Cooley-Tukey FFT on the Connection Machine

Cooley-Tukey FFT on the Connection Machine

S. Lennart Johnsson
Robert L. Krawitz

TR-24-91

September 1991

Parallel Computing Research Group

Center for Research in Computing Technology
Harvard University
Cambridge, Massachusetts

Cooley-Tukey FFT on the Connection Machine

S. Lennart Johnsson[1] and Robert L. Krawitz

Thinking Machines Corp.
245 First Street,
Cambridge, MA 02142
Johnsson@think.com,rlk@think.com

Abstract

We describe an implementation of the Cooley Tukey complex-to-complex FFT on the Connection Machine. The implementation is designed to make effective use of the communications bandwidth of the architecture, its memory bandwidth, and storage with precomputed twiddle factors. The peak data motion rate that is achieved for the interprocessor communication stages is in excess of 7 Gbytes/s for a Connection Machine system CM-200 with 2048 floating-point processors. The peak rate of FFT computations local to a processor is 12.9 Gflops/s in 32-bit precision, and 10.7 Gflops/s in 64-bit precision. The same FFT routine is used to perform both one- and multi-dimensional FFT without any explicit data rearrangement. The peak performance for a one-dimensional FFT on data distributed over all processors is 5.4 Gflops/s in 32-bit precision and 3.2 Gflops/s in 64-bit precision. The peak performance for square, two-dimensional transforms, is 3.1 Gflops/s in 32-bit precision, and for cubic, three dimensional transforms, the peak is 2.0 Gflops/s in 64-bit precision. Certain oblong shapes yield better performance. The number of twiddle factors stored in each processor is $\frac{P}{2N} + \log_2 N$ for an FFT on P complex points uniformly distributed among N processors. To achieve this level of storage efficiency we show that a decimation-in-time FFT is required for normal order input, and a decimation-in-frequency FFT is required for bit-reversed input order.

1 Introduction

The main contribution of this paper is an efficient adaptation of the well known Cooley-Tukey [1] Fast Fourier Transform to multi-processors interconnected by a Boolean cube network. The focus is on three issues: efficient use of the communications bandwidth, efficient use of the memory bandwidth, and minimum storage requirements with precomputed twiddle factors. The algorithms we describe have been implemented on the Connection Machine systems CM-2 and CM-200, and are part of the Connection Machine Scientific Software Library. Performance data is provided for both one-dimensional and multi-dimensional complex-to-complex Fourier Transforms. The implementation uses a mix of radix-2, 4 and 8 kernels for the computations involving only local data, while computations requiring interaction between data in different memory units only use radix-2 kernels.

[1]Also affiliated with Division of Applied Sciences, Harvard University, Cambridge, MA 02138

The Discrete Fourier Transform (DFT) is defined by

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

and the Inverse Discrete Fourier Transform (IDFT) is defined by

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

The coefficients w_P^{lj} are known as *twiddle factors*. We consider both *decimation-in-time*, DIT, and *decimation-in-frequency*, DIF, versions of the Cooley-Tukey FFT [1]. The *decimation-in-time* FFT is defined by the splitting formula

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

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

and the *decimation-in-frequency* FFT is defined by the splitting formula

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

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

The data interactions for the two splitting formulas, also known as butterflies, are shown in Figures 1 and 2. The splitting formulas above define radix-2 FFTs. Higher radix FFTs such as radix-4 and radix-8 FFT are defined similarly, see e.g. [13]. Details of a Connection Machine implementation are reported in [12].

Both the DIF and the DIT FFT reorder the data from *normal* to *bit reversed* order (or the converse). The most important difference between the DIF and DIT FFTs with respect to distributed memory systems is that the DIF and DIT FFTs use the twiddle factors in opposite order. This difference causes a difference in the demand for twiddle factor storage by a factor of $\log_2 N$ for an N processor system, as discussed in section 4.

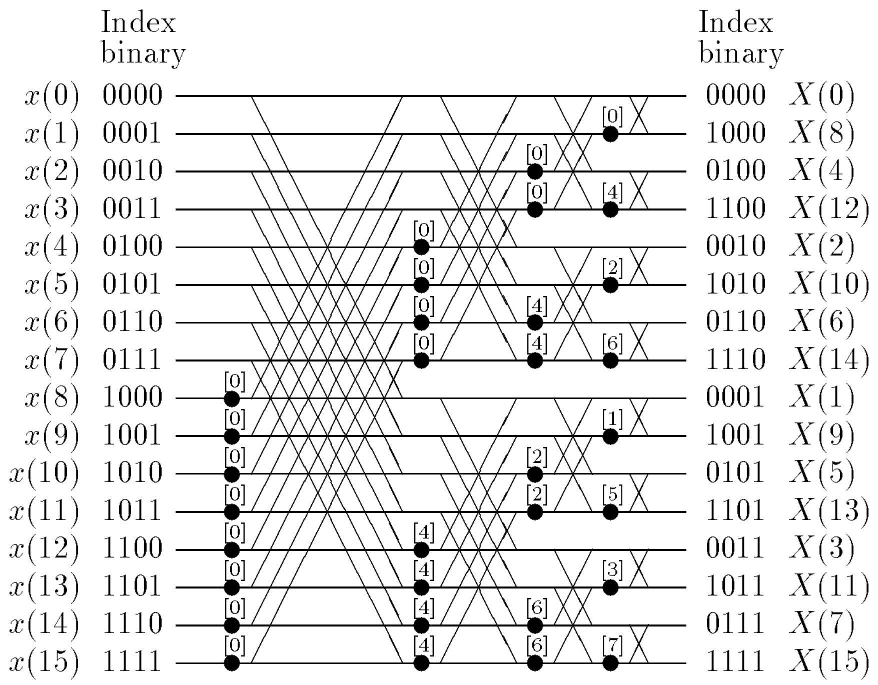

Figure 1: Decimation-in-time FFT.

The Connection Machine systems CM-2 and CM-200 have up to 2048 floating-point processors that support operations in both 32-bit and 64-bit precision. The memory is distributed among the processing units, with a maximum of 4 Mbytes of memory per unit, and a total memory of 8 Gbytes. Each processing unit has a single 32-bit wide data path to its memory. Data paths internal to the floating-point unit are 64-bits wide. The processing units are interconnected as an 11-dimensional Boolean cube, with two channels between every pair of nodes. Data may be sent on all 22 (11×2) channels concurrently.

The data layout has a significant impact on performance for most distributed memory architectures. In order to maximize the potential for parallelism data arrays are distributed evenly over the processor memories. Two common schemes for assigning data to processors are the consecutive (block) allocation, and cyclic (scatter) allocation [6, 3]. The two types of data allocation are illustrated in Figure 3 for a 32 element array distributed among 8 memory units. All Connection Machine compilers use the consecutive allocation scheme. However, for the FFT a cyclic data allocation would yield a lower communication time in some cases [11]. For multi-dimensional data arrays, the ability to configure the Connection Machine processors as a multi-dimensional array with axes lengths being arbitrary powers of two can be exploited for enhanced performance. The processor configuration is controlled by compiler directives.

The focus of the algorithm development presented here is an unordered FFT. The presented algorithms fully utilize the communication system. Reordering for ordered transforms is

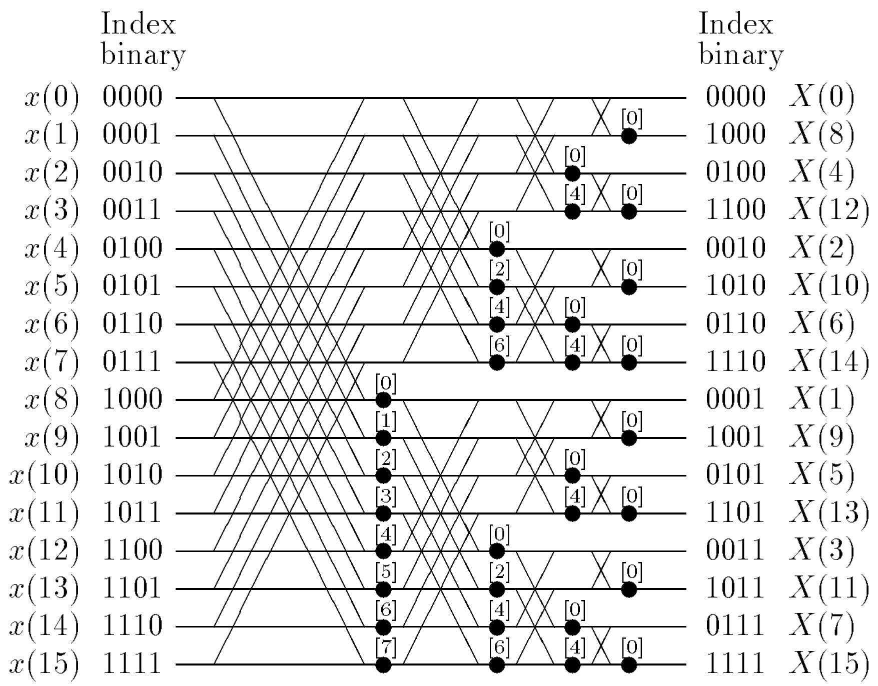

Figure 2: Decimation-in-frequency FFT.

