2:} & (\underbrace{\overline{x_{n-2}}}_{vp}\,\underbrace{x_n\overline{x_{n-1}}x_{n-3}\ldots x_0}_{rp}) \\
\vdots & \vdots \\
\text{Step } n\text{:} & (\underbrace{\overline{x_0}}_{vp}\,\underbrace{x_{n-1}\ldots\overline{x_1}}_{rp}).
\end{array}
$$

The bits of the address space on which the exchange in each step takes place are marked by $\overline{x_i}$. Since one of the dimensions is a local memory address half of the local data is exchanged between adjacent processors. The dimension representing the virtual processors is successively moved to the lowest order bit position. The exchange sequence in the illustration converts the cyclic allocation to consecutive allocation. It is also an

Proc. id	P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7
initial	0	1	2	3	4	5	6	7
alloc.	8	9	10	11	12	13	14	15
after	0	1	2	3	8	9	10	11
1st exch.	4	5	6	7	12	13	14	15
after	0	1	4	5	8	9	12	13
2nd exch.	2	3	6	7	10	11	14	15
after	0	2	4	6	8	10	12	14
3rd exch.	1	3	5	7	9	11	13	15

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

unshuffle. For $p - n > 1$ there exist many exchange algorithms that yield the correct partitioning, but the final permutation differs. A permutation that yields a consecutive data allocation (after possibly a local permutation) is obtained, if real processor bit k is exchanged with memory address bit $k \bmod p - n$ [12]. In any butterfly step only one dimension is used, and successive steps can be pipelined. The numbers in Table 2 denote the initial data indices. The transformed data items are obtained in bit-reversed order with respect to this ordering.

The recursive partitioning algorithm for more than two elements per processor can be illustrated as follows for $p - n > n$. A local unshuffle of order $p - 2n - 1$, or shuffle of order n establishes a consecutive data allocation. If $p - n < n$, then the local memory address bits are traversed cyclically in the exchange sequence.

$$\begin{aligned}
&\text{Initial allocation:} && (\underbrace{x_{p-1}x_{p-2}\ldots x_n}_{vp}\underbrace{x_{n-1}\ldots x_0}_{rp}).\\
&\text{1st exch.} && (\underbrace{x_{n-1}x_{p-2}x_{p-3}\ldots x_n}_{vp}\underbrace{x_{p-1}x_{n-2}\ldots x_0}_{rp}).\\
&\text{2nd exch.} && (\underbrace{x_{n-1}x_{n-2}x_{p-3}x_{p-4}\ldots x_n}_{vp}\underbrace{x_{p-1}x_{p-2}x_{n-3}\ldots x_0}_{rp}).\\
&\vdots && \vdots\\
&n\text{th exch.} && (\underbrace{x_{n-1}x_{n-2}\ldots x_0x_{p-n-1}\ldots x_n}_{vp}\underbrace{x_{p-1}x_{p-2}\ldots x_{p-n}}_{rp}).
\end{aligned}$$

If the initial data ordering is consecutive, then the recursive partitioning of the data set moves lower order, memory address bits into the real processor address field such that p inter-processor exchange steps are required (instead of n such steps and $p - n$ local steps).

4.2.3 Concurrent communication by reallocation through shuffle operations

By dividing the local data set into n sets and performing a reallocation by cyclic rotation of the addresses m steps for set m (with the sets numbered consecutively from 0) the communication required for any rank is in different dimensions for the different sets. The communication system is fully utilized for every butterfly step. A final reallocation is required after the completion of the computations to realign the data. A one step address 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 $2 \cdot \frac{P}{2N}\frac{n-1}{n} + \frac{P}{2N}$. 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 algorithm.

4.3 FFT on multi-dimensional arrays

Performing an FFT along a single axis of a multi-dimensional array implies a number of independent one-dimensional FFTs. The number of FFTs is determined by the product of the length of the axes on which the FFT is not performed.

Multi-dimensional FFTs can be performed as a number of independent one-dimensional FFTs along one axis followed by a number of independent one-dimensional FFTs along the other axes in succession. Pipelining can be performed over all inter-processor dimensions being part of any FFT as long as the inter-processor dimensions are not mixed with local FFT computations. Any axis with local memory address bits breaks the communication pipeline, if it requires an FFT.

4.4 Twiddle Factors

The total number of twiddle factors needed for a radix-2 FFT of size $P = 2^p$ is $\frac{P}{2} - 1$. For the computation of an FFT on a distributed memory machine it is important to minimize the need either for redundant storage of twiddle factors, or for communication of twiddle factors when required in a processor different from the one in which they are stored.

4.4.1 Decimation-in-frequency

All twiddle factors $\omega_P^j = e^{-\frac{2\pi i}{P}j}, j \in [0, \frac{P}{2} - 1]$ are used in the first rank of a radix-2 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 $x(j + \frac{P}{2})$ is $(j_{p-1}) \times (j_{p-2}j_{p-3}\dots j_0)$, where $j = (j_{p-1}j_{p-2}\dots j_0)$. For the second rank the exponent of the twiddle factor

ω_P is $(j_{p-2}) \times (j_{p-3}j_{p-4}\dots j_0 0)$ for the pairs in locations j and $j + \frac{P}{4}$. In general, for an *in-place* DIF radix-2 FFT [21] 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., butterfly stage $q \in [0, p-1]$ is $\omega_P^{(j_{p-q-1})\times(j_{p-q-2}j_{p-q-3}\dots j_0)2^q}$. The first butterfly stage is stage zero. The exponent of the twiddle factor is simply the common address below bit $p-q-1$ of the pair of complex elements in a butterfly computation, shifted left q steps with an end-off shift.

If the FFT of size $P = 2^p$ is computed on a Boolean p-cube, then node $P-1$ requires $p-1$ distinct twiddle factors. If the FFT is computed on an n-cube, $n < p$, and the allocation of data to processors is *cyclic*, then the set of twiddle factor indices required for the stages local to a processor is $(\{j_{p-2}j_{p-3}\dots j_n\}|j_{n-1}\dots j_0)$, $(\{j_{p-3}j_{p-4}\dots j_n\}|j_{n-1}\dots j_0)2$, ..., $(j_{n-1}\dots j_0)2^{p-n-1}$, or a maximum total of $\sum_{m=1}^{p-n} 2^{p-n-m} = \frac{P}{N} - 1$ for any processor. The notation $\{j_{p-2}j_{p-3}\dots j_n\}$ denotes the set of all values that can be assumed by the number of bits within the braces. After the first $p-n$ stages the remaining n stages consist of $\frac{P}{N}$ independent FFTs of size N, each with one element per processor. All $\frac{P}{N}$ FFTs have the same twiddle factor for a given butterfly stage. A total of $n-1$ twiddle factors are needed for the inter-processor communication stages, one for each butterfly stage, except the last. Hence, for cyclic data allocation and a radix-2 DIF FFT of size 2^p computed on N processors, $n < p$, the maximum number of distinct twiddle factors needed in a processor is $\frac{P}{N} + n - 2$. Allocating twiddle factor storage uniformly across all processors yield a total twiddle factor storage of $P + (n-2)N$, which for $P \gg N$ is about twice the storage required on a sequential computer. For $P = N$ uniform twiddle factor storage across processors yields a total storage of $(n-1)N$, which exceeds the sequential storage by a factor of approximately $2(n-1)$.

With a *consecutive* data allocation $\frac{P}{N}$ twiddle factors are needed in at least one processor for every stage of the first $n-1$ stages. The sets of twiddle factors for different stages are disjoint. For instance, consider the processor with address $(j_{p-1}j_{p-2}\dots j_{p-n}), j_k = 0, k \in \{p-n+1, p-n+2, \dots p-1\}$ and $j_n = 1$. This processor contains the data indices $(00\dots01|\{j_{p-n-1}j_{n-2}\dots j_0\})$. Shifting this set of addresses left by one step yields a completely new set of addresses, since the leading one defines a new range disjoint from the previous range. This observation is true for every inter-processor butterfly stage. The twiddle factor index for the remaining stages form subsets of the set of twiddle factors for the last inter-processor communication stage.

4.4.2 Decimation-in-time

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 stage q are $\omega_P^{(j_{p-q-1})(j_{p-q}j_{p-q+1}\dots j_{p-1})2^{p-1-q}}$, $q \in [0, p-1]$. Note, that the address is bit-reversed and shifted for the proper exponent. For the radix-2 DIT FFT the twiddle factors for stage 0 are all ω_P^0. With a *consecutive* data allocation to processors the processor address bits form the high order bits of the element index. The first stage does not require any

twiddle factors. The following $n-1$ stages require one twiddle factor per stage. All $\frac{P}{N}$ different FFTs of size N require the same twiddle factors, since the local addresses do not enter into the index computation. The last $p-n$ stages are local, and the maximum total number of twiddle factors required per processor is $\frac{P}{N}-1$, as in the case of *cyclic* allocation and decimation-in-frequency FFT.

With *cyclic* allocation the local addresses enter into the twiddle factor index computation immediately. The need for twiddle factors is the same as in the consecutive allocation of data and computing the FFT by a decimation-in-frequency algorithm.

