FFT Implementations on nCUBE Multiprocessor

Akhilesh Kumar and Laxmi N. Bhuyan
Department of Computer Science
Texas A&M University
College Station, TX 77843-3112
Email: {akhil,bhuyan}@cs.tamu.edu

Abstract

Three parallel FFT algorithms have been implemented on nCUBE multiprocessor and their performance have been analyzed. We have also developed a communication and computation model for the nCUBE system based on measurements. This model is then used to predict the execution time and communication time for the three FFT algorithms. These predictions have been verified with the execution of these algorithms on the nCUBE system.

1 Introduction

Discrete Fourier Transform (DFT) is a basic tool for many engineering and scientific applications. Some of its areas of application are: digital signal processing (spectrum analysis, digital filtering), digital image processing, speech processing, computer tomography, coding theory, weather prediction, oil exploration, statistics, multiplication of very large integers, polynomial multiplication, etc. The Fast Fourier Transform (FFT), proposed by Cooley and Tukey [5], is a method to compute DFT using fewer computation steps than required by direct computation of DFT. As the area of computer applications has broadened, the quantity of data to be transformed has greatly increased and parallel computer architectures have been proposed to fulfill the future computational requirements. This has also led to the parallelization of FFT algorithms to achieve a fast transformation by exploiting the features of advanced computer architectures. These algorithms basically depend on the novel schemes to store intermediate results such that they can be accessed efficiently as required in the next step of computation. Pease's [12] work on the parallelization of the FFT algorithm has been the basis of most of other works in this area. In vector computers, these algorithms exploit the features of memory organization such as the amount of interleaving, to access data efficiently by minimizing memory bank conflicts, and the resources such as the number of registers to achieve maximal utilization. Swarztrauber [13] provides a good overview of the FFT algorithms suitable for vector computers. Based on the variations of Stockham autosort FFT algorithms proposed in [14], Bailey [2] proposed an improved algorithm for vector computers by avoiding strides divisible by large powers of two in accessing data, thus reducing memory bank conflicts. Many variations of FFT algorithm for parallel computers have also been proposed. In parallel computers, the computation is divided among multiple processors and attention is given to communicate the intermediate results among processors for further computation. All the FFT algorithms for parallel computers attempt to minimize the interprocessor communication overhead. This is achieved by intelligent data distribution among processors and properly scheduling the data transfer over the links in the interconnection network connecting the processors. Swarztrauber [14], Jeon and Reeves [7], and Jesshope [8] have proposed FFT algorithms suitable for hypercube multiprocessor, multicluster computer, and array computers, respectively.

There has been several studies regarding the performance of various FFT algorithms on different multiprocessor architectures [1, 4, 6, 8, 11]. Most of these studies predict the performance based on an analytical model, without giving much emphasis on the actual implementation and experimental verification, except in [6]. In [6], experimental performance of one FFT algorithm has been studied on a shared-memory type multiprocessor workstation based on shared bus, taking into account various factors such as snoop protocol, operating system and synchronization scheme. There seems to be scope for investigating the experimental performance of various FFT algorithms on different multiprocessor architectures to verify the performance claims for these algorithms and to gain insight into the effects of some architectural features. Here, we have implemented three FFT algorithms on an nCUBE system [10] which is a message-passing multicomputer with hypercube interconnection. We have also developed a computation and communication model for the nCUBE system and have verified the analytical performance of the FFT algorithms based on this model with the actual performance.

The organization of this report is as follows. The following section introduces the FFT algorithm and describes its three different variants that have been implemented and analyzed during in this report. Section 3 describes the architecture of the nCUBE system. Section 4 deals with various issues involved in the design and implementation of parallel FFT algorithms. In section 5, we propose a computation and communication model for the nCUBE system which will be used to predict the performance of the FFT algorithms. Section 6 presents the analysis of the three FFT algorithms on the nCUBE multiprocessor based on the model proposed in section 5. Section 7 presents the analytical and experimental results obtained from the implementation of the FFT algorithms on the nCUBE multiprocessor. In the conclusion, we compare the performance of the three FFT algorithms and discuss its implications.

2 The FFT Algorithm

The discrete Fourier transform (DFT) of N point array $x_0, x_1, \ldots, x_{N-1}$ is defined as

$$y_j = \sum_{k=0}^{N-1} x_k w_N^{jk} \qquad \text{for } j = 0, 1, \ldots, N-1$$

where $w_N = e^{-2\pi i/N}$, $i = \sqrt{-1}$.