Consecutive data allocation

P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7
0	4	8	12	16	20	24	28
1	5	9	13	17	21	25	29
2	6	10	14	18	22	26	30
3	7	11	15	19	23	27	31

Cyclic data allocation

P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7
0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31

Figure 3: Consecutive and cyclic data allocation of 32 elements to 8 processors.

performed explicitly. Reordering algorithms that fully use communications systems allowing concurrent communication on multiple channels are presented in, for example, [7, 8, 9]. An implementation on the Connection Machine is presented in [2]. For reference we include performance data for some ordered transforms. Interleaving the reordering with the FFT computation offers no advantage on a system allowing concurrent communication on all channels, unlike the case with communication restricted to one channel at a time [15, 16].

We also restrict the algorithmic considerations to data allocated to processors using the standard binary encoding, which is advantageous for the Cooley-Tukey FFT. For multi-processors configured as Boolean cube networks, a binary-reflected Gray code [14] is often used. In such an encoding successive array elements are allocated either in local memory, or in an adjacent processor. This type of data allocation is advantageous for algorithms requiring nearest neighbor communication, such as for instance relaxation algorithms. Algorithms that compute a Cooley-Tukey FFT on such a data allocation without loss of efficiency are presented in [10].

The outline of this paper is as follows. We first review the mapping of the radix-2 FFT to Boolean cube networks. Particular emphasis is placed on effective use of the communication system [11]. We then briefly review the issues in efficient use of the data path between a floating-point unit and its memory, followed by a detailed account of twiddle factor computation and storage in the distributed memory system. We focus on the consecutive data allocation, since all Connection Machine compilers use this allocation scheme. Whenever a cyclic data allocation would result in significantly different conclusions we will state the difference. A few details of the implementation are given in Section 6, and performance data are reported in Section 7. Potential improvements of the current implementation are reviewed in Section 8.

2 Radix-2 FFTs on Boolean Cubes

For the radix-2 Cooley-Tukey FFT the data interaction in stage q, $0 \leq q < p$, with stages labeled from input to output, is between data with indices i and $i \oplus 2^{p-1-q}$. $\oplus$ denotes the bit-wise exclusive-or function. For example, in Figures 1 and 2 a 16-point transform is computed with the data interaction in the first stage being between data elements that differ by 8 in their indices. In the second stage the interaction is between elements that differ by 4 in their indices, etc.

In a Boolean cube network nodes can be labeled such that there is one adjacent node for every bit in the node address, i.e., for every node with address i in a Boolean cube of n dimensions and $N = 2^n$ nodes, there are n neighbors with addresses $i \oplus 2^j$, $0 \leq j < n$. The data interaction for the radix-2 Cooley-Tukey FFT matches well with the connectivity of the Boolean cube. For a precise assessment of the communication requirements we consider in some detail the consecutive allocation of data to memory units. In the data index space the allocation can be represented as follows, where x_i denotes one bit in the encoding of the indices. The total number of bits required for the encoding of the data indices is p for a data set of size $P = 2^p$.

$$\textit{Consecutive}\ \text{assignment:}\qquad (\underbrace{x_{p-1}x_{p-2}\dots x_{p-n}}_{rp}\underbrace{x_{p-n-1}x_{p-n-2}\dots x_0})_{vp}$$

Bits assigned to the processor address field are labeled rp, and bits in the encoding of indices assigned to the local memory address field are labeled vp. With consecutive data allocation the n most significant bits identify the processor, and the least significant $p-n$ bits identify the local memory addresses. Thus, there are $\frac{P}{N}$ elements per processor. For instance, a data array of 1024 points require 10 bits for its binary encoding. With a consecutive mapping to 128 processors, there are 8 data points per processor. Seven bits are required for the encoding of processor addresses, and three for the local addresses. With the consecutive mapping elements 0 though 7 are mapped to the first processor, elements 8 through 15 to the second processor, etc. The three least significant bits of the data index encoding are used for local memory addresses.

For the consecutive data allocation the first n steps require inter-processor communication, while the last $p-n$ steps are local to a processor. If the data is allocated in bit-reversed order, then the most significant bit in the data index corresponds to address bit 0, and the least significant index bit to address bit $p-1$. The order of the inter-processor communication and the local reference phases are reversed. With the data in normal input order, the first stage in the Cooley-Tukey FFT requires communication in the most significant cube dimension. The following $n-1$ stages require communication in successively lower dimensions. By exchanging all $\frac{P}{N}$ local data elements between pairs of processors the computations can be split between a pair of processors; one computes the "top" of a butterfly, one the "bottom".

A direct implementation of the above scheme yields poor utilization of the communication system. In each of the first n stages only one cube dimension per stage is used. The communications efficiency is $\frac{1}{n}$. By pipelining the FFT stages the efficiency can be improved to almost 100%. Table 1 illustrates the idea of pipelining FFT stages. Columns represent processors, and rows local memory addresses. The first five memory locations are shown. The data allocation is assumed consecutive, and the entries in the table show the cube dimensions in which data are exchanged during the first four time steps. After the pipeline startup of $n-1$ stages, all n dimensions are used for every time step until the pipeline shut down takes place. Once the pipeline is filled, n data items are exchanged in all processors in every stage, until the pipeline shut down phase is reached. The total number of element exchanges in sequence is $\frac{P}{N}+n-1$, which is an improvement over the non-pipelined algorithm by a factor of approximately n.

Our FFT implementation on the Connection Machine uses the above scheme to achieve full utilization of the communication capabilities of the architecture. The two channels between each pair of nodes are used to exchange concurrently the real and imaginary parts of a data point.

Time Step	Memory location	Processor							
		0	1	2	3	4	5	6	7
0	0	2	2	2	2	2	2	2	2
	1	-	-	-	-	-	-	-	-
	2	-	-	-	-	-	-	-	-
	3	-	-	-	-	-	-	-	-
	4	-	-	-	-	-	-	-	-
1	0	1	1	1	1	1	1	1	1
	1	2	2	2	2	2	2	2	2
	2	-	-	-	-	-	-	-	-
	3	-	-	-	-	-	-	-	-
	4	-	-	-	-	-	-	-	-
2	0	0	0	0	0	0	0	0	0
	1	1	1	1	1	1	1	1	1
	2	2	2	2	2	2	2	2	2
	3	-	-	-	-	-	-	-	-
	4	-	-	-	-	-	-	-	-
3	0	-	-	-	-	-	-	-	-
	1	0	0	0	0	0	0	0	0
	2	1	1	1	1	1	1	1	1
	3	2	2	2	2	2	2	2	2
	4	-	-	-	-	-	-	-	-

Table 1: The communication for the first four steps of a radix-2 FFT with pipelined stages.

3 Minimizing local data motion

The most effective use of a memory hierarchy with respect to data transfer bandwidth [4, 5] is to use a high radix FFT algorithm. For each level in the hierarchy the radix shall be equal to the size of the maximum data set that fits in the memory at that level. With a single data path between a floating-point unit and its memory, the radix shall be chosen according to the number of registers in the floating-point unit. Table 2 summarizes the arithmetic and data motion requirements for radix-2, 4, and 8 FFT. Only the most significant terms are accounted for. The reduction in the need for memory bandwidth is much more significant than the reduction in the number of arithmetic operations. An FFT based on radix-8 kernels requires about 1/3 as many memory references as a radix-2 based FFT. The reduction in the required number of arithmetic operations is about 20%. The Connection Machine FFT uses a combination of radix-2, 4, and 8 kernels as required to fit the size of the local data set. A higher radix than 8 is not used because of shortage of registers in the floating-point unit.

FFT	Arithmetic Operations			Storage References		
	Add	Mult	Total	Data	Twiddles	Total
Radix-2	$3Pp$	$2Pp$	$5Pp$	$4Pp$	Pp	$5Pp$
Radix-4	$\frac{22}{8}Pp$	$\frac{12}{8}Pp$	$\frac{17}{4}Pp$	$\frac{16}{8}Pp$	$\frac{6}{8}Pp$	$\frac{11}{4}Pp$
Radix-8	$\frac{66}{24}Pp$	$\frac{32}{24}Pp$	$\frac{49}{12}Pp$	$\frac{32}{24}Pp$	$\frac{14}{24}Pp$	$\frac{23}{12}Pp$

Table 2: Arithmetic and memory operations for radix-2, 4, and 8 FFTs.

4 Twiddle Factors

The total number of twiddle factors needed for a radix-2 FFT of size $P = 2^p$ is $\frac{P}{2} - 1$, and for a radix-R FFT it is $(R-1)\frac{P}{R}$. With precomputed twiddle factors stored in a distributed memory, it is important to minimize the need either for redundant storage of twiddle factors, or for communication of twiddle factors. Below we discuss the issues related to computation and allocation of twiddle factors for a radix-2 FFT on a distributed memory architecture. Computation and storage of twiddle factors for high radix FFT are presented in [12]. In the analysis below the order of the input data is assumed to be normal.