4.4.3 Bit-reversed input

With the input in bit-reversed order the traversal of the bits in the address field is from the lowest order to the highest order bit. With the data indices being bit-reversed with respect to the addresses the decimation-in-frequency FFT requires addresses in bit-reversed order instead of normal order for the twiddle index computation. Similarly, the decimation-in-time FFT requires addresses in normal order instead of in bit-reversed order for normal order inputs. With these differences the consecutive ordering yields the smallest requirements for twiddle factor storage for the decimation-in-frequency FFT, and cyclic storage for the decimation-in-time FFT. The preferred combinations of data allocation and FFT type are opposite to those preferred with normal order input.

4.4.4 Inverse FFT

The Inverse Discrete Fourier Transform (IDFT) is defined by

$$\tilde{x}(j) = \sum_{l=0}^{P-1} \omega_P^{-lj} X(l), \quad \forall j \in [0, P-1], \quad \omega_P = e^{-\frac{2\pi i}{P}}.$$

It is easy to show that $\tilde{x}(j) = Px(j)$. For the computation of the IDFT we notice that $\omega_P^{-lj} = \omega_P^{(P-l)j}$. Hence, the IDFT can be computed by either using $P-l$ as the index of the twiddle factors used for the DFT, or by using the conjugates of those twiddle factors. The scaling can either be made by $\sqrt{P}$ during both the DFT and the IDFT, or by P during either the DFT, or the IDFT. With exception of the twiddle factor index the computations are identical.

4.4.5 Multi-dimensional FFT

In general, each axis has its own set of twiddle factors. The twiddle factors are a function of the axis length. The twiddle factor for an axis is a subset of the twiddles for the longest axis. With axes of length $P_1 \times P_2 \times \ldots P_k$ the minimum number of twiddle factors

is $\frac{\max_\ell(P_\ell)}{2}$. With separate storage of the twiddle factors for each axis the total storage is $\frac{\sum_\ell P_\ell}{2}$, which is less than the required storage for a one-dimensional FFT of size $\Pi_\ell P_\ell$.

4.4.6 Reduced twiddle factor storage

For the consecutive data allocation, normal order input, and decimation-in-time radix-2 FFT, the set of twiddle factor indices in the last stage is $\{j_1 j_2 \dots j_{n-1}\}|j_n \dots j_{p-1}$. The highest order bit j_1 corresponds to bit position $p-2$. Hence, $\{1 j_2 \dots j_{n-1}\}|j_n \dots j_{p-1} = \frac{P}{4} + \{0 j_2 \dots j_{n-1}\}|j_n \dots j_{p-1}$. But, $\omega_P^{j+\frac{P}{4}} = \omega_P^j \cdot e^{-\frac{2\pi i}{P}\frac{P}{4}} = -i \cdot \omega_P^i$. Half of the twiddle factors can be obtained from the other half without any arithmetic. This property is true for all on-processor stages. The same property is true for

- decimation-in-frequency FFT, cyclic data allocation, and normal input order,
- decimation-in-time FFT, cyclic data allocation, and bit-reversed input order,
- decimation-in-frequency FFT, consecutive data allocation, and bit-reversed input order.

The observation can be generalized to bits $p-3, p-4, \dots$ in the twiddle factor index, but complex arithmetic is required for all of them. Bit $p-3$ is associated with a 45-degree rotation, i.e., $-\frac{1+i}{\sqrt{2}}$.

4.5 Summary of algorithmic and data layout issues

A butterfly network can be effectively emulated on a Boolean cube network by mapping the butterfly stages into time. The butterfly interconnections correspond to interconnections in the Boolean cube. With multiple elements per processor the communication efficiency can be improved from $\frac{1}{n}$ for an n-dimensional cube to $\frac{\frac{P}{N}}{\frac{P}{N}+n-1}$ by pipelining.

For normal order input and cyclic data allocation a recursive partitioning of the data set yields a reduction in the number of element transfers by a factor of two compared to the straightforward pipelined algorithm, and a perfect load balance. The optimum partitioning is an N-way partitioning. Second to that is 4-way partitioning, and with an insignificant additional expense 2-way partitioning. If the data allocation is consecutive, then the recursive partitioning requires additional inter-processor communication. For bit-reversed input order the data allocation shall be consecutive for optimum effectiveness of the recursive partitioning technique.

The preferred combinations of data allocation, input order, and FFT type with respect to twiddle factor storage for the pipelined algorithm are:

FFT	Data alloc.	Twiddle index stage q	Max. number of twiddles per proc.
DIT	consec.	$\{j_{p-q}j_{p-q+1}\cdots j_{p-n-1}\}\vert j_{p-n}\cdots j_{p-1}2^{p-1-q}$	$\frac{P}{N}+n-2$
	cyclic	$j_{p-q}j_{p-q+1}\cdots j_{n-1}\vert\{j_n\cdots j_{p-1}\}2^{p-1-q}$	$(n-1)\frac{P}{N}$
DIF	consec.	$j_{p-q-2}j_{p-q-3}\cdots j_{p-n}\vert\{j_{p-n-1}\cdots j_0\}2^q$	$(n-1)\frac{P}{N}$
	cyclic	$\{j_{p-q-2}j_{p-q-3}\cdots j_n\}\vert j_{n-1}\cdots j_0 2^q$	$\frac{P}{N}+n-2$

Table 3: Radix-2 twiddle factor storage, normal input order.

FFT	Data alloc.	Twiddle index stage q	Max. number of twiddles per proc.
DIT	consec.	$j_{q-1}j_{q-2}\cdots j_{p-n}\vert\{j_{p-n-1}\cdots j_0\}2^{p-1-q}$	$(n-1)\frac{P}{N}$
	cyclic	$\{j_{q-1}j_{q-2}\cdots j_n\}\vert j_{n-1}\cdots j_0 2^{p-1-q}$	$\frac{P}{N}+n-2$
DIF	consec.	$j_{q+1}j_{q+2}\cdots j_{n-1}\vert\{j_n\cdots j_{p-1}\}2^q$	$\frac{P}{N}+n-2$
	cyclic	$\{j_{q+1}j_{q+2}\cdots j_{p-n-1}\}\vert j_{p-n}\cdots j_{p-1}2^q$	$(n-1)\frac{P}{N}$

Table 4: Radix-2 twiddle factor storage, bit-reversed input order.

- normal input order, consecutive data allocation, decimation-in-time FFT
- normal input order, cyclic data allocation, decimation-in-frequency FFT
- bit-reversed input order, consecutive data allocation, decimation-in-frequency FFT
- bit-reversed input order, cyclic data allocation, decimation-in-time FFT

The storage requirements and the formula for the twiddle factor index computations are summarized in Tables 3 and 4. The storage requirements for on-processor twiddles can be reduced by a factor of two, by computing half of the twiddles by performing 90-degree rotations "on-the-fly".

With the exception of the twiddle factors there is no essential difference between a multi-dimensional FFT and a one-dimensional FFT. Pipelining can be extended across axes of the array on which the FFT is performed, as long as there are no on-processor dimensions inter-mixed with inter-processor dimensions.

5 Implementation

The FFT implementation for which performance measurements are given below is a radix-2 FFT. Radix-4 and radix-8 FFT on the Connection Machine are described in [9]. The standard scheme for allocation of multiple elements to processors on the Connection Machine is the consecutive scheme. A decimation-in-time radix-2 FFT is used for data in normal input order, and a decimation-in-frequency radix-2 FFT for bit-reversed input order. The twiddle factor storage is $\frac{P}{2N} + \log_2 N - 2$ in each of N processors for an FFT of size P with the data uniformly distributed across the processors. The inverse Discrete Fourier Transform is computed by a FFT using the conjugated twiddle factors. The inter-processor communication stages are pipelined. For multi-dimensional FFTs each axis is treated independently. No sharing of twiddle factors between axes takes place.

The FFT routine is a complex-to-complex FFT. The data is assumed to be mapped into the address space by a binary encoding. For arrays mapped to the address space by a binary-reflected Gray code a remapping to binary encoding is made prior to the computation of the FFT, or the inverse FFT. The remapping is currently not optimized. It is performed by a call to the Connection Machine router. For normal order input the output is in bit-reversed order. If the output is desired in normal order a reordering is made after the FFT is computed. An optimized bit-reversal routine based on the algorithms in [4, 23, 7, 16] is under development.

For the description of the implementation and the explanation of some of the performance data it is necessary to present one more architectural feature of the Connection Machine. The software systems allocate data serially to the processors. But, 32 processors share a floating-point unit that can access the memories of the 32 processors in bit-slices. Transposing the data of 32 processors from *field-wise* to *slice-wise* allocates each word across the memories of groups of 32 processors sharing a floating-point unit. The FFT is developed with this view of the Connection Machine. In the following a processor refers to a floating-point processor.

The change from field-wise to slice-wise storage interchanges the lowest order off-chip bit and the processor bits (5 bits) with the memory address field. A 5-shuffle is performed. For a one-dimensional array the memory stride for bit k, with the lowest order bit being bit zero, is $2^{(k+5) \bmod (1+chip+memory\ bits)}$. The stride for successive array elements is 32. The stride is increasing for elements at increasing distance up to a point. The stride for the fifth highest order bit (the lowest order real processor bit)is one. In the case of multidimensional arrays the stride of the different axes becomes fairly complex. Therefore, a memory reordering is performed such that the stride of the first array axis is one, the stride for the second equal to the length of the first axis, etc.

