A Radix-2 FFT on the Connection Machine

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

YALEU/DCS/TR-734
September 1989

To appear in the Proceedings of Supercomputing 1989

A Radix-2 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. Of particular interest in distributed memory architectures such as the Connection Machine is the allocation of twiddle factors to processors. We show that with a consecutive data allocation scheme and normal order input a decimation-in-time FFT results in a factor of $log_2 N$ less storage for twiddle factors than a decimation-in-frequency FFT for N processors. Similarly, with consecutive storage and bit-reversed input a decimation-in-frequency FFT requires a factor of $log_2 N$ less storage than a decimation-in-time FFT. The performance of the local FFT has a peak of about 3 Gflops/s. The "global" FFT has a peak performance of about 1.7 Gflops/s.

1 Introduction

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 implementation is based on the pipelined algorithm described in [13].

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.

2 Data Allocation

2.1 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 per processor, then stores that many successive elements in each processor, *consecutive storage* [10]. Another frequently used address form is *cyclic* assignment, for which the lowest order address bits determine the *real processor* address. In the cyclic assignment 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 different Connection Machine languages provide different means for user controlled data allocation. In CM-Fortran compiler directives allow a user to specify 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. The extension is made by extending the axis length in all languages, except CM-Fortran. 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.

2.2 Encoding of array axes

In the common 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 [10, 14, 16]. This Gray code is periodic. 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 $2^{\sum_{i=1}^{d} \lceil \log_2 N_i \rceil} / \Pi_{i=1}^{d} N_i$, which may be as high as $\sim 2^d$ [4, 5]. The expansion can be reduced by allowing some successive array indices to be encoded at a Hamming distance of two [5, 1, 2, 7].

In CM-Fortran array axes are by default encoded in a binary-reflected Gray code for the off-chip segment of the address field. In the other languages the Gray code encoding is invoked by configuring the Connection Machine as a lattice of the appropriate number of dimensions.

2.3 Data Representation

All basic software on the system allocates the data serially to a processor. The bits of a word occupy successive memory addresses. But, 32 Connection Machine processors share a floating-point unit that can access the memories of the 32 processors in bit-slices. Hence, transposing the data of 32 processors from *field-wise* to *slice-wise* allocates each word across the memories of groups of 32 processors, each sharing a floating-point unit. A fully configured Connection Machine can be viewed as having 2048 32-bit wide processors, each with 256 kbytes to 1 Mbyte of memory. The FFT is developed with this view of the Connection Machine.

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.

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 [10, 11].

3 Radix-2 FFT

3.1 One-dimensional FFT

A radix-2 butterfly network for $P = 2^p$ inputs and outputs has $P(p+1)$ nodes. With the 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 the 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 in the node address can be interpreted as time. During step z the communication is between ranks z and $z+1$.

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. This embedding corresponds to a binary encoding of array indices.

With $N < P$ processors performing $\frac{N}{2}$ butterfly computations concurrently, $\frac{P}{N}$ butterfly computations must be performed sequentially in each stage. In each of the first n butterfly stages the lowest order $p-n$ bits are identical for the pair of data elements in a butterfly computation. The first n butterfly stages can be viewed as consisting of $\frac{P}{N}$ independent FFTs, each of size N with one complex data element per processor. This property was used in [9] for devising sorting algorithms on Boolean cubes. The independent FFTs can be pipelined. Every FFT performs communication in processor dimensions $n-1, n-2, \ldots, 0$. Each FFT is delayed by one communication with respect to the preceding one. After the n butterfly stages with interprocessor 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 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 $\frac{P}{N} \gg n$.

3.2 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. But with multiple elements per processor, even a one-dimensional FFT is equivalent to

performing multiple independent FFTs. The same techniques as have been described above are applicable.

3.3 Multi-dimensional FFT

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 a FFT.

3.4 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.