4.1 Decimation-in-time (DIT) FFT

For a radix-2 DIT FFT *in-place* algorithm, the exponents of the twiddle factors for stage q are $\omega_P^{(j_{p-q-1})(j_{p-q}j_{p-q+1}\ldots j_{p-1})2^{p-1-q}}$, $q \in [0, p-1]$, and $q = 0$ for the first stage. Note that the address is bit-reversed and shifted for the proper exponent. The twiddle factors for stage 0 are all ω_P^0. 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. For a P element transform performed on an N processor Boolean cube with *consecutive* data allocation, stages 1 through $n-1$ each requires one twiddle factor per stage and processor. All $\frac{P}{N}$ local elements have the same twiddle factor in a given processor. The last $p-n$ stages are local, and the maximum total number of twiddle factors required per processor is $\frac{P}{N} - 1$.

We will now prove the claims made above. In the consecutive data allocation the assignment of data indices to processors is $(j_{p-1}j_{p-2}\ldots j_{p-n}|\{j_{p-n-1}j_{p-n-2}\ldots j_0\})$, where $\{j_{p-n-1}j_{p-n-2}\ldots j_0\}$ denotes the set of values assumed by the bits within the braces. Clearly, for $q < n$ none of the data index bits mapped into local memory enters into the twiddle factor exponent. Hence, all local data elements have the same twiddle factors for the stages requiring inter-processor communication. The last stage, which is local, requires $\frac{P}{2N}$ twiddle factors, since the set of exponents are computed from $(\{(j_0)(j_1j_2\ldots j_{p-n-1}\}|j_{p-n}\ldots j_{p-1})$. In processor zero the twiddle factors for stages $n < q < p-1$ are always a subset of the twiddle factors for stage $p-1$, whereas in processor $N-1$ successive stages have unique twiddle factors. Hence, a maximum total of $\sum_{k=1}^{p-n}\frac{P}{2^kN} = \frac{P}{N} - 1$ different twiddle factors per processor is needed for the local stages, and the second part of the claim has been verified.

Allocating twiddle factor storage uniformly across all processors yields a total twiddle factor storage of $P+(n-2)N$, which for $P \gg N$ is about twice the storage required on a sequential or shared memory 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 *cyclic* allocation the $p-n$ most significant bits are mapped into local memory. The last n steps require communication, and different local elements often have different twiddle factors. For instance, consider processor $(011\ldots1)$, which in the cyclic allocation is assigned indices $(\{j_{p-1}j_{p-2}\ldots j_n\}|011\ldots1)$. The last stage requires twiddle factors with indices $(1\ldots110|\{j_nj_{n+1}\ldots j_{p-1}\})$, or $\frac{P}{N}$ twiddle factors. The second to last stage requires another $\frac{P}{N}$ twiddle factors in processor $(011\ldots1)$, since the index set $(1\ldots110\{j_n|j_{n+1}\ldots j_{p-1}\}0)$ is disjoint from the set for the last stage. In general, at least $(n-1)\frac{P}{N}$ twiddle factors are required in a processor for the cyclic data allocation and a DIT FFT.

4.2 Decimation-in-frequency (DIF) FFT

For an *in-place* radix-2 DIF FFT the twiddle factors required for the butterfly computation in stage q is $\omega_P^{(j_{p-q-1})\times(j_{p-q-2}j_{p-q-3}\ldots j_0)2^q}$. The exponent of the twiddle factor for a pair of complex elements in a butterfly computation is formed from address bits 0 through $p-q-2$ shifted left q steps with an end-off shift.

With a *consecutive* data allocation at least one processor needs $\frac{P}{N}$ twiddle factors for each of the first $n-1$ stages. The sets of twiddle factors for different stages are disjoint. For instance, consider the processor with address $(j_{p-1}j_{p-2}\ldots j_{p-n+1}j_{p-n}) = (11\ldots10)$. This processor contains the data indices $(11\ldots10|\{j_{p-n-1}j_{p-n-2}\ldots j_0\})$, with the consecutive allocation. Shifting this set of addresses left by one step yields a new set of addresses if $n > 2$. This property holds for $n-1$ left shifts, i.e., the first $n-1$ stages. For $n = 1$ it is easily seen that $\frac{P}{N}$ twiddle factors are needed, and for $n = 2$ processor (11) requires $3\frac{P}{2N}$ twiddles. Hence, at least $(n-1)\frac{P}{N}$ twiddles are required in some processor.

For a *cyclic* allocation the first n stages are local to a processor. The sets of twiddle factor indices required for the stages local to a processor are $(\{j_{p-2}j_{p-3}\ldots j_n\}|j_{n-1}\ldots j_0)$, $(\{j_{p-3}j_{p-4}\ldots j_n\}|j_{n-1}\ldots j_0)2$, ..., $(j_{n-1}\ldots j_0)2^{p-n-1}$, for stages 0, 1, etc., or a maximum total of $\sum_{m=1}^{p-n} 2^{p-n-m} = \frac{P}{N} - 1$ for any processor. 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 and processor. A maximum of $n-1$ twiddle factors per processor is 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 $\frac{P}{N} + n - 2$, the same as for a DIT FFT with consecutive data allocation.

4.3 Bit-reversed input

With the input in bit-reversed order the traversal of the bits in the address field is from the least significant bit to the most significant bit. The indices used for twiddle factor computation for the DIF FFT are obtained by using bit-reversed addresses, instead of addresses in normal order. Analogously, the DIT FFT uses addresses in normal order for the index computation, instead of bit-reversed addresses for normal order input.

With the input data in bit-reversed order the DIF FFT requires the least twiddle factor storage for the consecutive data allocation, while the DIT FFT requires the least storage for the cyclic data allocation. The preferred combinations of data allocation and FFT type are the opposite to those preferred with normal order input.

4.4 Reduced twiddle factor storage

For the consecutive data allocation, normal order input, and a radix-2 DIT 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, $\omega_P^{j+\frac{P}{4}} = \omega_P^j \cdot e^{-\frac{2\pi i}{P}\frac{P}{4}} = -i \cdot \omega_P^i$. Notice that multiplication by $-i$, which is associated with a 90-degree rotation, simply involves exchanging the real and imaginary parts of ω_P^i and negating the imaginary, which requires only one negation. Therefore, half of the twiddle factors can be obtained from the other half with no 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 $p-3, p-4, \ldots$ 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., $-\frac{1+i}{\sqrt{2}}$. However, these rotations do require complex arithmetic.

4.5 Summary

We conclude that the preferred combinations of data allocation, input order, and FFT type with respect to twiddle factor storage are:

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

FFT	Data alloc.	Twiddle index stage q	Max. number of twiddles per proc.
Normal input order			
DIT	consec.	$\{j_{p-q}j_{p-q+1}\dots j_{p-n-1}\}\|j_{p-n}\dots j_{p-1}2^{p-1-q}$	$\frac{P}{N}+n-2$
	cyclic	$j_{p-q}j_{p-q+1}\dots j_{n-1}\|\{j_n\dots j_{p-1}\}2^{p-1-q}$	$\geq (n-1)\frac{P}{N}$
DIF	consec.	$j_{p-q-2}j_{p-q-3}\dots j_{p-n}\|\{j_{p-n-1}\dots j_0\}2^q$	$\geq (n-1)\frac{P}{N}$
	cyclic	$\{j_{p-q-2}j_{p-q-3}\dots j_n\}\|j_{n-1}\dots j_0 2^q$	$\frac{P}{N}+n-2$
Bit-reverse input order			
DIT	consec.	$j_{q-1}j_{q-2}\dots j_{p-n}\|\{j_{p-n-1}\dots j_0\}2^{p-1-q}$	$\geq (n-1)\frac{P}{N}$
	cyclic	$\{j_{q-1}j_{q-2}\dots j_n\}\|j_{n-1}\dots j_0 2^{p-1-q}$	$\frac{P}{N}+n-2$
DIF	consec.	$j_{q+1}j_{q+2}\dots j_{n-1}\|\{j_n\dots j_{p-1}\}2^q$	$\frac{P}{N}+n-2$
	cyclic	$\{j_{q+1}j_{q+2}\dots j_{p-n-1}\}\|j_{p-n}\dots j_{p-1}2^q$	$\geq (n-1)\frac{P}{N}$

Table 3: Radix-2 twiddle factor storage for DIF and DIT FFT for different data allocation schemes and input orders.

- 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 Table 3. With 90-degree rotations performed "on-the-fly", the storage requirements for on-processor twiddles is reduced by a factor of two compared to what is stated in Table 3.

5 FFT on multi-dimensional arrays

Computing 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. Using the pipelined type of FFT algorithm described above, there is no need for data rearrangement between FFTs on different axes. Pipelining can be performed over all inter-processor dimensions being part of any FFT, as long as successive axes have no local components. If the axes have local components, such that the inter-processor dimensions are interleaved with local dimensions, then the communication pipeline must be broken for the local butterfly stages.

Each axis in a multi-dimensional FFT has its own set of twiddle factors. The twiddle factors for an axis is a subset of the twiddle factors for the longest axis. With axes of length $P_1 \times P_2 \times \dots P_k$ 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 $\frac{\sum_\ell P_\ell}{2}$.