The conversion from field-wise to slice-wise storage is performed after a potential remapping to binary encoding, and before the local memory reordering. If the array is encoded by a binary-reflected Gray code, then the transposition from field-wise storage

to slice-wise storage also needs to convert the lowest order off-chip bit, which encodes the chip pairs sharing storage, from Gray code to binary code. Algorithms for the conversion are described in [12, 14].

Any decimation-in-frequency FFT has unique twiddle factors for every butterfly computation in the first step. The second step of a radix-2 FFT consists of two half size FFTs, and the set of twiddle factors are used twice. All butterfly computations have the same coefficients in the last step. The local loops are ordered such that the total number of twiddle factor loads is $\frac{P}{N} - 1$. The same property is true for decimation-in-time FFT. The local FFT is computed one stage at a time, and for each stage a stride is determined for butterfly computations requiring the same twiddle factors, as well as for butterfly computations requiring successive twiddle factors. In the event of a multi-dimensional array, a number of local FFTs are performed, with the number being determined by the product of the length of the axes currently not subject to an FFT. The local FFT kernels for decimation-in-time and decimation-in-frequency FFT are almost identical, with the exception that the pipelines in the floating-point unit are organized slightly differently, resulting in a performance difference of up to $\sim 3\%$.

For the FFT stages requiring inter-processor communication the sign change is integrated with the communication such that one processor involved in the exchange performs a complex addition while the other performs a complex subtraction. Only one of the processors performs a useful complex multiplication. Data exchange in different inter-processor dimensions is pipelined. For each exchange of a pair of complex elements across a set of inter-processor dimensions butterfly computations are made on local data and exchanged data. The data exchange - butterfly computation is repeated until all data elements along every instance of the axis subject to an FFT computation is treated. The communication pipeline is active for all array elements in memory.

For ease of implementation the current butterfly computations for the inter-processor communication phase are organized into groups of size four, one for each of four inter-processor dimensions. This detail explains some of the performance data presented below. If the number of non-local dimensions is an exact multiple of four, then the data is moved into and out of the floating point unit from and to the memory allocated to the array. However, if the number of dimensions is not an exact multiple of four, a temporary storage area is used. The size of this storage area corresponds to four butterfly computations. Data for each inter-processor dimension subject to a butterfly computation is moved to the temporary storage area. Then four butterfly computations are performed on the data in the entire temporary storage area, and the result returned to the storage area. The desired results are then returned to the appropriate memory locations. Hence, the time for butterfly computations is the smallest when the use of temporary storage is avoided, and increases with the number of inter-processor dimensions in the range one to three.

In the current implementation the decimation-in-frequency butterfly computation is performed as a single floating-point pipeline, but the decimation-in-time butterfly operation consists of two pipelines, resulting in a performance difference of up to $\sim 15\%$.

5.1 Twiddle factor computation

For the stages requiring communication between different floating-point units, the twiddle factors depend only on the stage and the unit. The twiddle factor index can be computed by

- extracting the $p-1$ highest order bits of the data element index, i.e., bits 1 through $p-1$ into a word t with bit locations 0 to $p-2$.
- bit-reverse the extracted word t.
- perform $p-q-1$ steps of end-off left shifts of t with bits $p-q-2$ to 0 set to zero, $q = \{0, 1, \ldots, n-1\}$.

Each value of q corresponds to a different butterfly stage. For the first stage $q = 0$. The computation is performed by all floating-point units concurrently. The computations are completely uniform.

The computations in the local stages of the FFT and the table of twiddle factors are organized such that, regardless of stage, successive butterfly computations in the same reduced size FFT (there are $\frac{N}{2}$ FFTs of size two in the first stage of a DIT FFT, and one FFT of size N in the last stage) accesses the twiddle factors within the table with a stride of one complex number. The twiddle factors for the first local stage of a decimation-in-frequency radix-2 FFT forms the first block in a table. The twiddles for the second local stage form a second block in the table, etc. For the decimation-in-frequency FFT the blocks of twiddle factors are accessed in order. For the decimation-in-time FFT the blocks are accessed in reverse order. Within a block the twiddle factors are stored in bit-reversed order.

All floating-point units concurrently compute the same number of twiddle factors, but the indices differ. The twiddle factors are computed in-place, and accounts for the shuffle operation taking place in conversion from the field-wise to the slice-wise representation. The twiddle factor indices for the local stages are computed in the field-wise representation, assuming a cyclic storage scheme on each pair of processor chips sharing a floating-point unit. After transposition to slice-wise storage the order is consecutive. Moreover, the twiddle factors for the largest block is computed by the first set of $\frac{P}{2N}$ processors in this ordering, the second largest block (the second stage in a decimation-in-frequency FFT) by the next $\frac{P}{4N}$ processors, etc. The field-wise to slice-wise transposition can be represented as:

$$(\underbrace{xxxxxy}_{off-chip}|\ \underbrace{yyyy}_{on-chip}\ |\underbrace{zzzzzz}_{memory}) \rightarrow (\underbrace{xxxxxz}_{off-chip}|\underbrace{zzzzzyyyyy}_{memory})$$

The twiddle factor computation that precedes the transposition is as follows:

1. Form a number with the local memory address appended to the processor address with the memory address forming the high order part, i.e., form $(zzzzzzyyyyy)$.

2. Let q be the number of leading ones in $(zzzzzzyyyyy)$. q is the local stage number for a decimation-in-frequency FFT.

3. Set the leading ones to zero. $(\underbrace{00\ldots0}_{q}zzyyyyy)$.

4. Bit-reverse $(\underbrace{00\ldots0}_{q}zzyyyyy)$ to $(yyyyyzz\underbrace{00\ldots0}_{q})$.

5. Append n low order bits with value zero. $(yyyyyzz\underbrace{00\ldots0}_{q+n})$.

6. Bit-reverse the floating-point unit address, and shift it left q steps, and perform a *logical-or* operation with $(yyyyyzz\underbrace{00\ldots0}_{q+n})$.

All the above steps can be performed concurrently. The only processor dependent operation occurs in steps two and three. The result is the twiddle exponents as described in section 4.4, in order of stage number, and for each stage number in bit-reversed order.

The maximum number of complex data elements per floating-point unit with the reduced twiddle factor storage scheme is 16384 for a one-dimensional FFT, the same as for two- or more dimensions, for 64k words of local memory (512 Mbytes of total memory).

5.2 Performance measurements

The performance measurements have been made on Connection Machine configurations with 32-bit floating-point processors. A fully configured Connection Machine has 2048 such processors. The performance of the local kernel for different sizes is given in Table 5 and Figures 3 and 4. All reported timings and Mflop rates include the time for conversion from field-wise storage to slice-wise storage. This time amounts to about 15% of the total time. The times for a potential reordering from Gray code to binary code, or from bit-reversed to normal order if needed, are not included. With the organization of the current implementation the time for the local part is of the following form $c_0 + (\text{number of FFT of size} N)(c_1 + c_2 \log_2 N + c_3 N + c_4 N \log_2 N)$, where c_1 models the initialization for each independent FFT as defined by the input array, c_2 models the initialization time for a FFT stage, c_3 models the time for twiddle factor loading, and c_4 models the time for a butterfly computation. The time for each butterfly computation is approximately 5.05 μsec, and the time for the loading of a twiddle factor 2.2 μsec.

For a one-dimensional FFT there is only one phase with inter-processor communication. The performance for the inter-processor communication stages is considerably

Axis length	Time msec	Mflops per sec
2	0.944	347
4	1.050	624
8	1.076	914
16	1.005	1304
32	1.062	1543
64	2.026	1941
128	3.879	2366
256	8.397	2497
512	17.436	2706
1024	38.629	2714
2048	79.704	2894
4096	169.333	2972
8192	353.905	3081

Table 5: Performance for 2048 concurrent local radix-2 DIF FFT.