3.4.1 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 $x(j + \frac{P}{2})$, where $j = (j_{p-1}j_{p-2}\cdots j_0)$, is $(j_{p-1}) \times (j_{p-2}j_{p-3}\cdots j_0)$. For the second rank the exponent of the twiddle factor ω_P is $(j_{p-2}) \times (j_{p-3}j_{p-4}\cdots j_00)$ for the pairs in locations j and $j + \frac{P}{4}$. In general, for an *in-place* DIF radix-2 FFT [15] 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 $q \in [0, p-1]$ (address bit $p-q-1$) is $\omega_P^{(j_{p-q-1})\times(j_{p-q-2}j_{p-q-3}\cdots j_0)2^q}$. 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 with an end-off shift q steps.

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 $(\{j_{p-2}j_{p-3}\cdots j_n\}|j_{n-1}\cdots j_0)$, $(\{j_{p-3}j_{p-4}\cdots j_n\}|j_{n-1}\cdots j_0)2$, ..., $(j_{n-1}\cdots j_0)2^{p-n-1}$, or a maximum total of $\sum_{m=1}^{p-n} 2^{p-n-m} = \frac{P}{N} - 1$ for any processor. The notation $\{j_{p-2}j_{p-3}\cdots j_n\}$ 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 $(j_{p-1}j_{p-2}\cdots j_{p-n}), j_k = 0, k \in \{p-n+1, p-n+2, \ldots p-1\}$ and $j_n = 1$. This processor contains the data indices $(00\ldots01|\{j_{p-n-1}j_{n-2}\cdots j_0\})$. 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.

3.4.2 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 $\omega_P^{(j_{p-q-1})(j_{p-q}j_{p-q+1}\cdots j_{p-1})2^{p-1-q}}$, $q \in [0, p-1]$. 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.

3.4.3 Bit-reversed input

With the input in normal order the FFT computation proceeds from the highest order bit to the lowest order bit with respect to data being paired for a butterfly computation. 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.

3.4.4 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.

3.4.5 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 $\frac{\max_l(P_l)}{2}$. With separate storage of the twiddle factors for each axis the total storage is $\frac{\sum_l P_l}{2}$, which is less than the required storage for a one-dimensional FFT of size $\Pi_l P_l$.

3.4.6 Reduced twiddle factor storage

A reduction in the twiddle factor storage in a processor is possible by the following observation. 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 $\{j_1 j_2 \ldots j_{n-1}\}|j_n \ldots j_{p-1}$. The highest order bit j_1 corresponds to bit position $p-2$. Hence, $\{1 j_2 \ldots j_{n-1}\}|j_n \ldots j_{p-1} = \frac{P}{4} + \{0 j_2 \ldots j_{n-1}\}|j_n \ldots j_{p-1}$. But, $\omega_P^{j+\frac{P}{4}} = \omega_P^j \cdot e^{-\frac{2\pi i}{P}\frac{P}{4}} = -i \cdot \omega_P^j$. Half of the twiddle factors can be obtained from the other half without any arithmetic. It is easy to see that this property is true for all on-processor stages. For $P \gg N$ this observation effectively reduces the need for twiddle factor storage by a factor of two. 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 $p-3, p-4, \ldots$ 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 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 [8]. 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 $\frac{P}{N} + \log_2 N - 2$ in each of N processors for a 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

performed by a call to the Connection Machine router. Likewise, a reordering to normal order from bit-reversed order is made by a call to the router. An optimized bit-reversal routine based on the algorithms in [3, 17, 6, 12] is under development.

The FFT routines are developed for data stored slice-wise. The conversion from field-wise to slice-wise storage is performed after a potential remapping to binary encoding, but before the actual FFT computation. For programming convenience a reordering of the local memory is performed after the change of storage form such that the stride for the first axis is one, the stride for the second axis is equal to the length of the local part of the first axis, etc. The strides for each axis are therefore constant.