6 Implementation

All compilers for the Connection Machine systems use the consecutive data allocation scheme. In order to minimize the twiddle factor storage with precomputed twiddle factors, a decimation-in-time FFT is used for data in normal input order, and a decimation-in-frequency FFT is used for bit-reversed input order. 90-degree rotations are performed “on-the-fly”. The inverse Discrete Fourier Transform uses conjugated twiddle factors, and requires no additional storage. The inter-processor communication stages are pipelined, as described above, but for multi-dimensional FFTs each axis is treated independently. In order to simplify the implementation no sharing of twiddle factors between axes takes place. Likewise, each stage of each axis has its own set of twiddle factors.

For the FFT stages requiring inter-processor communication the following implementation detail deserves to be mentioned. The Connection Machine systems CM-2 and CM-200 are of the SIMD variety. In the butterfly computations one processor in a pair performs a complex addition and the other a complex subtraction. By integrating the negation of one of the operands into the communication both operations can be performed concurrently with no measurable loss in efficiency.

For a data array axis with d bits assigned to the processor address field, d data elements are exchanged in each communication, except during the start-up and shut down of the communications pipeline. d butterfly computations can be performed after each communication. The butterfly computations belong to different stages, and require different twiddles. A local data element is updated d times in sequence. Then, it will not be updated until the local phase, if any. In order to reduce the number of loads and stores to local memory, the local data items are cached in the register set of the floating-point unit. Twiddle factors are (re)read from memory. The data caching scheme is used for up to 10 dimensions. For 11 dimensions there are insufficient registers in the floating-point unit, resulting in a performance loss, as can be seen from the timings in the next section.

For the local stages of the FFT a mix of radix-2, 4, and 8 kernels are used. Each kernel comes in several varieties depending upon the number of twiddle factors required, and whether or not the twiddles are cached in the register set. The radix-8 kernels yield the best performance, and for an FFT of a given size as many stages as possible are performed using these kernels. When the FFT cannot be computed using only these kernels, then one radix-2 or radix-4 stage is used. Details of the implementation are given in [12].

The scheduling of the radix-2 kernels attempts to minimize the loading of twiddle factors. Consider the DIF FFT in Figure 2. In the first stage all twiddle factors are different, but in the second stage each twiddle factor that is used is repeated twice. In the third stage each twiddle factor is repeated four times, etc. Hence, by performing the butterfly computations in suitable order a twiddle factor needs to be retrieved only once from memory. It is easily verified that the same property is true for a DIT FFT. Furthermore, the 90-degree rotation property is used to further reduce the number of twiddles needed. It is possible to load a number and perform a unary operation (in this case, negation) on it in the same cycle, which eliminates the need to perform any arithmetic whatsoever to use the rotated twiddle. The radix-4 and radix-8 kernels reload their twiddle factors each time, due to lack of sufficient registers. In these cases, however, the twiddle factor loading can be overlapped with computation efficiently.

The kernels for the local DIT and DIF FFT are very similar, but differ slightly in how the pipelines in the floating-point unit are organized. This difference result in a performance difference of up to about 7% in 32-bit precision, and up to about 11% in 64-bit precision, as seen from the tables in the next section.

The twiddles for all radix-2 stages are computed by the following algorithm.

1. 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 $p-2$.
2. bit-reverse the extracted word t.
3. 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. The computation is performed by all floating-point units concurrently. The computations are completely uniform. The twiddle factor computation for radix-4 and 8 stages is analogous, but somewhat more complex [12]. In our implementation there is a separate table for each stage. Each table is organized such that a sequence of twiddle factors are accessed with a stride of one. Within each table the twiddle factors are stored in bit-reversed order. The same set of tables are used for both the DIF and DIT FFT, and for both forward and inverse FFTs.

7 Performance measurements

The performance measurements below have been made on a Connection Machine system CM-200 with 2048 64-bit floating-point units. All performance data refer to a complex-to-complex FFT, CCFFT, implemented as described above, and included as part of the Connection Machine Scientific Software Library version 3.0. The order of the input data is normal in all cases unless otherwise specified. We provide data for both ordered and unordered transforms. All floating-point rates are based on $5N \log_2 N$ arithmetic operations for a transform of size N.

The performance of the local kernels for different sizes is given in Table 4 and Figure 4. The peaks in Figure 4 correspond to data sizes for which only radix 8 FFT's are used. The performance for 64-bit precision is about 75-80% of the performance for 32-bit precision; the difference is entirely due to the fact that the data path between each floating-point unit and its memory is 32-bits wide. Data paths internal to the floating-point unit are 64-bits wide. The performance of the DIT kernels is 90 - 95% of the DIF kernel performance for most sizes; the difference here is entirely due to implementation details of the pipelines. Table 5 gives performance data for ordered local transforms. Large ordered transforms are about

Axis	Time, msec				Gflops/s			
	32-bit prec.		64-bit prec.		32-bit prec.		64-bit prec.	
length	DIT	DIF	DIT	DIF	DIT	DIF	DIT	DIF
32	0.18	0.17	0.23	0.21	9.077	9.476	7.211	7.896
64	0.35	0.33	0.43	0.39	11.293	12.032	9.170	10.171
128	0.84	0.80	1.12	1.07	10.958	11.396	8.157	8.567
256	1.94	1.82	2.48	2.24	10.825	11.504	8.449	9.378
512	3.92	3.66	4.92	4.42	12.051	12.887	9.599	10.669
1k	9.07	8.69	12.05	11.34	11.565	12.065	8.700	9.249
2k	20.66	19.44	26.60	24.01	11.167	11.865	8.671	9.609
4k	41.79	39.29	53.00	47.75	12.043	12.809	9.496	10.541
8k	93.65	89.69	123.92	115.60	11.644	12.159	8.800	9.434
16k	207.86	196.21	268.28	242.17	11.300	11.971	8.755	9.699
32k	419.82	395.98	535.17	482.54	11.989	12.711	9.405	10.431
64k	920.42	881.03	1213.82	1126.12	11.666	12.187	8.846	9.535
128k	2005.73	1897.08	2737.99	2593.42	11.376	12.027	8.333	8.798
256k	4355.31	4167.26			11.094	11.595		

Table 4: Performance data for local, unordered, CCFFT on a 2048 processor CM-200.

10% slower than unordered transforms; for transforms of size 1024 the ordered transform is about 20% slower than the unordered transform.

The performance for the inter-processor communication stages is considerably less than for the local stages. The communication time is constant regardless of the number of processors over which the data set is distributed. With communication in d dimensions, d butterflies can be computed for each communication. The construction of the arithmetic pipelines for the DIT and DIF CCFFT for the inter-processor stages is almost identical, except for the case where 11 processor dimensions are used. In this case the DIF pipeline is about 5% less efficient in 32-bit precision, and 2.5% less efficient in 64-bit precision. The efficiency of the arithmetic pipeline is constant for d in the range 2 – 10. With a communication time that is almost independent of d, it follows that the overall efficiency of the inter-processor butterfly stages increases with the number of processors over which the data set is distributed. The peak inter-processor data motion rate exceeds 7 Gbytes/s for 11 processor dimensions, as is apparent from Table 6 and Figure 5. The floating-point rates are scaled to a 2048 processor CM-200. For instance, in calculating the floating-point rates for FFT on a data set spread across 512 processors, it is assumed that 4 such computations are performed concurrently on distinct data sets and processors.

The performance for one-dimensional DIF CCFFT computations for a few different sizes of the data set is given in Table 7 and Figure 6. The performance difference between the DIT and DIF versions is about 2 – 3% in 32-bit precision, and about 1% in 64-bit precision. The difference is due to the difference in efficiency of the local kernels. The performance for large one-dimensional FFTs is largely determined by the inter-processor butterfly stages. These stages smooth out the variations in the performance of the local kernels, and a monotonic

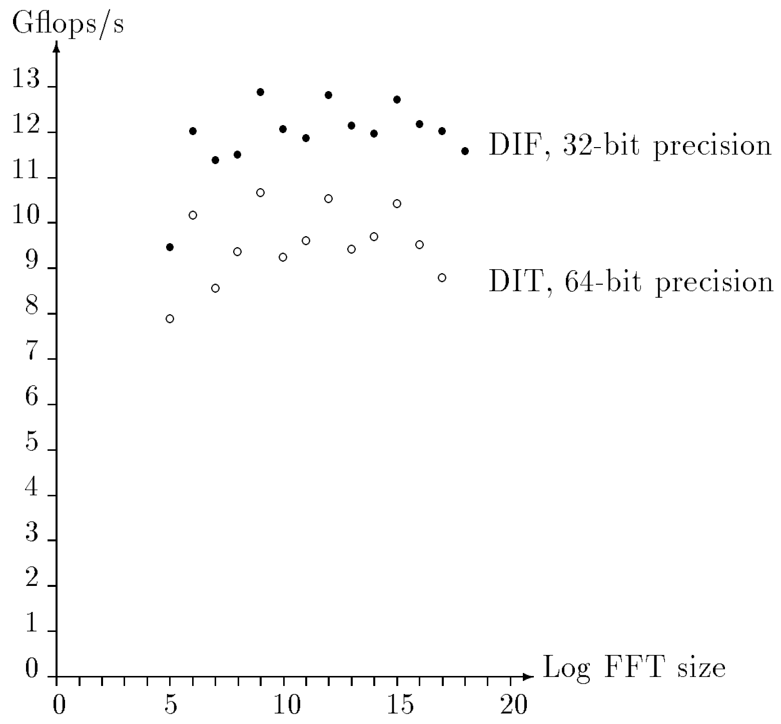

Figure 4: The performance of local, unordered, DIF CCFFT on a 2048 processor CM-200.