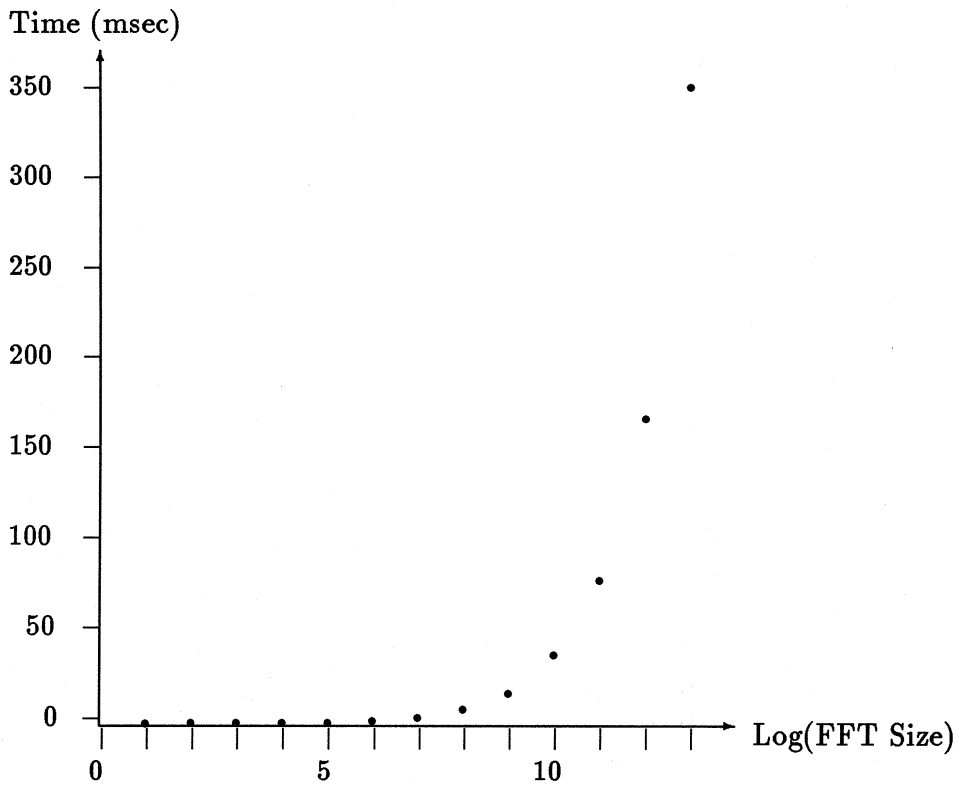

Figure 3: The execution time for local radix-2 DIF FFT.

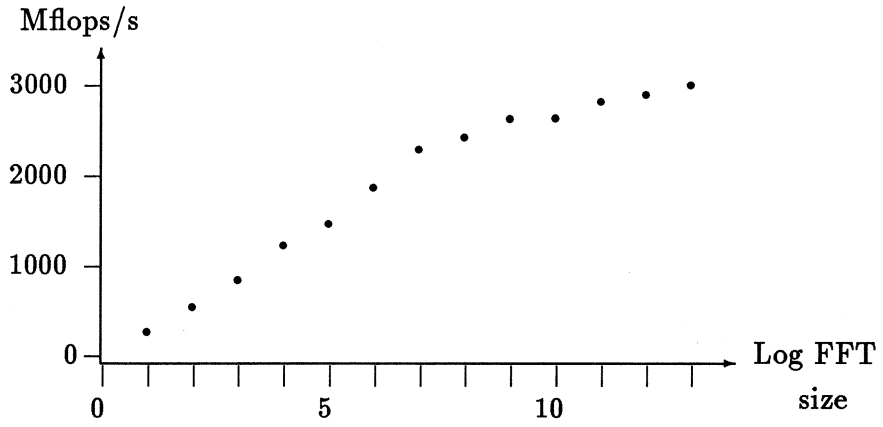

Figure 4: The performance of 2048 concurrent local radix-2 FFT.

less than for the local stages. Some performance data for a purely inter-processor, one-dimensional, radix-2 DIF FFT are given in Table 6 and Figures 5, 6, and 7. The behavior of the execution time clearly reflects the additional memory moves when the number of inter-processor communication stages is not a power of four, and the increased effective use of the floating-point unit as the number of inter-processor dimensions approaches a multiple of four. As the number of elements along the sequential axis increases the pipeline start-up and shut-down phases become less significant, and the performance measured in Mflops/s increases.

The performance for a single one-dimensional radix-2 DIF FFT as a function of Connection Machine system size, and FFT size for a few fixed ratio's of FFT size to machine size are given in Table 7 and Figures 8, 9, and 10 for a few different machine sizes. The entry for one floating-point unit gives the performance for the local FFT. The performance for this part of the total FFT increases with size to 3081 Mflops/s. The performance variation in the inter-processor communication part of the FFT is apparent.

Table 8 and Figures 11 and 12 shows the performance for fixed size DIF FFT of size 8k, 128k, and 2048k as a function of the number of floating-point units used for the computation. For a given size FFT the efficiency for the local part decreases as the data set is allocated to more processors. For the inter-processor communication part the efficiency increases with the number of inter-processor dimensions, and decreases with a reduced number of data elements per floating-point unit. The net effect is that for one to three inter-processor dimensions the efficiency is approximately constant. For four dimensions the efficiency increases by about 10%, due to the significantly more efficient computation of the butterflies for the inter-processor dimensions. Tables 9 and 10 give performance data for FFT sizes in the range of 1M – 16M points. The execution times and floating-point rates are shown in Figures 13 and 14 for a 64k CM–2 system.

Some sample timings for two- and three-dimensional radix-2 FFT are given in Tables

Seq axis	fpu/ FFT	Number of conc. FFT	Time msec	Mflops per sec
32	2	1024	2.552	128
	4	512	2.764	237
	8	256	2.990	328
	16	128	2.950	444
	32	64	3.583	457
	64	32	3.819	514
	128	16	4.053	566
	256	8	4.092	641
	512	4	4.816	612
	1024	2	4.870	673
	2048	1	5.136	702
512	2	1024	39.92	131
	4	512	41.18	255
	8	256	42.43	371
	16	128	39.32	533
	32	64	49.32	532
	64	32	51.16	615
	128	16	52.43	700
	256	8	49.30	851
	512	4	59.29	796
	1024	2	61.12	858
	2048	1	62.42	924
8192	2	1024	628.9	133
	4	512	658.7	255
	8	256	668.9	376
	16	128	608.9	551
	32	64	778.9	538
	64	32	798.7	630
	128	16	828.9	708
	256	8	758.9	884
	512	4	928.8	812
	1024	2	948.7	884
	2048	1	978.7	943

Table 6: The performance of inter-processor, one-dimensional, radix-2 DIF FFT.

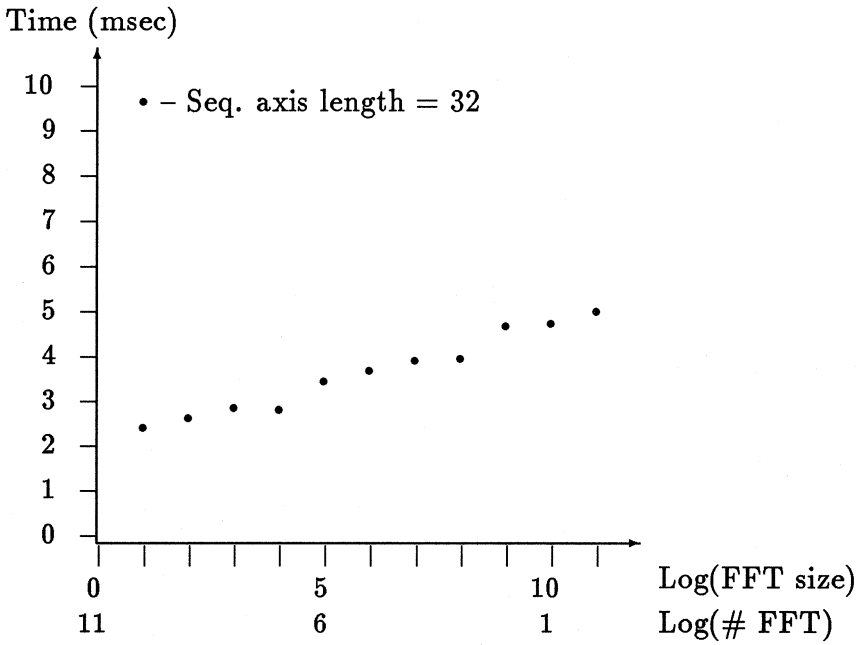

Figure 5: The execution time of one-dimensional radix-2 DIF FFTs with one complex point per processor. Sequential axis length 32.

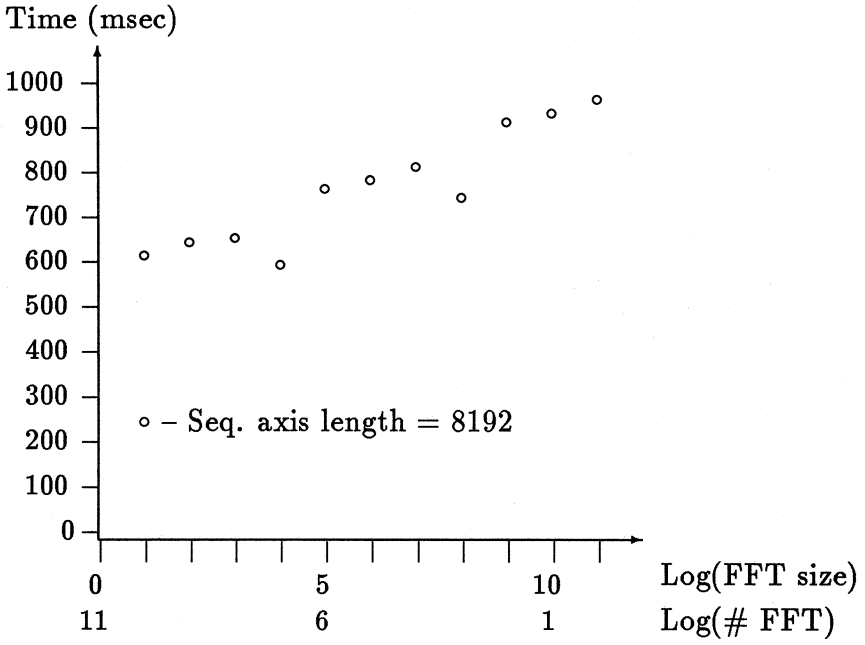

Figure 6: The execution time of one-dimensional radix-2 DIF FFTs with one complex point per processor. Sequential axis length 8192.