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 fact that several butterflies in a given stage have the same twiddle factors is used advantageously in the local FFT kernels. The same property is true for decimation-in-time FFT. The local loops are ordered such that the need for loading twiddle factors into the register of the floating-point unit is minimized. A total of $\frac{P}{N} - 1$ loadings of twiddle factors is performed.

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 minor performance difference (of up to $\sim 3\%$).

The part of the FFT requiring inter-processor communication performs an exchange of data across each inter-processor dimension included in the encoding of an array axis subject to FFT computations. One processor performs a complex addition while the other performs a complex subtraction. The sign change is integrated with the exchange of data. Only one of the processors performs a useful complex multiplication. Data exchange in different inter-processor dimensions is pipelined, as described earlier. 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.

In the implementation for which timings are reported here the 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. This difference is the main reason why the performance for the two FFTs differ (with up to $\sim 15\%$).

4.1 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

- Extract 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 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. The computations in a stage of the FFT 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 stages requiring inter-processor communication are computed separately from the twiddle factors for stages within the unit. The computation is performed by all floating-point units concurrently. The computations are completely uniform.

The twiddle factors for the local stages are also computed concurrently. All floating-point units compute the same number of twiddle factors, but the indices differ. The twiddle factors are computed in-place, which requires some care. The conversion from the field-wise programming model to the slice-wise, 32-bit wide model, interchanges the memory and on-chip address fields with the lowest order off-chip address bit appended to the on-chip field:

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

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 computations are:

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 $(yyyyyzz\underbrace{00\ldots0}_{q+n})$.

Each step 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 3.4, in order of stage number, and for each stage number in bit-reversed order.

Axis length	Time msec	Mflops /s
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 1: Performance for 2048 concurrent local radix-2 DIF FFT.

4.2 Performance measurements

The performance measurements have been made on Connection Machine configurations with 32-bit floating-point hardware. The performance of the local kernel for different sizes is given in Table 1 and Figure 1. 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.

Some performance data for a purely inter-processor, one-dimensional, radix-2 DIF FFT are given in Table 2 and Figure 2. 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.

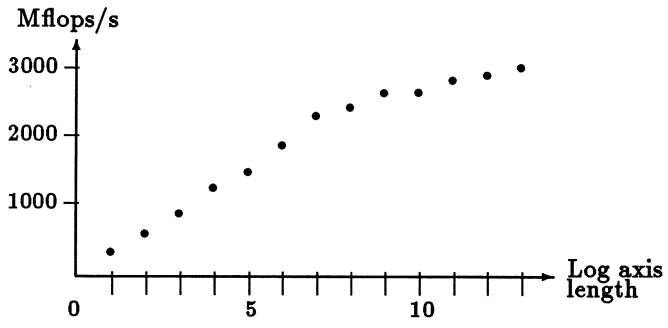

Figure 1: The performance of 2048 concurrent local radix-2 FFT.

Seq axis	fpu/ FFT	Number of conc. FFT	Time msec	Mflops /s
	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
32	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
	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
512	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
	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
8192	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 2: The performance of inter-processor, one-dimensional, radix-2 DIF FFT.

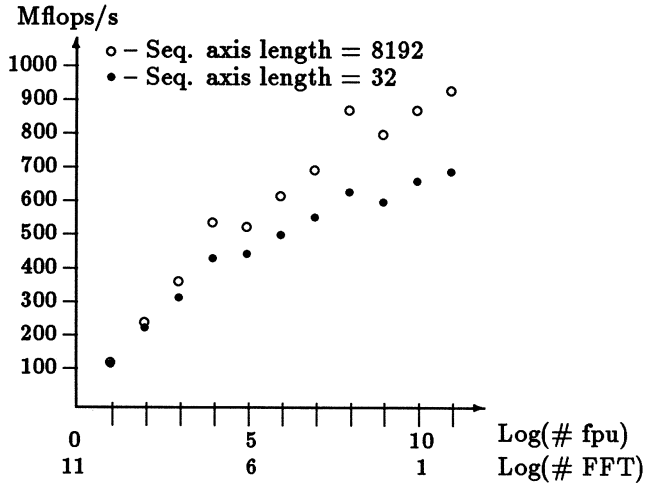

Figure 2: The floating-point rate for inter-processor, one-dimensional, radix-2 DIF FFT with one complex point per processor.