Axis	Time, msec				Gflops/s			
	32-bit prec.		64-bit prec.		32-bit prec.		64-bit prec.	
length	DIT	DIF	DIT	DIF	DIT	DIF	DIT	DIF
32	0.23	0.23	0.33	0.32	7.062	7.155	4.965	5.201
64	0.45	0.43	0.63	0.59	8.680	9.123	6.192	6.631
128	1.04	1.01	1.53	1.49	8.814	9.075	5.977	6.178
256	2.35	2.24	3.32	3.08	8.916	9.354	6.309	6.805
512	4.74	4.49	6.59	6.10	9.961	10.516	7.162	7.742
1024	10.74	10.37	15.45	14.74	9.765	10.116	6.786	7.112
2048	24.04	22.84	33.47	30.87	9.595	10.101	6.892	7.473
4096	48.65	46.16	66.83	61.58	10.346	10.903	7.531	8.173
8192	107.41	103.48	151.64	143.33	10.152	10.538	7.191	7.608
16384	235.51	223.91	323.89	297.81	9.973	10.490	7.252	7.887
32768	475.21	451.45	646.46	593.88	10.591	11.149	7.786	8.475
65536	1031.33	992.09	1436.59	1349.00	10.411	10.823	7.474	7.960
131072	2227.67	2119.29	3183.99	3039.30	10.243	10.766	7.166	7.507
262144	4801.31	4615.54			10.064	10.469		

Table 5: Performance data for local, ordered, CCFFT on a 2048 processor CM-200.

Axis	Time, msec				Gflops/s			
	32-bit prec.		64-bit prec.		32-bit prec.		64-bit prec.	
length	DIT	DIF	DIT	DIF	DIT	DIF	DIT	DIF
2	248.57	248.59	428.38	430.17	0.337	0.337	0.196	0.195
4	251.08	251.13	435.45	437.00	0.668	0.668	0.385	0.384
8	256.44	256.44	445.30	447.04	0.981	0.981	0.565	0.563
16	262.95	262.90	459.00	460.64	1.276	1.276	0.731	0.728
32	269.86	269.87	472.19	474.17	1.554	1.554	0.888	0.885
64	276.63	276.64	485.77	487.56	1.819	1.819	1.036	1.032
128	283.32	283.32	500.68	502.50	2.073	2.073	1.173	1.169
256	290.09	290.03	513.06	514.70	2.313	2.314	1.308	1.304
512	296.85	296.85	525.59	528.48	2.543	2.543	1.436	1.429
1024	303.62	303.65	540.31	542.03	2.763	2.763	1.553	1.548
2048	328.89	346.91	570.25	589.93	2.806	2.660	1.618	1.564

Table 6: Performance data for pure inter-processor unordered CCFFT on a 2048 processor CM-200, 8192 elements/processor.

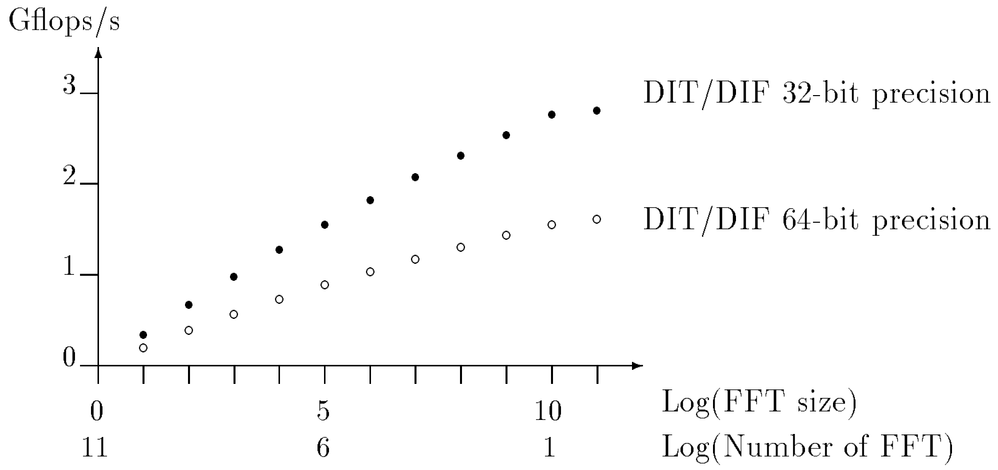

Figure 5: The floating-point rate for pure inter-processor, unordered, CCFFT on a 2048 processor CM-200, 8192 elements/processor.

Axis	Time, msec				Gflops/s			
	32-bit prec.		64-bit prec.		32-bit prec.		64-bit prec.	
length	DIT	DIF	DIT	DIF	DIT	DIF	DIT	DIF
1M	24.98	25.84	41.16	41.89	4.198	4.058	2.547	2.503
2M	50.65	52.52	84.01	88.65	4.347	4.193	2.621	2.484
4M	103.28	106.57	172.15	173.95	4.467	4.329	2.680	2.652
8M	206.51	213.01	339.77	342.37	4.671	4.529	2.839	2.818
16M	422.53	436.59	691.90	703.12	4.765	4.611	2.910	2.863
32M	865.08	889.48	1405.81	1416.91	4.848	4.715	2.984	2.960
64M	1733.71	1781.97	2803.52	2829.55	5.032	4.896	3.112	3.083
128M	3547.64	3652.48	5749.73	5819.32	5.107	4.961	3.151	3.114
256M	7259.64	7439.44	11809.21	11981.89	5.177	5.052	3.182	3.136
512M	14862.54	15251.33			5.238	5.104		

Table 7: Performance data for one-dimensional, unordered, CCFFT on a 2048 processor CM-200.

Machine	1M points				16M points			
	32-bit prec.		64-bit prec.		32-bit prec.		64-bit prec.	
size	Time(msec)	Speedup	Time(msec)	Speedup	Time(msec)	Speedup	Time(msec)	Speedup
128	373.01	1.00	616.90	1.00	6424.98	1.00	10606.91	1.00
256	184.51	2.02	305.27	2.02	3198.41	2.00	5237.77	2.03
512	94.00	3.97	156.57	3.94	1581.96	4.06	2593.47	4.09
1024	47.10	7.92	82.50	7.48	802.97	8.00	1325.68	8.00
2048	25.84	14.44	41.89	14.73	436.59	14.72	703.12	15.09

Table 8: Speedup of one-dimensional CCFFT as a function of the number of CM-200 processors.

increase in performance is observed with increased transform size.

Some sample speedup figures for the computations as a function of machine size are given in Table 8. The speedup is almost perfect for both the FFT's of one million and 16 million points. The loss of efficiency for the largest configuration is due to the decrease in efficiency for the inter-processor communication stages, when 11 inter-processor dimensions are involved in the FFT. The marginally better than perfect speedup that is achieved in some cases is due to the efficiency variations of the local FFT stages. For instance, a 1 million point data set has 8k points per processor for a 128 processor configuration, but 4k points per processor in a 256 processor configuration. The latter uses only radix-8 kernels, which are the most efficient.

Timings for two- and three-dimensional CCFFT are given in Table 9, and shown in Figure 7. The significant increase in performance for the two-dimensional CCFFT between the 1024×1024 array and the 2048×2048 array is due to one of the axis being local to a processor for the larger array. The subsequent minor decrease in performance for the next larger array

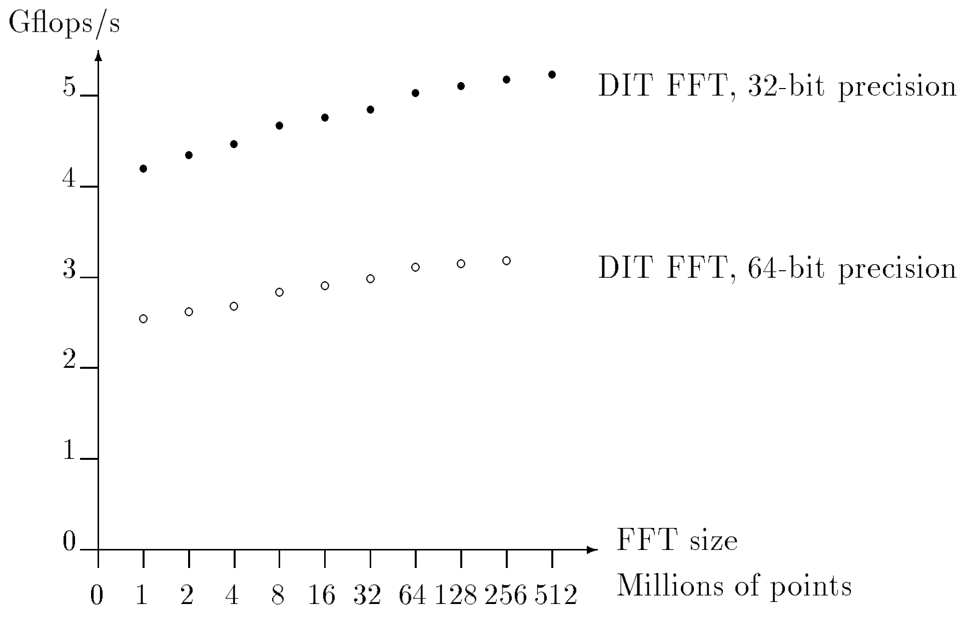

Figure 6: The execution rate for one-dimensional, unordered, DIT CCFFT on a 2048 processor CM-200.

is due the fact that the axis distributed over all processors also has a local component of length two. This part of the axis requires a radix-2 kernel, which is less efficient than the radix-4, and the radix-8 kernels normally used. For reference, performance data for ordered two and three-dimensional transforms are given in Table 10. The execution time increases by 50 - 100% for our examples, considerably more than for entirely local transforms.