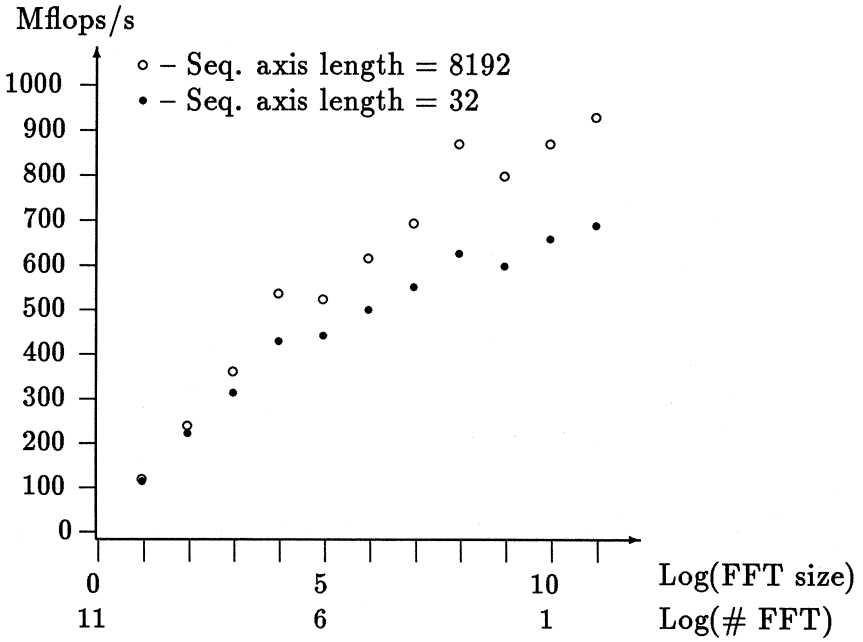

Figure 7: The floating-point rate for inter-processor, one-dimensional, radix-2 DIF FFTs with one complex point per processor.

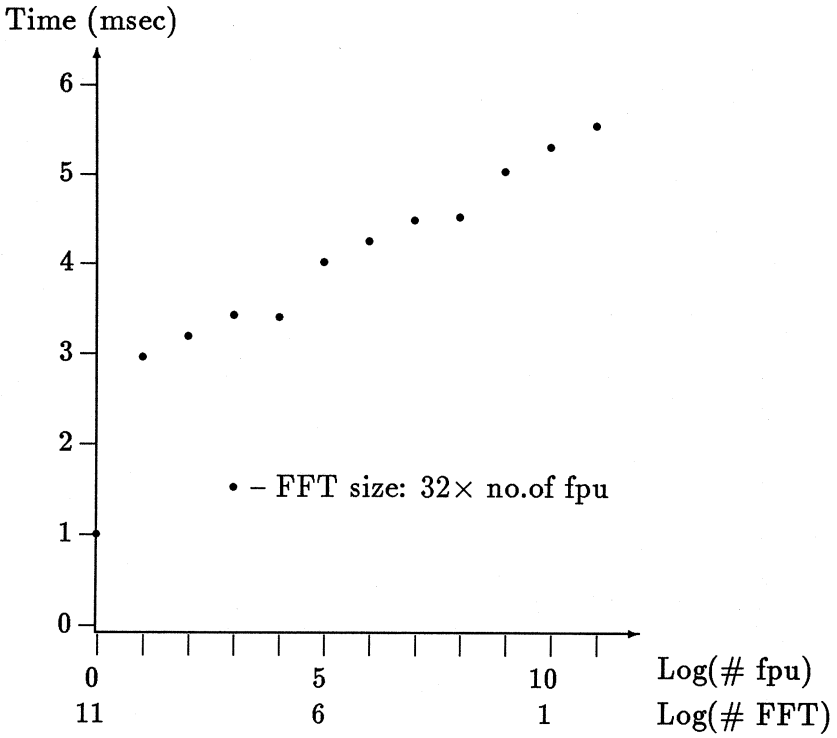

Figure 8: The execution time for a single one-dimensional, radix-2 DIF FFT with 32 points per fpu.

Elements per fpu	FFT size	No. of fpu	Number of conc. FFT	Time msec	Mflops per sec
	32	1	2048	1.073	1527
	64	2	1024	3.036	648
	128	4	512	3.270	701
	256	8	256	3.500	749
	512	16	128	3.482	847
32	1024	32	64	4.092	801
	2048	64	32	4.325	833
	4096	128	16	4.557	863
	8192	256	8	4.592	928
	16384	512	4	5.106	898
	32768	1024	2	5.381	913
	65536	2048	1	5.620	933
	512	1	2048	18.01	2619
	1k	2	1024	52.49	999
	2k	4	512	54.31	1062
	4k	8	256	55.57	1132
	8k	16	128	52.42	1300
512	16k	32	64	62.44	1175
	32k	64	32	64.19	1225
	64k	128	16	66.04	1270
	128k	256	8	61.82	1442
	256k	512	4	72.43	1303
	512k	1024	2	73.68	1352
	1024k	2048	1	74.94	1399
	8k	1	2048	358.7	3040
	16k	2	1024	928.8	1277
	32k	4	512	948.7	1326
	64k	8	256	968.8	1385
	128k	16	128	908.8	1569
8192	256k	32	64	1069	1413
	512k	64	32	1099	1451
	1024k	128	16	1109	1513
	2048k	256	8	1059	1664
	4096k	512	4	1219	1514
	8192k	1024	2	1239	1558
	16384k	2048	1	1269	1587

Table 7: Performance for a single one-dimensional radix-2 DIF FFT distributed over a number of floating-point units.

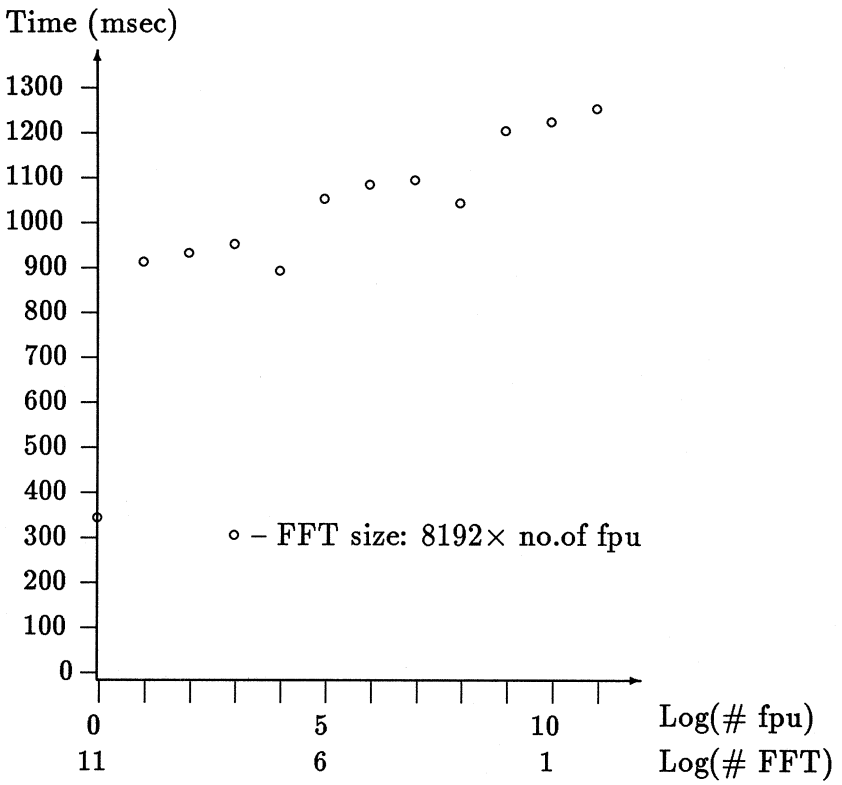

Figure 9: The execution time for a single one-dimensional, radix-2 DIF FFT with 8192 points per fpu.

Number of fpu's per FFT	Number of FFT	Time (msec)			Mflops/s		
		FFT size					
		8k	128k	2048k	8k	128k	2048k
1	2048	358.9			3040		
2	1024	449.4			1213		
4	512	224.3			1215		
8	256	114.8			1187		
16	128	52.39	908.7		1301	1569	
32	64	31.20	524.4		1092	1360	
64	32	16.07	264.7		1061	1347	
128	16	8.66	133.6		984	1334	
256	8	4.63	61.80	1059	919	1442	1664
512	4		36.20	604.4		1231	1457
1024	2		18.88	302.2		1180	1457
2048	1		10.22	153.6		1090	1434

Table 8: Execution time and floating-point rates for a 64k processor CM-2.