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 ratios of FFT size to machine size is given in Table 3 and Figure 3 for a few different machine sizes. Tables 4 and 5 and Figures 4 and 5 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.

Some sample timings for two- and three-dimensional radix-2 FFT are given in Tables 6 and 7.

4.3 Optimizing the configuration of the address space

The execution time for a one-dimensional FFT is minimized if the FFT the data set is spread over as many floating-point units as possible (but never over two units). The efficiency is highest for a single processor. For FFTs of a size requiring at least two floating-point units for memory reasons the time decreases monotonicly with the number of processors used. If an FFT shall be computed along a single axis of a multi-dimensional array, then effectively several FFTs must be performed. If the length of that axis is sufficiently short to fit in

Elements per fpu	FFT size	No. of fpu	Number of conc. FFT	Time msec	Mflops /s
32	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
	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	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
	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
8192	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
	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 3: Performance for a single one-dimensional radix-2 DIF FFT distributed over a number of floating-point units.

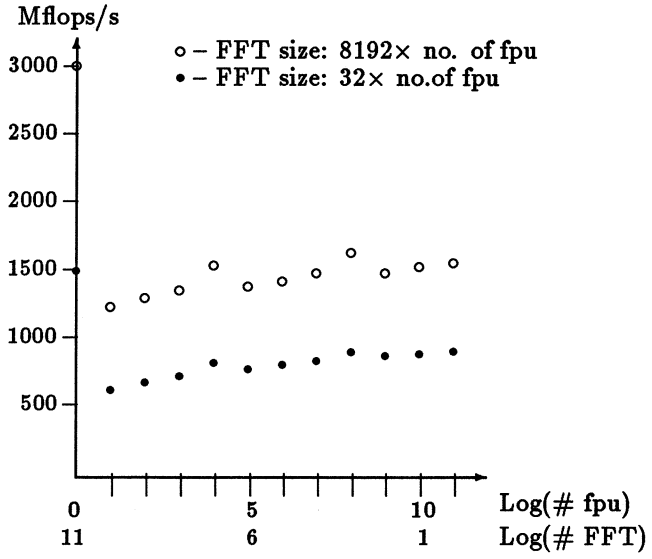

Figure 3: The floating-point rate for one-dimensional radix-2 DIF FFT of a size proportional to the number of fpu's per FFT.

Number of fpu's per FFT	Number of FFT	Size		
		8k	128k	2048k
1	2048	358.9		
2	1024	449.4		
4	512	224.3		
8	256	114.8		
16	128	52.39	908.7	
32	64	31.20	524.4	
64	32	16.07	264.7	
128	16	8.66	133.6	
256	8	4.63	61.80	1059
512	4		36.20	604.4
1024	2		18.88	302.2
2048	1		10.22	153.6

Table 4: Execution time in msec for a fully configured CM-2.

Number of fpu's per FFT	Number of FFT	Size		
		8k	128 k	2048k
1	2048	3040		
2	1024	1213		
4	512	1215		
8	256	1187		
16	128	1301	1569	
32	64	1092	1360	
64	32	1061	1347	
128	16	984	1334	
256	8	919	1442	1664
512	4		1231	1457
1024	2		1180	1457
2048	1		1090	1434

Table 5: Floating-point rates in Mflops/s for a fully configured CM-2.

FFT	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 6: Execution time in msec for some two and three dimensional radix-2 DIF FFT.