7.1 Optimizing the configuration of the address space

With pipelining of communications the number of element transfers in sequence is $\frac{P}{N}+d-1$, where $\frac{P}{N}$ is the number of elements per processor, and d the number of inter-processor dimensions over which an axis subject to transformation is spread. Hence, the number of element transfers in sequence is approximately independent of the number of axis d, except if $d = 0$ in which case no communication is required. The number of arithmetic operations are clearly independent of the data layout. Hence, if load balance can be achieved by allocating all axes subject to transformation to local memory, then the best possible performance for a given array is achieved. If such an allocation is not feasible, either because of poor load balance, or because the data set is too large to fit in local memory, then optimizing performance represents a trade-off between the efficiencies of the local kernels, and the inter-processor stages. The variation in efficiency is small. We consider the one and multi-dimensional cases in some detail below.

Axis	Time, msec				Gflops/s			
	32-bit prec.		64-bit prec.		32-bit prec.		64-bit prec.	
length	DIT	DIF	DIT	DIF	DIT	DIF	DIT	DIF
256×256	2.8	2.8	4.7	4.7	1.862	1.856	1.118	1.115
512×512	10.6	10.8	17.6	17.8	2.227	2.181	1.340	1.325
1024×1024	43.0	43.0	71.1	73.1	2.439	2.439	1.476	1.435
2048×2048	103.3	106.7	171.5	173.8	4.464	4.326	2.691	2.655
4096×4096	487.1	501.5	760.6	770.6	4.133	4.014	2.647	2.613
8192×8192	1868.1	1921.8	2986.0	3022.7	4.670	4.539	2.922	2.886
16384×16384	7470.4	7648.2	11846.3	11916.6	5.031	4.914	3.172	3.154
$64 \times 64 \times 64$	10.6	10.6	17.6	17.6	2.221	2.228	1.342	1.342
$128 \times 128 \times 128$	79.3	78.9	134.0	133.7	2.777	2.791	1.643	1.647
$256 \times 256 \times 256$	724.2	721.8	1180.4	1176.4	2.780	2.789	1.706	1.711
$512 \times 512 \times 512$	5608.4	5554.7	9227.5	9128.8	3.231	3.262	1.964	1.985

Table 9: Performance data for two and three-dimensional, unordered, CCFFT on a 2048 processor CM-200.

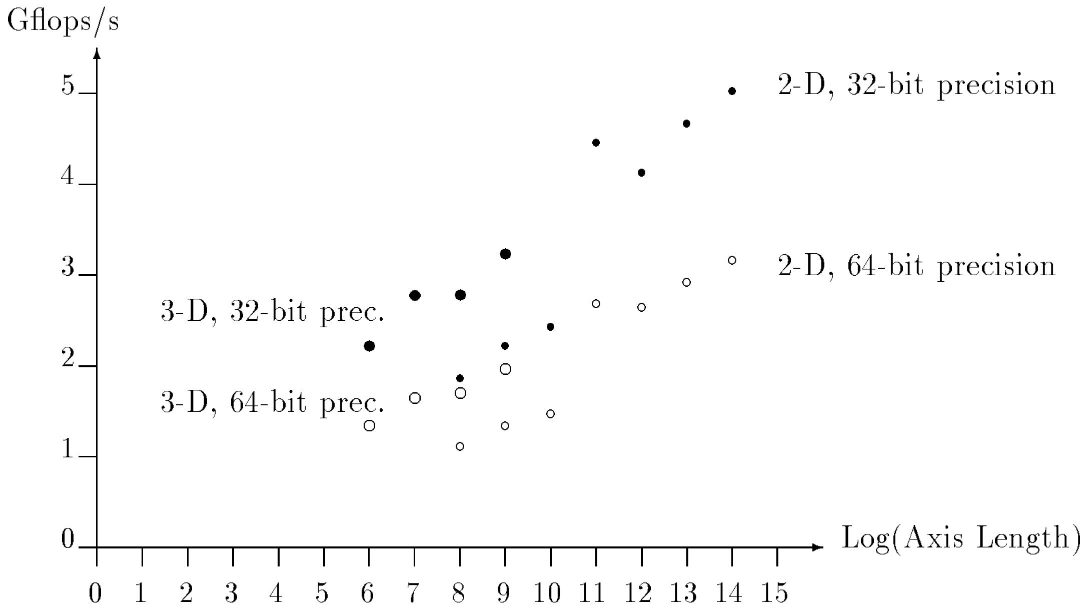

Figure 7: The execution rate for two and three-dimensional, unordered, DIT CCFFT on a 2048 processor CM–200.

Axis	Time, msec				Gflops/s			
	32-bit prec.		64-bit prec.		32-bit prec.		64-bit prec.	
length	DIT	DIF	DIT	DIF	DIT	DIF	DIT	DIF
256×256	4.62	4.63	8.05	8.06	1.134	1.132	0.652	0.650
512×512	16.77	16.98	29.66	29.77	1.407	1.389	0.795	0.793
1024×1024	68.55	68.71	122.73	124.39	1.530	1.526	0.854	0.843
2048×2048	183.36	186.66	329.43	331.76	2.516	2.472	1.401	1.391
4096×4096	907.03	923.01	1598.08	1609.40	2.220	2.181	1.260	1.251
8192×8192	3393.68	3448.63	6049.18	6091.06	2.571	2.530	1.442	1.432
16384×16384	13715.35	13894.09	24409.33	24475.00	2.740	2.705	1.540	1.535
$64 \times 64 \times 64$	18.89	18.89	32.10	32.11	1.249	1.249	0.735	0.735
$128 \times 128 \times 128$	123.10	122.71	221.53	221.27	1.789	1.794	0.994	0.995
$256 \times 256 \times 256$	1101.56	1100.11	1930.29	1927.64	1.828	1.830	1.043	1.044
$512 \times 512 \times 512$	8302.94	8251.76	14655.03	14560.63	2.182	2.196	1.236	1.244

Table 10: Performance data for two and three-dimensional, ordered, CCFFT on a 2048 processor CM-200.

7.2 One-dimensional FFT

The execution time for a single one-dimensional FFT is minimized by spreading the data over as many floating-point units as possible. For multiple one-dimensional FFTs we distinguish between two cases. If the number of instances of the one-dimensional FFT is at least half the number of processors, and the length of the axis on which the transform is performed fits in local memory, then maximum performance is attained if the data is layed out with the FFT axis local to a processor. The other case is where there are fewer instances, or the axis subject to transformation will not fit in local memory. In this case the performance is fairly independent of the layout. The optimum layout is mostly determined by the variation in the efficiency of the local FFT kernels as a function of the local axis length. The efficiency of the arithmetic for the interprocessor communication stages is constant, except for $d = 1$ and $d = 11$, for which cases the efficiency is lower. The communication time is constant with d. Table 11 shows performance data for one-dimensional FFT on a 4096×4096 data array for various data layouts. The peak performance is achieved for the axis subject to transformation entirely local. The performance for the FFT axis spread over more than one processor is fairly constant, with a noticeable drop off for the FFT axis spread over all processors. The performance variation for the FFT axis spread over 2 to 1024 processors is about 10% in both 32-bit and 64-bit precision. For $d = 11$ the performance decreases by another 10%.

7.3 Multi-dimensional FFT

Optimal efficiency is attained by maximizing the number of axes that has no non-local component. The performance variation once an axis is distributed across processors is minor,

Local length of axis 0	Time, msec				Gflops/s			
	32-bit prec.		64-bit prec.		32-bit prec.		64-bit prec.	
	DIT	DIF	DIT	DIF	DIT	DIF	DIT	DIF
4096	83.94	78.66	106.60	95.73	11.992	12.796	9.443	10.515
2048	331.28	326.38	535.21	526.45	3.039	3.084	1.881	1.912
1024	323.69	320.79	532.24	528.16	3.110	3.138	1.891	1.906
512	318.95	314.95	524.15	517.99	3.156	3.196	1.920	1.943
256	324.90	321.28	538.93	532.72	3.098	3.133	1.868	1.890
128	323.21	321.36	543.97	542.66	3.114	3.132	1.851	1.855
64	320.53	317.79	540.21	536.74	3.140	3.168	1.863	1.875
32	328.40	326.18	556.65	553.25	3.065	3.086	1.808	1.819
16	331.51	330.89	567.28	568.38	3.037	3.042	1.774	1.771
8	328.72	327.03	561.13	561.48	3.062	3.078	1.794	1.793
4	342.73	342.12	587.73	587.97	2.937	2.942	1.713	1.712
2	398.74	419.52	641.37	665.16	2.525	2.399	1.569	1.513

Table 11: Performance of unordered CCFFT along one axis of a 4096×4096 array on a 2048 processor CM-200.