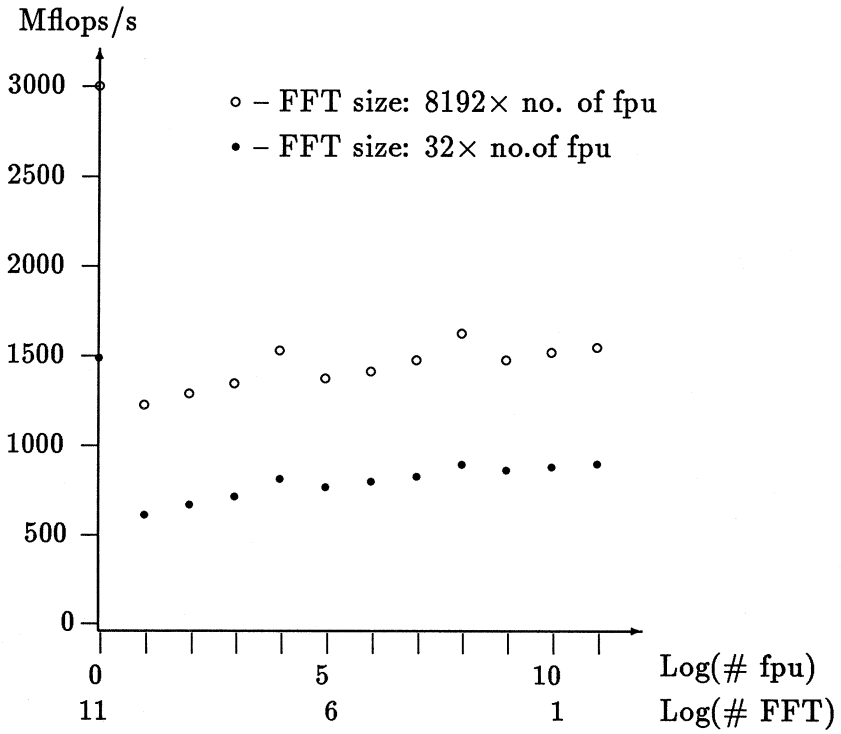

Figure 10: The floating-point rate for one-dimensional radix-2 DIF FFT of a size proportional to the number of fpu's per FFT.

Number of fpu's per FFT	Number of FFT	FFT size				
		1024k	2048k	4096k	8192k	16384k
128	16	1.119	—	—	—	—
256	8	0.519	1.059	—	—	—
512	4	0.297	0.604	1.218	—	—
1024	2	0.150	0.302	0.614	1.248	—
2048	1	0.075	0.154	0.307	0.624	1.259

Table 9: Execution times for a 64k processor CM-2.

Number of fpu's per FFT	Number of FFT	FFT size				
		1024k	2048k	4096k	8192k	16384k
128	16	1500	—	—	—	—
256	8	1615	1664	—	—	—
512	4	1411	1457	1515	—	—
1024	2	1400	1457	1502	1545	—
2048	1	1399	1434	1502	1545	1600

Table 10: Execution rates for a 64k processor CM-2.

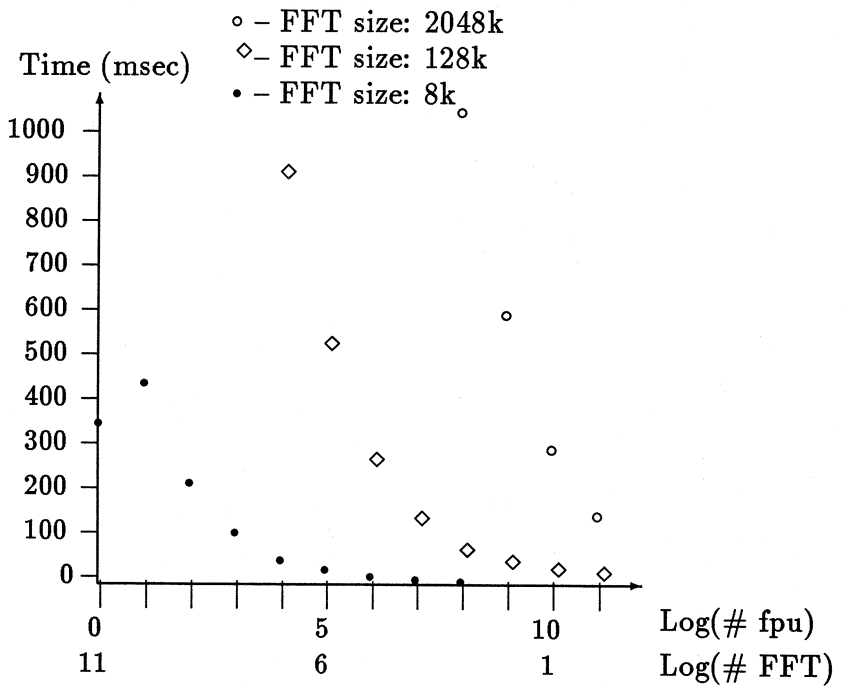

Figure 11: The execution time of one-dimensional radix-2 DIF FFT of size 8k, 128k, and 2048k as a function of the number of fpu's.

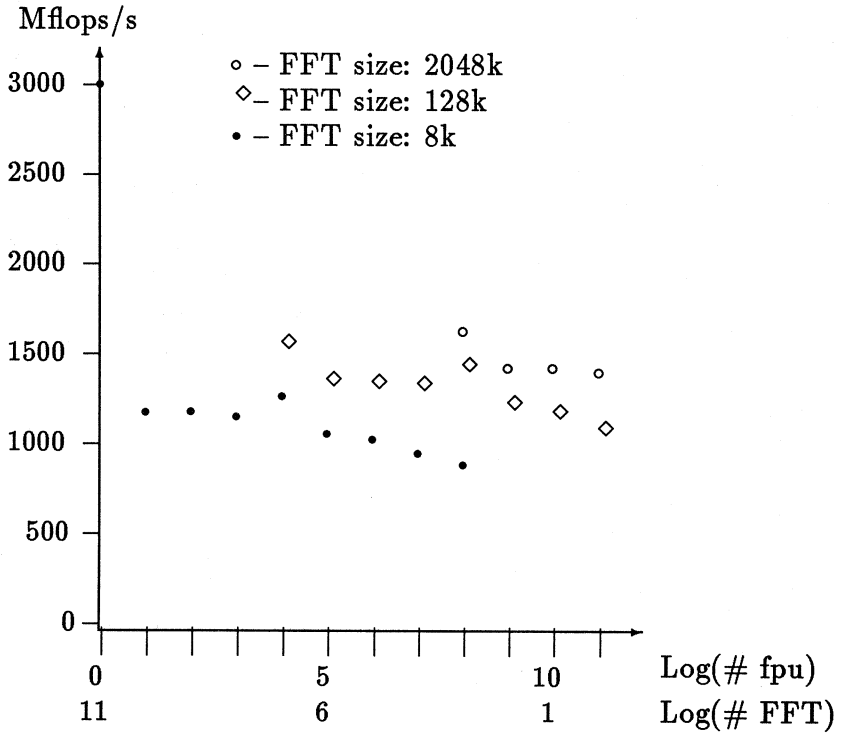

Figure 12: The floating-point rate for one-dimensional radix-2 DIF FFT of size 8k, 128k, and 2048k as a function of the number of fpu's/FFT for a 64k processor CM–2.

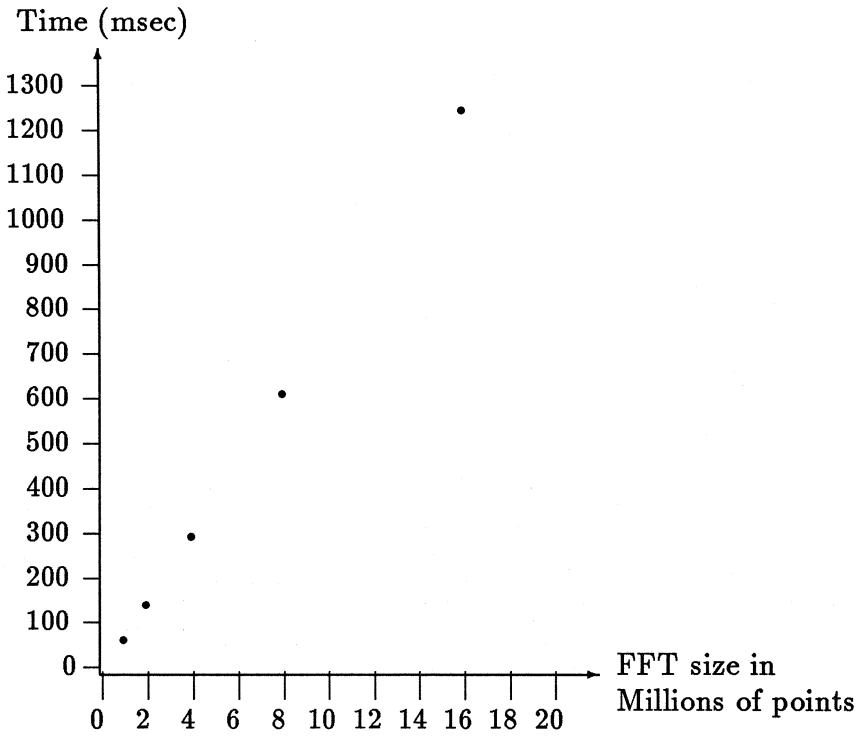

Figure 13: The execution time of one-dimensional radix-2 DIF FFT on a 64k CM–2 system.