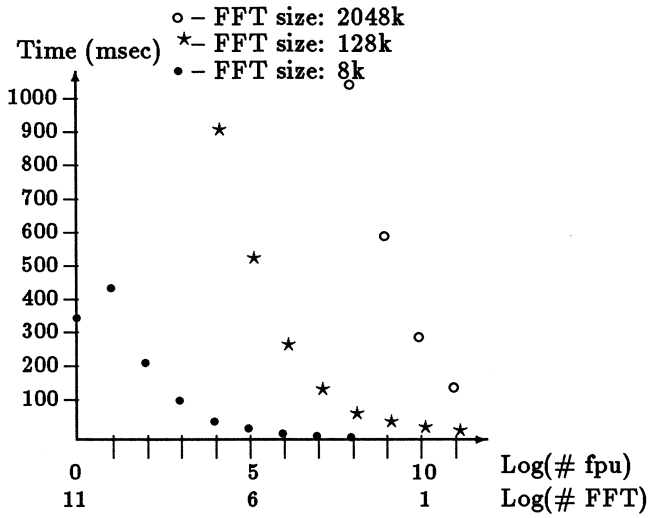

Figure 4: The execution time of one-dimensional radix-2 DIF FFT of size 8k, 128k, and 2048k as a function of the number of fpu's.

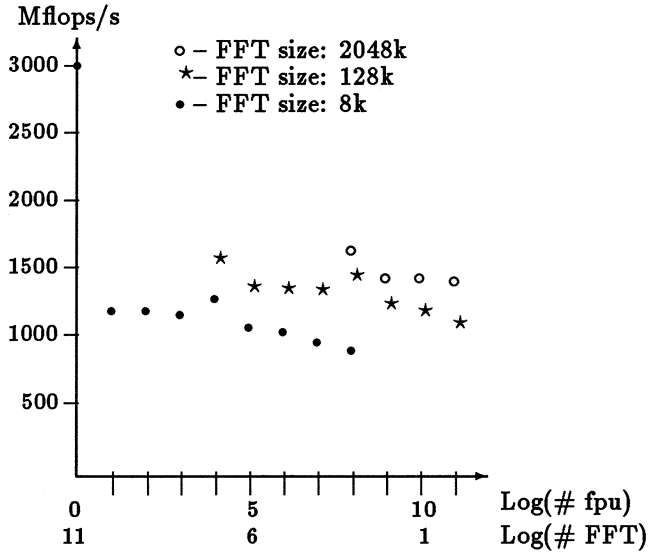

Figure 5: The floating-point rate for one-dimensional radix-2 DIF FFT of size 8k, 128k, and 2048k as a function of the number of fpu's/FFT for a fully configured CM–2.

FFT	Number of fpu					
	64	128	256	512	1024	2048
128x128	1089	1110	698	656		
256x256	1207	1193	1346	851	796	723
512x512	1312	1295	1388	1303	916	897
1024x1024	1441	1405	1490	1323	1399	920
2048x2048			1610	1421	1426	1502
4096x4096					1499	1499
32x32x 32	811	801	853	756	674	
64x64x 64	1387	933	965	892	916	836
128x128x128		1519	1108	1008	1008	1055
256x256x256					1128	1126

Table 7: Floating-point rates in Mflop/s for some two and three dimensional radix-2 DIF FFT.

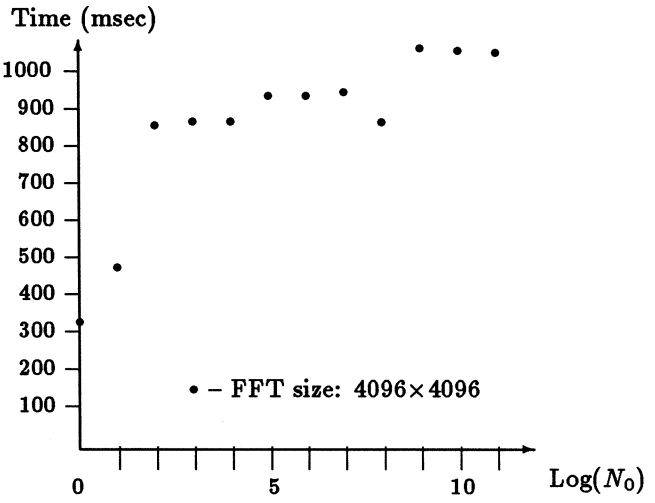

Figure 6: Total execution time for a one-dimensional FFT on a square array as a function of the configuration of 2048 fpu's.