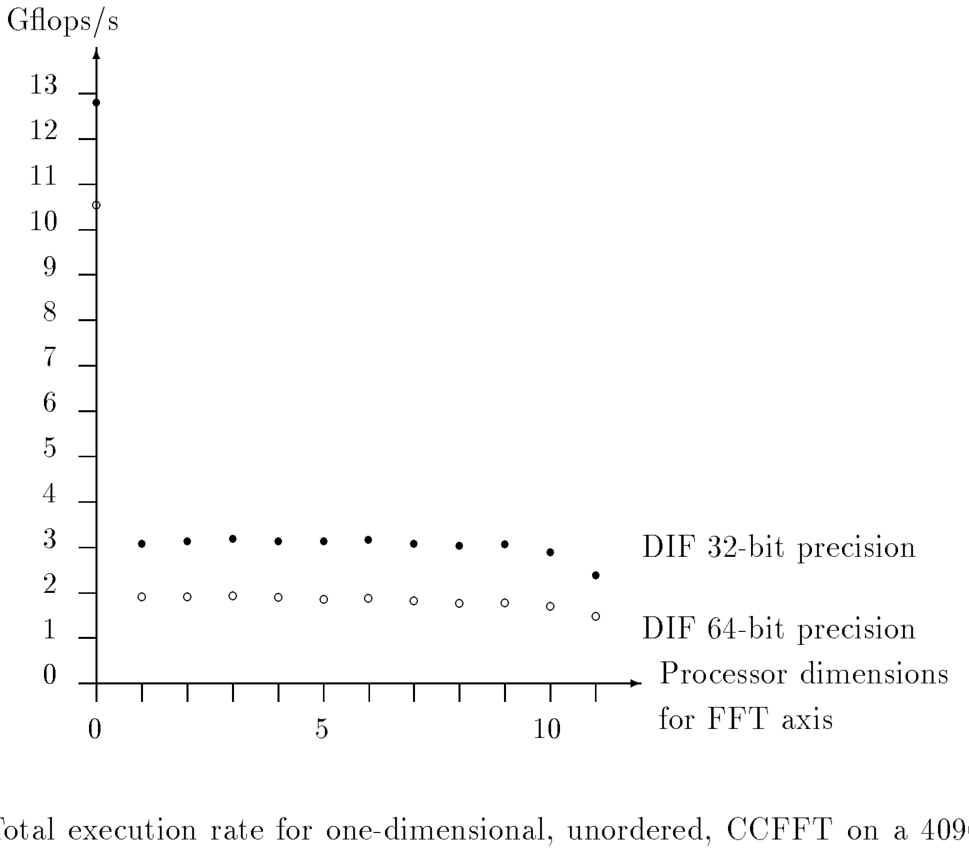

Figure 8: Total execution rate for one-dimensional, unordered, CCFFT on a 4096×4096 array as a function of the configuration of a 2048 processor CM-200.

Local axis length	Time, msec				Gflops/s			
	32-bit prec.		64-bit prec.		32-bit prec.		64-bit prec.	
	DIT	DIF	DIT	DIF	DIT	DIF	DIT	DIF
4096	485.49	500.34	757.98	770.68	4.147	4.024	2.656	2.612
2048	681.96	675.88	1132.61	1123.06	2.952	2.979	1.778	1.793
1024	654.05	649.90	1097.09	1091.76	3.078	3.098	1.835	1.844
512	653.05	648.55	1097.75	1092.43	3.083	3.104	1.834	1.843
256	654.92	649.30	1099.37	1089.70	3.074	3.101	1.831	1.848
128	645.21	641.13	1087.37	1082.66	3.120	3.140	1.852	1.860
64	645.33	641.32	1087.34	1082.70	3.120	3.139	1.852	1.859
32	654.82	649.23	1099.20	1089.65	3.075	3.101	1.832	1.848
16	652.59	647.95	1094.04	1088.97	3.085	3.107	1.840	1.849
8	654.74	650.95	1096.36	1093.12	3.075	3.093	1.836	1.842
4	679.62	674.27	1130.14	1123.14	2.962	2.986	1.781	1.793
2	487.10	501.52	760.58	770.56	4.133	4.014	2.647	2.613

Table 12: Performance of a two-dimensional unordered CCFFT on a 4096×4096 array computed on a 2048 processor CM-200.

as can be seen in Table 12. For a two-dimensional FFT of shape 4096×4096 the worst performance once an axis is distributed across processors is at most 5% below the peak in 32-bit precision, and at most 3.5% below peak in 64-bit precision. The difference between a distributed axis, and a local axis is about 20% in 32-bit precision and close to 30% in 64-bit precision.

Smaller and high-dimensional FFTs cannot always be layed out in such a way as to have only one axis with any non-local component. Figures 10 and 11 present various two and three dimensional FFTs layed out optimally on different size machines. Notice the discontinuity as the machine size exceeds the axis length in both the two and three dimensional transforms. The performance degradation is caused by an axis changing from being entirely local to becoming distributed over at least two processors for the larger machine size. Other performance fluctuations are minor. For example, the point at which the FFT axis length equals the machine size runs relatively slightly faster than the other transforms in the series, and the point at which the FFT axis length is exactly twice the machine size runs relatively slightly slower. The reason is that in the former case the looping is exceptionally simple. In the latter case one local stage is needed for the axis with a non-local extent. This local stage is performed with a radix-2 kernel, which is less efficient than the radix-4 and radix-8 kernels used normally. Very careful inspection will show other similar effects due to minor differences in the kernels.

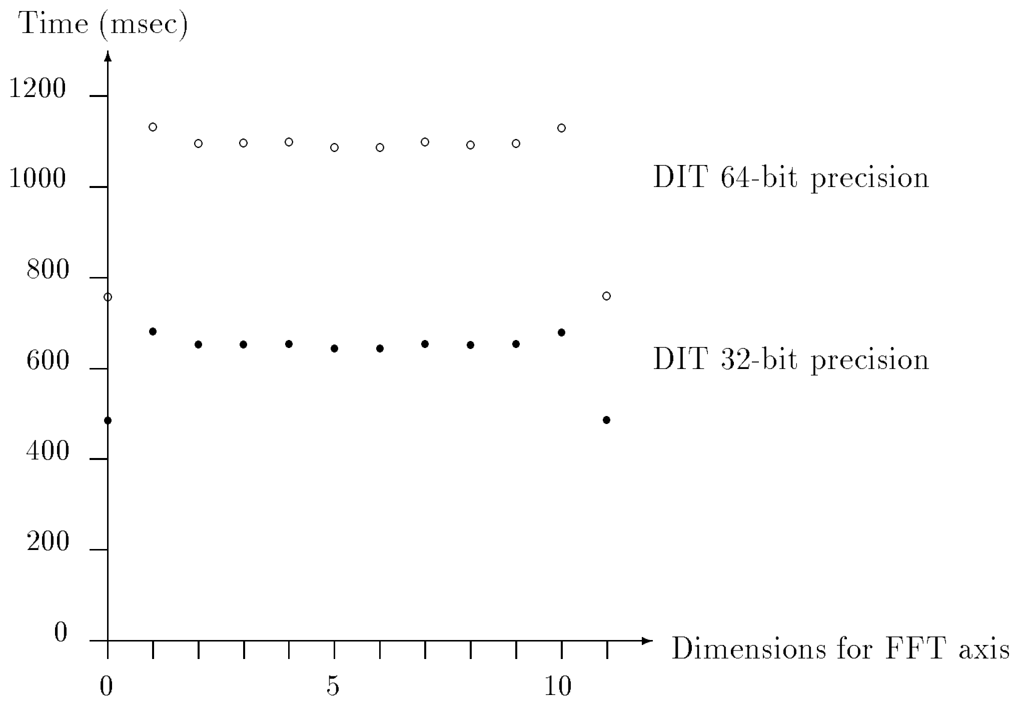

Figure 9: Total execution time for a two-dimensional unordered CCFFT on a 4096 × 4096 array as a function of the configuration of 2048 CM-200 processors.