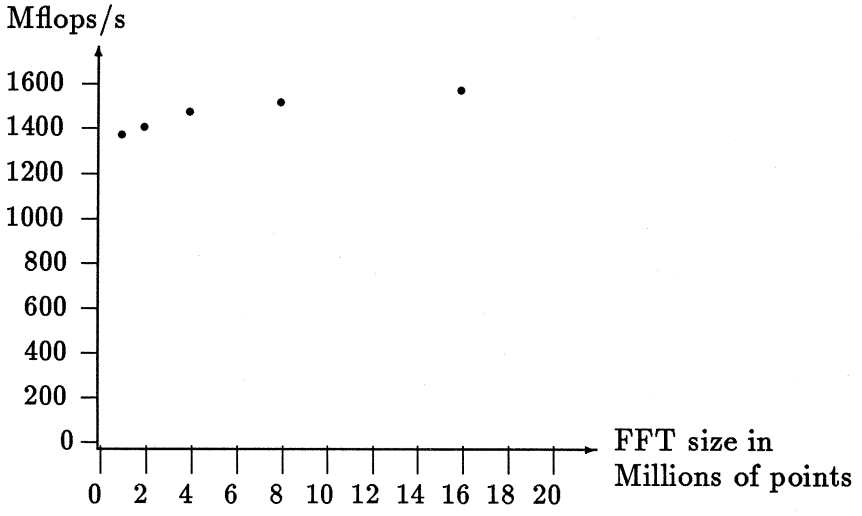

Figure 14: The execution rate for one-dimensional radix-2 DIF FFT on a 64k CM–2 system.

FFT size	Number of fpu					
	64	128	256	512	1024	2048
128×128	33.77	16.53	13.17	7.01		
256×256	139.0	70.34	31.17	24.65	13.18	7.25
512×512	575.4	291.6	136.0	72.43	51.53	26.29
1024×1024	2328	1194	562.9	317.0	149.9	114.0
2048×2048			2292	1298	646.9	307.2
4096×4096					2686	1343
32×32× 32	96.96	49.07	23.04	13.00	7.29	
64×64× 64	544.4	404.4	195.5	105.8	51.53	28.22
128×128×128		2319	1589	873.9	436.7	208.8
256×256×256					3569	1788

Table 11: Execution time in msec for some two and three dimensional radix-2 DIF FFT.

11 and 12.

5.3 Optimizing the configuration of the address space

The execution time for a one-dimensional FFT is normally minimized by spreading the data over as many floating point units as possible, as this permits use of the maximum number of processors. The only exception to this rule is if an FFT can be spread over either one or two processors; in this case, it is usually more efficient to arrange the data so that the entire vector fits on a single unit, as no communication takes place. FFT's that are short enough to fit inside a single processor should be performed in that manner if the number of instances is at least one-half of the number of FPU's in the system. This allocation makes the computation "embarrassingly" parallel.

If it is necessary to allocate an axis to more than a single processor, or the number of instances desired is less than half of the number of processors available, the data should be spread over as many processors as possible to minimize the the number of data elements per processor. Due to the current implementation, efficiency is greatest if the number of non-local dimensions is a multiple of four (either four or eight).

To optimize a multi-axis FFT, similar principles should be followed. Optimal efficiency is attained by firstly minimizing the number of axes that have any non-local component, then by minimizing the number of axes that have neither four nor eight non-local dimensions.

With a fixed number of processors N the optimum assignment of processors N_0 to the

FFT size	Number of fpu					
	64	128	256	512	1024	2048
128×128	1089	1110	698	656		
256×256	1207	1193	1346	851	796	723
512×512	1312	1295	1388	1303	916	897
1024×1024	1441	1405	1490	1323	1399	920
2048×2048			1610	1421	1426	1502
4096×4096					1499	1499
32×32× 32	811	801	853	756	674	
64×64× 64	1387	933	965	892	916	836
128×128×128		1519	1108	1008	1008	1055
256×256×256					1128	1126

Table 12: Floating-point rates in Mflop/s for some two and three dimensional radix-2 DIF FFT.

Number of fpu	M	N_0											
		1	2	4	8	16	32	64	128	256	512	1024	2048
128	256	16.20	50.57	50.57	51.11	45.86	56.82	58.94	59.01				
	1024	308.9	838.9	838.8	848.7	748.7	905.6	923.7	950.9				
512	512	17.45	51.73	51.82	52.43	47.34	56.48	58.86	60.80	55.53	59.28		
	2048	318.8	858.8	858.7	857.9	776.4	928.8	937.1	942.8	894.3	1053.3		
2048	1024		53.06	53.08	53.06	48.68	57.43	57.94	60.59	56.37	64.37	61.18	
	4096	339.0	868.9	878.9	879.0	799.0	949.0	949.0	959.0	877.7	1076	1072	1066

Table 13: Execution time (msec) for a one-dimensional FFT on a square array as a function of processor configuration.

axis of the FFT and N_1 to other axes can be determined by an expression of the form

$$\frac{\Pi_{j=0}^{k} M_j}{N}\left(\frac{\log_2 \frac{M_0}{N_0}}{r_\ell(\frac{M_0}{N_0})} + \frac{\log_2 N_0}{r_c(\log_2 N_0, \frac{\Pi_{j=0}^{k} M_j}{N})}\right)$$

where the length of array axis j is M_j, and the number of axis is k. The local floating-point rate is denoted r_ℓ, and the rate for inter-processor communication is r_c. An example of the sensitivity to the total performance as a function of the shape of the configuration of the address space is given in Table 13 and Figure 15. The array is assumed to be square of size $M \times M$, and the set of floating-point units is configured as $N_0 \times N_1$ processors.

The optimum machine configuration for multi-dimensional FFT can be found in much the same way as in the one-dimensional multi-axes FFT case. With a fixed number of processors N the optimum assignment of processors N_j to array axis $j < m$ on which an FFT shall be computed and all other axes, can be determined by an expression of the

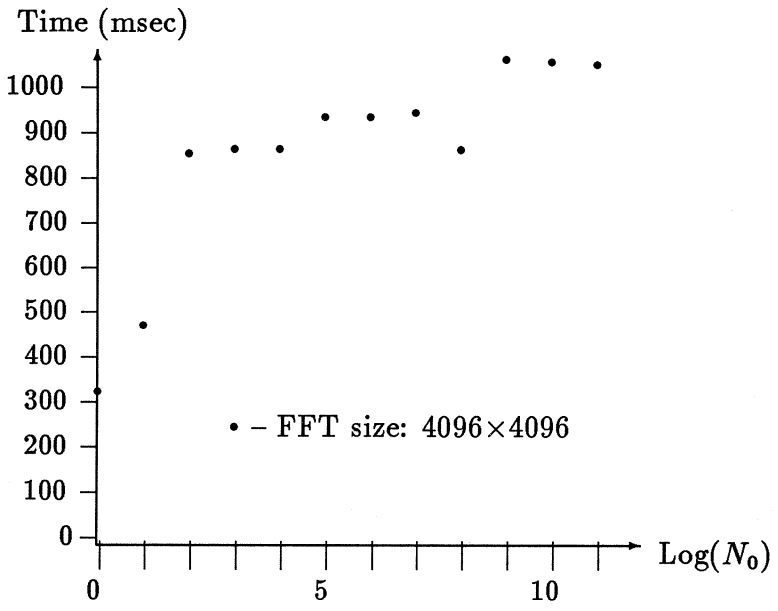

Figure 15: Total execution time for a one-dimensional FFT on a square array as a function of the configuration of 2048 fpu's.

form

$$\frac{\Pi_{j=0}^{k} M_j}{N}\left(\sum_{j=0}^{m-1} \frac{\log_2 \frac{M_j}{N_j}}{r_\ell(\frac{M_j}{N_j})} + \sum_{j=0}^{m-1} \frac{\log_2 N_j}{r_c(\log_2 N_j, \frac{\Pi_{j=0}^{k} M_j}{N})}\right)$$

where the length of array axis j is M_j, and the number of axis is k. The local floating-point rate is denoted r_ℓ, and the rate for inter-processor communication is r_c, as before. An example of the sensitivity to the total performance as a function of the shape of the configuration of the address space is given in Table 14 and Figures 16 and 17. The array is assumed to be square of size $M \times M$, and the set of floating-point units is configured as $N_0 \times N_1$ processors.

Number of fpu	M	N_0											
		1	2	4	8	16	32	64	128	256	512	1024	2048
128	256	69.79	103.8	102.4	92.45	92.71	103.1	104.2	71.20				
	1024	1183	1697	1679	1539	1539	1689	1712	1178				
512	512	72.39	101.5	107.3	105.6	99.16	99.87	106.8	107.7	102.3	72.45		
	2048	1304	1691	1739	1739	1639	1637	1737	1743	1694	1313		
2048	1024		108.7	114.1	105.0	104.2	111.7	111.0	104.2	105.5	113.1		
	4096	1342	1888	1872	1698	1678	1829	1829	1698	1705	1877	1894	1348

Table 14: Execution time (msec) for a two-dimensional FFT on a square array as a function processor configuration.