The time complexity of computing the DFT directly using the above equation is $T(N) = O(N^2)$.

Fast Fourier transform (FFT) is a technique to compute DFT recursively by decomposing the transform of a large data sequence into transforms of two smaller data sequences. The computation of y_j can be decomposed as follows:

$$\begin{aligned} y_j &= \sum_{m=0}^{\frac{N}{2}-1} x_{2m} w_N^{2mj} + \sum_{m=0}^{\frac{N}{2}-1} x_{2m+1} w_N^{(2m+1)j} \\ &= \underbrace{\sum_{m=0}^{\frac{N}{2}-1} x_{2m} w_{N/2}^{mj}}_{\frac{N}{2}\text{ point DFT}} + w_N^j \underbrace{\sum_{m=0}^{\frac{N}{2}-1} x_{2m+1} w_{N/2}^{mj}}_{\frac{N}{2}\text{ point DFT}} \end{aligned}$$

which shows that the DFT of N points can be computed by combining the results of two DFTs on the sequences of length $N/2$. FFT uses this decomposition recursively to compute DFT. Thus, the time complexity of FFT can be derived as $T(N) = 2T(N/2) + CN = O(N \log N)$. This shows that FFT is much more efficient for computing DFT than the direct computation. Cooley and Tukey [5] first proposed the FFT algorithm based on this recursive decomposition. A data flow diagram of Cooley-Tukey algorithm is shown in Fig 1 for $N = 32$ without showing the initial data partitioning stages resulting from the recursive decomposition. It shows the input data in bit-reversed order after the partitioning stages and then shows the stage by stage computation resulting from combination of 2 smaller DFT sequences to compute a larger DFT sequence as per the above expression.

As indicated earlier, there are several variants of FFT algorithm depending on the data storage pattern during the stages of computation. We shall overview some of the main variants in the following subsection.

2.1 Variants of parallel FFT algorithm

The three variants of FFT algorithm considered in this report are

- Cooley-Tukey FFT
- Jeon-Reeves FFT
- Hypercube FFT

Here, we shall discuss each of these algorithms.

2.1.1 Cooley-Tukey FFT

This algorithm has been described earlier in this section in the context of computing DFT by recursive decomposition. The data flow diagram is shown in Fig 1 for 32 data points ($N = 32$). Mapping of this algorithm on multiprocessors is straight forward. The bit-reversed data is equally divided among available