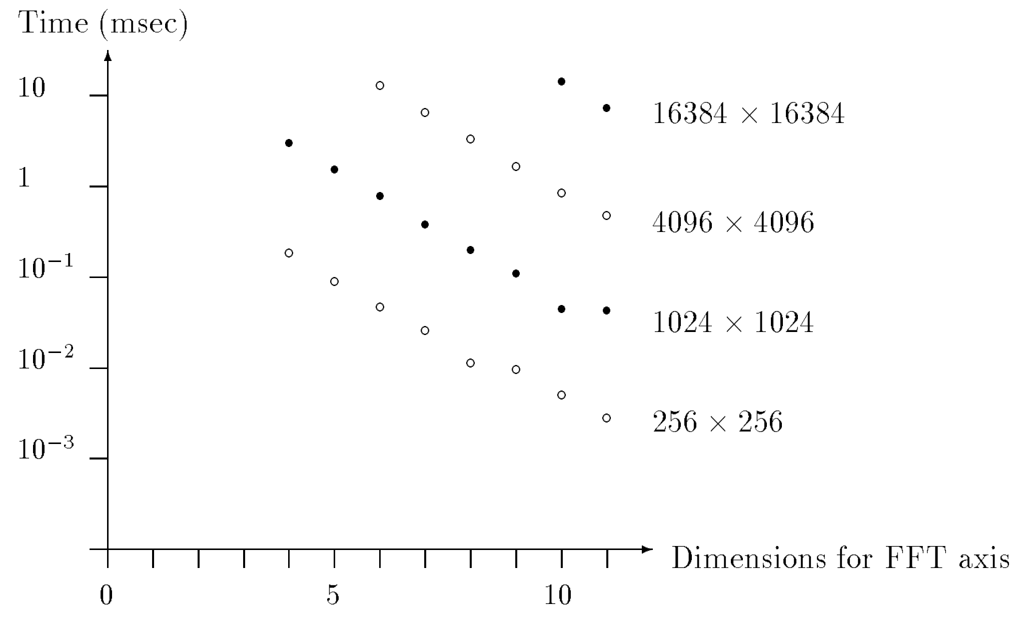

Figure 10: Total execution time for two-dimensional unordered CCFFT on CM-200 configurations ranging from 16 to 2048 processors.

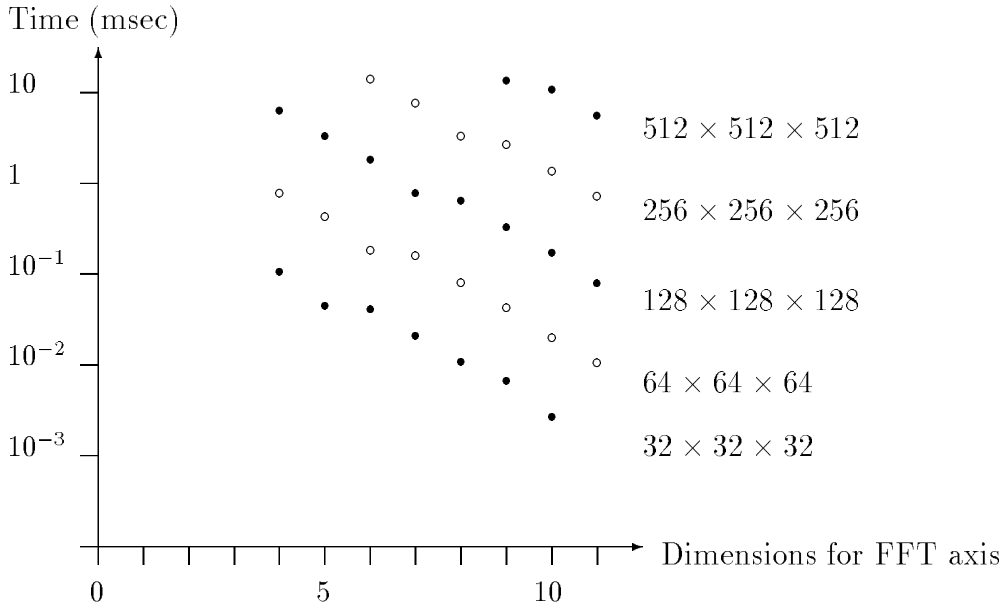

Figure 11: Total execution time for three-dimensional unordered CCFFT on CM-200 configurations ranging from 16 to 2048 processors.

8 Summary and Discussion

We have presented a multi-processor FFT adapted for efficient use of both the communication system and the memory bandwidth. We have also shown that the data layout and twiddle factor usage demands that both decimation-in-time FFT and decimation-in-frequency FFT are supported to accommodate data in either normal or bit-reversed order, when the twiddle factors are precomputed. Supporting only one type would increase the twiddle factor storage by a factor of $\log_2 N$ for N processors. Multi-dimensional FFTs are performed with no explicit data rearrangement.

The peak performance for the entirely local complex-to-complex FFT is about 12.9 Gflops/s in 32-bit precision and about 10.7 Gflops/s in 64-bit precision. The peak performance for one-dimensional FFT in the range one million to half a billion points is in the range 4.2 - 5.2 Gflops/s in 32-bit precision, and in the range 2.5 - 3.2 Gflops/s in 64-bit precision. In 32-bit precision, the peak rates for two-dimensional complex-to-complex FFTs are in the range 1.8 - 5.0 Gflops/s, and for cubic three-dimensional complex-to-complex FFTs the range is 2.2 - 3.2 Gflops/s. In 64-bit precision the corresponding numbers are 1.1 - 3.2 Gflops/s and 1.3 - 2.0 Gflops/s, respectively. The peak data motion rate between processors is in excess of 7 Gbytes/s. The performance for the corresponding ordered transforms is about 10 - 20% less for transforms on data entirely local to each processor, and 2/3 - 1/2 for transforms on data sets spread over several processors.

Despite the fact that cubic three-dimensional FFTs do not perform as well on large machines as two-dimensional FFTs, certain three-dimensional shapes perform very well, particularly on smaller machines. This is due to the fact that communication pipelining is done only within a given array axis, but not across axes.

Once an axis subject to transformation is distributed across processors, the performance is quite insensitive to the actual configuration of the set of processors. The performance difference as a function of the number of processors over which an instance of the array axis subject to transformation is distributed is in the 5 - 10% range.

The load balance for the FFT stages requiring inter-processor communication can be improved. Only one processor in a pair computing a butterfly performs a useful complex multiplication. A partitioning type of algorithm, in which all stages are performed locally, as described in [11, 15, 16], results in perfect arithmetic load balance. In many cases, however, the efficiency for such an algorithm on the Connection Machine systems will not be higher. It may even be lower, as we will explain below.

The partitioning type of FFT also has the advantage that only half the data set is communicated in each inter-processor communication stage, as opposed to all of the data in the direct pipelined algorithm used for the Connection Machine implementation. However, with the consecutive data allocation at least one of the processor dimensions must be used twice for this type of FFT [11, 15, 16]. With concurrent communication on all channels the number of element transfers in sequence is the same as for the direct pipelined algorithm used in our implementation. Hence, for normal order input and consecutive data allocation the partitioning type FFT does not offer any reduction in the communication requirements. Indeed, because of low level features of the Connection Machine architecture, each communication in such an FFT implementation would require longer time, and the savings in time due to improved arithmetic load balance would be nullified in most cases. However, for cyclic data allocation the partitioning type FFT would offer a moderate improvement, since in this case no inter-processor dimension needs to be used more than once for an unordered FFT [11].

Acknowledgment

The authors express their sincere thanks to the anonymous referees whose comments, suggestions and careful reading of the manuscript helped to greatly improve the presentation.

References

[1] James C. Cooley and J.W. Tukey. An algorithm for the machine computation of complex fourier series. *Math. Comp*, 19:291–301, 1965.

[2] Alan Edelman. Optimal matrix transposition and bit-reversal on hypercubes: All-to-all personalized communication. *Journal of Parallel and Distributed Computing*, 11(4):328–331, 1991.

[3] Geoffrey C. Fox, Mark A. Johnson, Gregory A. Lyzenga, Steve W. Otto, John K. Salmon, and David W. Walker. *Solving Problems on Concurrent Processors*. Prentice-Hall, 1988.

[4] W. Morven Gentleman and G. Sande. Fast Fourier transforms – for fun and profit. In *Proceedings – Fall Joint Computer Conference, 1966*, pages 563–578. AFIPS, 1966.

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

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

[7] S. Lennart Johnsson and Ching-Tien Ho. Matrix transposition on Boolean n-cube configured ensemble architectures. *SIAM J. Matrix Anal. Appl.*, 9(3):419–454, July 1988.

[8] 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.

[9] S. Lennart Johnsson and Ching-Tien Ho. Maximizing channel utilization for all-to-all personalized communication on Boolean cubes. In *The Sixth Distributed Memory Computing Conference*, pages 299–304. IEEE Computer Society Press, 1991.

[10] S. Lennart Johnsson and Ching-Tien Ho. Boolean cube emulation of butterfly networks encoded by Gray code. *Journal of Parallel and Distributed Computing*, 20(3):261–279, 1994. Department of Computer Science, Yale University, Technical Report, YALEU/DCS/RR-764, February, 1990.

[11] 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.

[12] S. Lennart Johnsson, Michel Jacquemin, and Ching-Tien Ho. High radix FFT on Boolean cube networks. Technical Report Department of Computer Science, Yale University, Technical Report YALEU/DCS/RR-751, November 1989, Thinking Machines Corp., November 1989. To appear in the Journal of Computational Physics.

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

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

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

[16] Charles Tong and Paul N. Swarztrauber. Ordered Fast Fourier transforms on a masively parallel hypercube multiprocessor. *Journal of Parallel and Distributed Computing*, 12(1):50–59, May 1991.