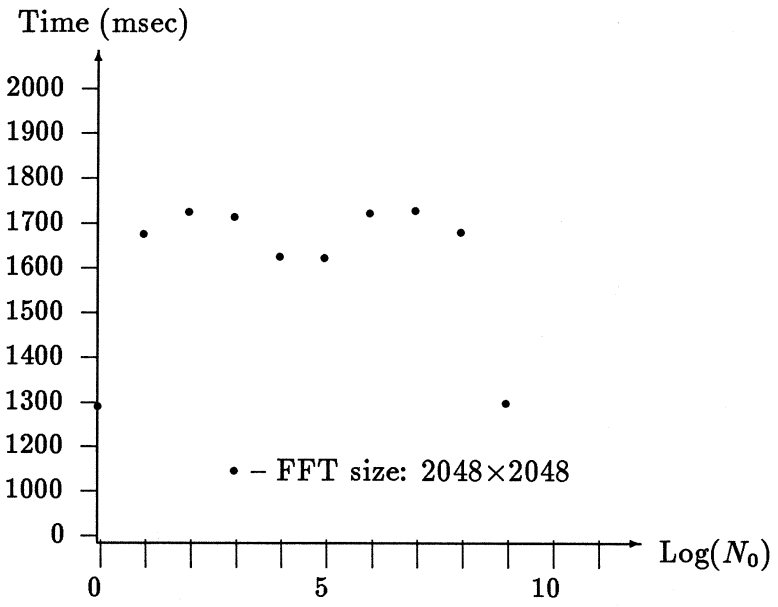

Figure 16: Total execution time for a two-dimensional FFT on a square array as a function of the configuration of 512 fpu's.

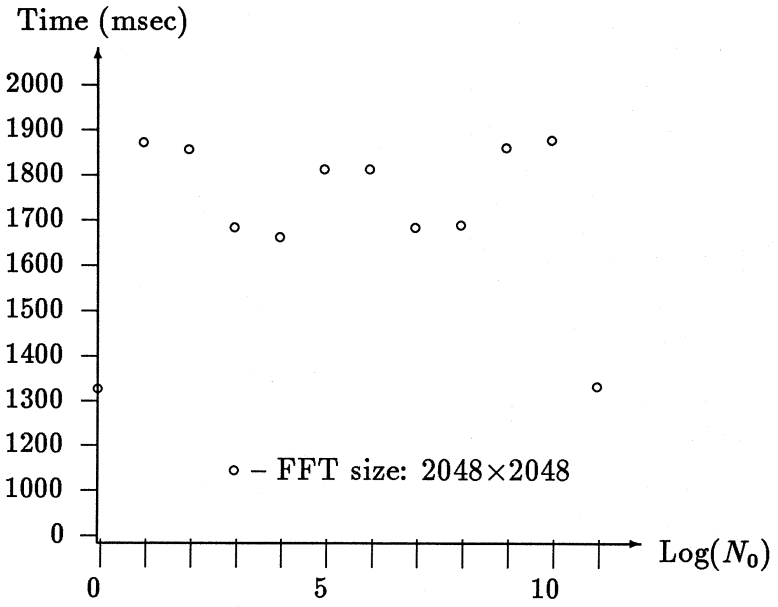

Figure 17: Total execution time for a two-dimensional FFT on a square array as a function of the configuration of 512 fpu's.

6 Summary

We have presented a radix-2 FFT for the Connection Machine that efficiently uses the communication system in Boolean cube networks. With the combination of consecutive storage, normal order input, decimation-in-time FFT, and bit-reversed input, decimation-in-frequency FFT, the requirements for twiddle factor storage is $\frac{P}{2N} + \log_2 N - 2$ twiddle factors per processor for a data set of P elements uniformly distributed across N processors. A performance of 3 Gflops/sec for the local FFT is achieved, and for the global FFT the peak performance is 1.7 Gflops/s. The maximum size FFT that can be performed in 512 Mbytes of storage is 32M complex points.

The performance can be enhanced by using higher radix FFT. For the local FFT a performance enhancement by a factor of about three is possible with the register set on the currently used floating-point unit. For the inter-processor communication the performance can also be enhanced by a higher radix FFT by improving the load balance. The performance enhancement compared to a radix-2 FFT will not be as significant as for the local FFT.

The performance of the radix-2 FFT can also be improved somewhat by removing some coding deficiencies. The number of twiddle factor loadings during the local FFT can be reduced for higher dimensional FFT by computing the same butterfly stage for all independent FFTs in succession, instead of complete FFTs in succession. For the inter-processor communication phase, performing individual butterflies instead of sets of four, or four butterflies for the same communications channel instead of different channels, will enhance performance somewhat. The expected performance gain from improvements of the code is at most 10%.

Acknowledgment

Many people have contributed in many ways to the radix-2 FFT routines in the Connection Machine Scientific Software Library. Alex Vasilevsky wrote the routines for the local butterfly computations. Alan Ruttenberg wrote the first version of the complete radix-2 FFT. Mike McKenna wrote the current Connection Machine local memory reordering routine. Ching-Tien Ho of Yale University contributed many ideas to the memory reordering algorithm. Tom Kraay of MRJ has independently implemented the reduced storage twiddle factor scheme in a FFT (proprietary to MRJ) based on cyclic data allocation.

References

[1] M.Y. Chan. Dilation-2 embeddings of grids into hypercubes. Technical Report UTDCS 1-88, Computer Science Dept., University of Texas at Dallas, 1988.

[2] M.Y. Chan. Embeddings of 3-dimensional grids into optimal hypercubes. Technical report, Computer Science Dept., University of Texas at Dallas, 1988. To appear in the Proceedings of the Fourth Conference on Hypercubes, Concurrent Computers, and Applications, March, 1989.

[3] Jim C. Cooley, P.A.W. Tukey, and P.D. Welch. *J. Sound Vibrations*, 12(3):315–337, 1970.

[4] Peter M. Flanders. A unified approach to a class of data movements on an array processor. *IEEE Trans. Computers*, 31(9):809–819, September 1982.

[5] I. Havel and J. Móravek. B-valuations of graphs. *Czech. Math. J.*, 22:338–351, 1972.

[6] Ching-Tien Ho and S. Lennart Johnsson. On the embedding of arbitrary meshes in Boolean cubes with expansion two dilation two. In *1987 International Conf. on Parallel Processing*, pages 188–191. IEEE Computer Society, 1987.

[7] Ching-Tien Ho and S. Lennart Johnsson. Stable dimension permutations on Boolean cubes. Technical Report YALEU/DCS/RR-617, Department of Computer Science, Yale University, October 1988.

[8] Ching-Tien Ho and S. Lennart Johnsson. Embedding meshes in boolean cubes by graph decomposition. *Journal of Parallel and Distributed Computing*, 7(3), December 1989. Technical Report YALEU/DCS/RR-746, Department of Computer Science Yale University, September 1989. This is a revision of Technical Report YALEU/DCS/RR-689 March 1989.

[9] Michel Jacquemin and S. Lennart Johnsson. Radix-4 and radix-8 fft on the connection machine. Technical report, Thinking Machines Corp., 1989. in Preparation.

[10] 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.

[11] S. Lennart Johnsson. Odd-even cyclic reduction on ensemble architectures and the solution tridiagonal systems of equations. Technical Report YALE/DCS/RR-339, Dept. of Computer Science, Yale University, October 1984.

[12] S. Lennart Johnsson. Communication efficient basic linear algebra computations on hypercube architectures. *J. Parallel Distributed Comput.*, 4(2):133–172, April 1987.

[13] S. Lennart Johnsson. Solving tridiagonal systems on ensemble architectures. *SIAM J. Sci. Statist. Comput.*, 8(3):354–392, May 1987.

[14] S. Lennart Johnsson. *Optimal Communication in Distributed and Shared Memory Models of Computation on Network Architectures.* Morgan Kaufman, 1989.

[15] 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 University, New Haven, CT, June 1987.

[16] S. Lennart Johnsson and Ching-Tien Ho. Shuffle permutations on Boolean cubes. Technical Report YALEU/DCS/RR-653, Department of Computer Science, Yale University, October 1988.

[17] S. Lennart Johnsson and Ching-Tien Ho. Emulating butterfly networks on gray code encoded data in boolean cubes. Technical report, Department of Computer Science, Yale University, 1989. in Preparation.

[18] S. Lennart Johnsson and Ching-Tien Ho. Spanning graphs for optimum broadcasting and personalized communication in hypercubes. *IEEE Trans. Computers*, 38(9):1249–1268, September 1989.

[19] S. Lennart Johnsson, Ching-Tien Ho, Michel Jacquemin, and Alan Ruttenberg. Computing fast Fourier transforms on Boolean cubes and related networks. In *Advanced Algorithms and Architectures for Signal Processing II*, volume 826, pages 223–231. Society of Photo-Optical Instrumentation Engineers, 1987.

[20] S. Lennart Johnsson and Peggy Li. Solutionset for ama/cs 146. Technical Report 5085:DF:83, California Institute of Technology, May 1983.

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

[22] E M. Reingold, J Nievergelt, and N Deo. *Combinatorial Algorithms.* Prentice-Hall, Englewood Cliffs. NJ, 1977.

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