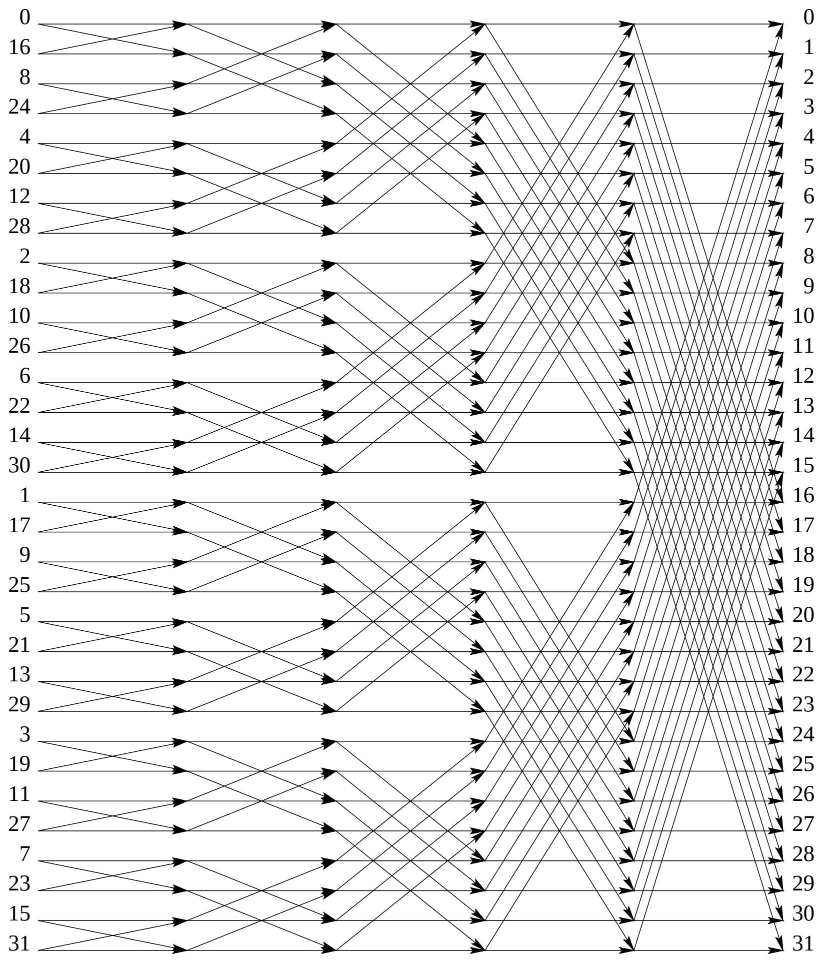

Figure 1: Cooley-Tukey FFT data flow diagram for 32 points.

number of processors. Let the number of processors be P. For simplicity, we shall assume that N and P are some integral power of 2 and $n = \log_2 N$ and $p = \log_2 P$. Each processor has N/P data elements. All the processors perform first $(n-p)$ stages of FFT on its own data, concurrently. After $(n-p)$ stages, every processor need to get data from some other processor during the rest p stages. In every communication step, all the data from a processor is transmitted to some other processor. The communication pattern in the Cooley-Tukey FFT algorithm is very suitable for a binary cube, since the communication will always be among neighboring processors. The disadvantages of this algorithm are that it requires bit-reversed data as input and the amount of communication overhead is large.

2.1.2 Jeon-Reeves FFT

Pease [12] proposed an FFT algorithm suitable for parallel processing on an SIMD machine. The idea was to permute (using the perfect shuffle) intermediate results at each stage such that the two data elements involved in the next arithmetic operations are always fed to the same processing unit. While excellent for an SIMD machine, Pease's implementation is not very efficient for multicomputers because each permutation would require a large amount of communication not always between neighboring processors. Jeon and Reeves proposed a modification of the Pease FFT algorithm by reducing the number of permutation stages, and thus reducing the number of communication stages. A signal flow graph of this algorithm is shown in Fig 2. Instead of permuting the data after every stage as proposed by Pease, in the modified algorithm the data is permuted after every $(n-p)$ stages. Thus the amount of communication required is reduced considerably. In most of the cases where $n \geq 2p$, only one permutation stage is required. The problems with this algorithm are that it also requires bit-reversed data as input and that the communication pattern is not as regular as in case of Cooley-Tukey FFT and is difficult to map on a binary cube. For $n \geq 2p$, every processor needs to communicate to all other processor in the cube, which is very costly.

2.1.3 Hypercube FFT

Swarztrauber [14] proposed an FFT algorithm specially designed for hypercube multiprocessors. This algorithm does not require bit-reversed data as input and the communication is always between neighboring processing elements on the hypercube. Fig 3 shows the data flow diagram for hypercube FFT of 16 points on 2 processors. Though the number of stages requiring communication is larger than the Cooley-Tukey algorithm, but the bit-reversal stage may offset this advantage for Cooley-Tukey algorithm.

3 nCUBE Multiprocessor System

nCUBE is a message-passing multicomputer with hypercube interconnection between the processing nodes. Each processing node is a full fledged computer with its own CPU and memory. The processing units interact by explicitly sending messages from one processor to another processor through the links in the interconnection network. The system consists of 64 nodes arranged as a 6-dimensional binary cube. Smaller dimensional subcubes can be allocated from the system for a particular task. Each processing node in a subcube is identified