the storage of a single floating-point unit, and the number of instances of this axis is greater or equal to half the number of floating-point units in the system, then the optimum allocation is with the axis entirely in local memory. This allocation makes the computations "embarrassingly" parallel. No communication is required. If an axis is allocated to more than a single processor, then in order to minimize the time the array shall be allocated to all processors as indicated by Tables 4 and 5, because the efficiency is sufficiently uniform. Preferably four or eight inter-processor dimensions shall be used for the axis subject to FFT computation. An example of the dependence of the total performance on the configuration of the address space is given in Table 8 and Figure 6. The data 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. An example of the dependence of the performance upon the configuration of the address space is given in Table 9 and Figure 7. 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.

5 Summary

We have presented a radix-2 FFT for the Connection Machine that efficiently uses the communication system of the Connection Machine. With the combination of consecutive storage, normal order input, decimation-in-time FFT, and bit-reversed input, decimation-in-

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

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			1874	1707	1688	1829	1829	1697	1697	1866	1890	1338

Table 9: Execution time (msec) for a two-dimensional FFT on a square array as a function processor configuration.

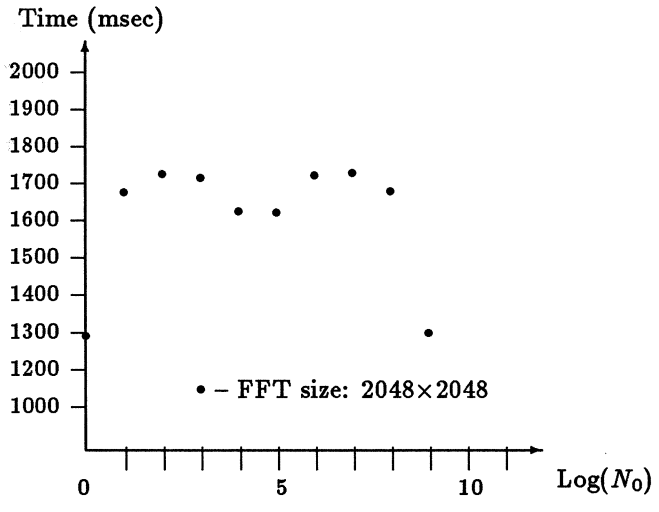

Figure 7: Total execution time for a two-dimensional FFT on a square array as a function of the configuration of 512 fpu's.

frequency FFT, the requirements for twiddle factor storage is $\frac{P}{N} + \log_2 N - 2$ twiddle factors per processor for a data set of P elements uniformly distributed across N processors. By computing half of the twiddle factors by performing 90-degree rotations "on-the-fly" a reduction in twiddle factor storage by a factor of two is possible.

Increased performance of the radix-2 FFT compared to the current implementation is possible by some loop reordering. The number of twiddle factor loadings during the local FFT can be reduced by computing the same butterfly stage for all independent FFTs in succession, instead of complete FFTs in succession. Furthermore, 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, is likely to enhance performance.

Performance can also be enhanced by 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.

If the data is encoded by Gray code, then an explicit reordering to binary order is performed before the FFT computation. The Connection Machine router is currently used for this reordering. An optimum reordering is given in [10].

Acknowledgement

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 version of the local memory reordering routine. 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] 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.

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

[5] 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.

[6] 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.

[7] Ching-Tien Ho and S. Lennart Johnsson. Embedding meshes in boolean cubes by graph decomposition. Technical Report YALEU/DCS/RR-689, Department of Computer Science, Yale University, March 1989.

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

[9] 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.

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

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

[12] 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.

[13] 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.

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

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

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

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