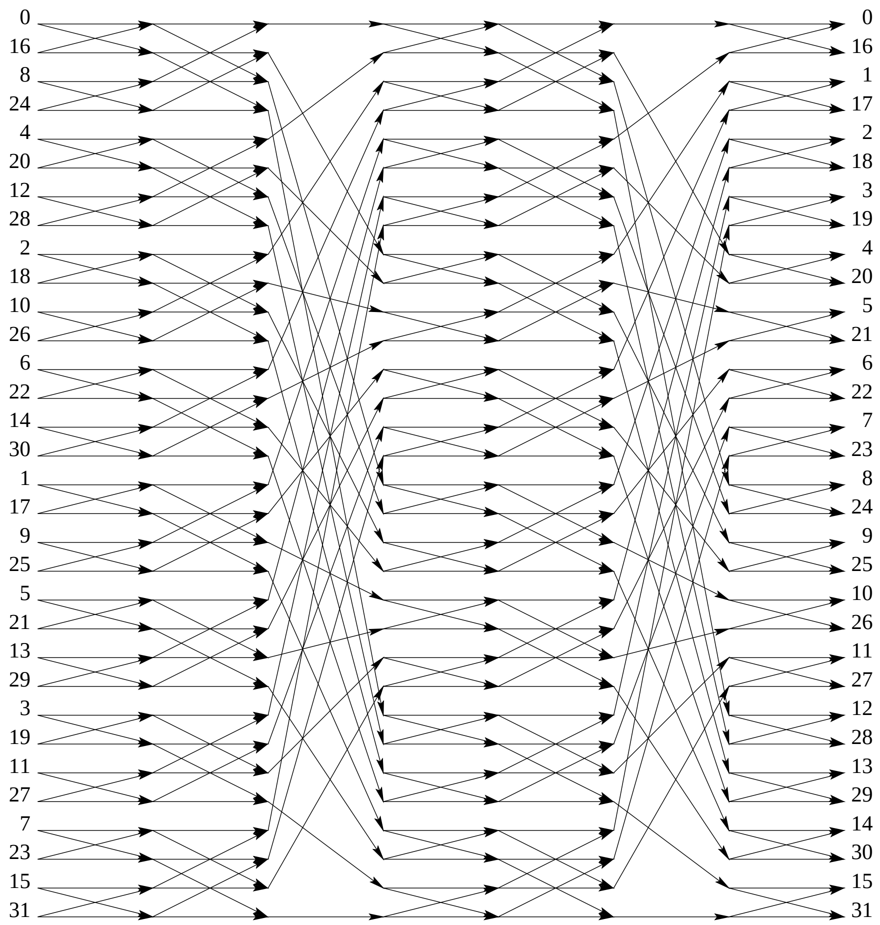

Figure 2: Jeon-Reeves FFT data flow diagram for 32 points on 8 processors.

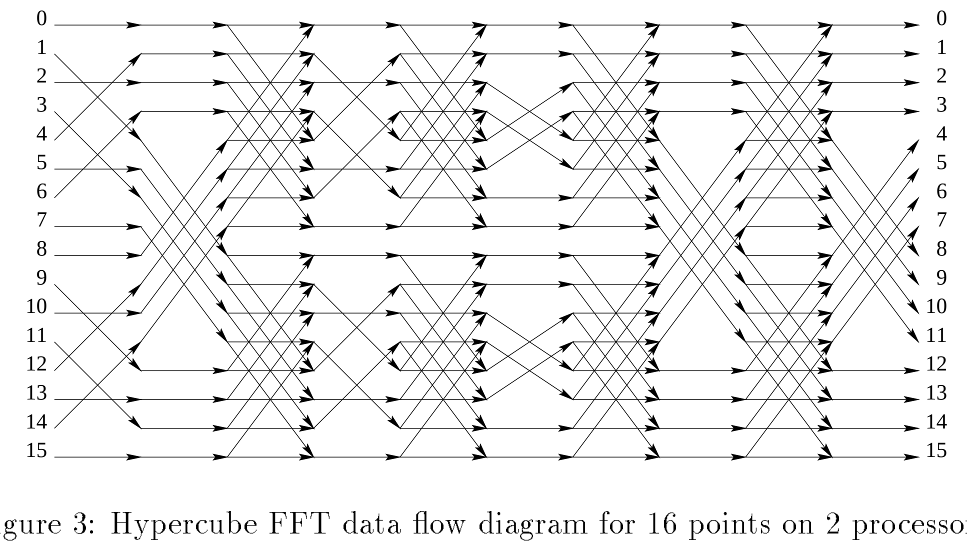

Figure 3: Hypercube FFT data flow diagram for 16 points on 2 processors.

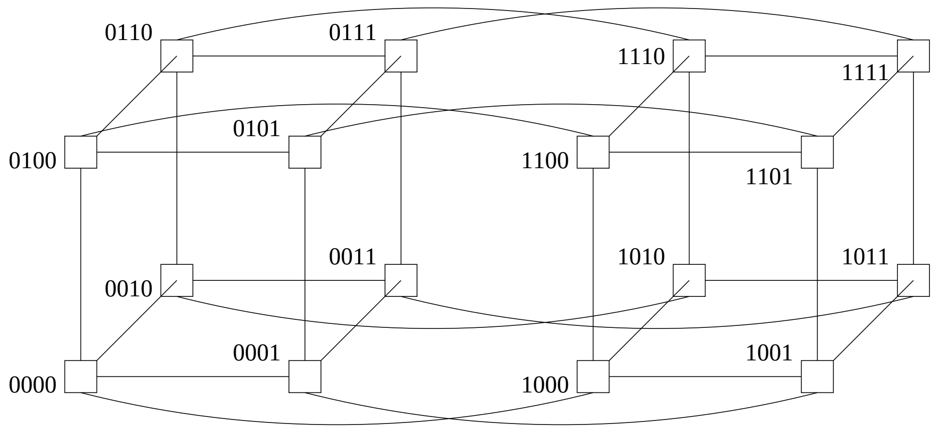

Figure 4: Hypercube of dimension 4.

by its identity number, as shown in Fig 4 for a subcube of dimension 4. The processing nodes in nCUBE system consists of a custom built 64-bit processor and 2 to 4 Mbytes of memory. The processor includes a Network Communication Unit (NCU) to take care of incoming and outgoing messages without CPU intervention [10]. The processor uses cut-through routing hardware, which accelerates message passing. A message is not stored in the memories of nodes along the routing path, and software is only executed at the initiating and receiving nodes. The communication software is interrupt driven, which reduces the load on the CPU. Automatic cut-through routing allows the messages to pass through the intermediate nodes without interrupting the intermediate node CPUs, thus improving the performance of the system. A message can be visualized as a chain of packets being relayed from processor to processor. Messages are automatically broken into discreet packets of 32 data bits during transmission and automatically reconstructed upon receipt. The first packet contains an address that is used to determine the next link in the path. As the address packet passes through a link, it is automatically reserved for the following data packets. The network communication units (NCU) at each processor can buffer up to 2 packets on each incoming link. The NCU will request another packet as soon as it has space for it, until the End of Transmission (EOT) packet arrives. As an EOT packet passes through a link, it causes the processor to release it for use by other messages. Reception of a message is acknowledged by sending an acknowledgment back to the source processor. Each link can transfer data with a peak data rate of 2.2 Mbytes/sec.

4 Issues in Parallel FFT Algorithm

There are several issues involved in parallelization of FFT algorithms. These issues arise mainly because of architectural features of multiprocessor systems. Some of the architectural features affecting the design and performance of parallel FFT algorithms are:

1. **Shared Memory vs Message Passing:** In shared memory multiprocessors, all the memory locations are accessible to all the processors. Thus there is no interprocessor communication involved, but the algorithm must take care of the memory access conflicts that may arise when multiple processors try to access the same memory module. These access conflicts can be resolved by putting the sensitive data in critical regions or by putting synchronization points in the processes to ensure proper event ordering. Such a situation does not arise in message passing multicomputers, since processors access only its local memory. The processors interact through explicitly transmitting the messages from one processor to another. Here, the main concern is to avoid conflicts in the interconnection network while transmitting messages.

2. **Memory Organization:** In shared memory multiprocessors, the main memory is usually interleaved to facilitate concurrent access. If the data is not accessed consecutively in a stage of computation, which is the case in some FFT algorithms, it may lead to hot spot in memory organization since some of the modules may be accessed heavily at some time while the other modules may be free. The algorithm must take this into account and try to avoid such hot spots. The algorithms designed for vector computers take care of this effect and can be used in shared memory multiprocessors.

3. **Cache Parameters:** The effect of cache parameters like the size of cache, the block size, and the cache coherence protocol may be significant on the performance of an algorithm in shared memory multiprocessors, and needs to be investigated. In case of FFT algorithms, the block size may have considerable effect on the performance if all the data in the block is not utilized in a stage and multiple processors operate on the data belonging to one block, which may lead to large number of cache misses and invalidations. The algorithm must be suitably modified to avoid such situations. We have developed an FFT algorithm for cache based shared memory multiprocessors [9].

4. **Interprocessor Communication:** Cut-through routing is being employed in second generation message passing multicomputers which im-

proves the performance over store-and-forward technique used in earlier systems. But the performance of cut-through routing deteriorates if there is conflict in routing the messages through the network. The effect of cut-through routing needs to be investigated for FFT algorithms on the binary cube, and suitable modifications needs to be looked into to take advantage of cut-through routing without overloading the network.

In this report we are mainly concerned with investigating the issues related to message passing multiprocessors, particularly a multiprocessor configured as hypercube, to develop a model for such a system, to implement various FFT algorithms on this system, and to study its performance. The following sections address these issues one by one.

5 Modeling the nCUBE System

Here we propose a simple model for the nCUBE system taking into account both computation and communication. Various system parameters were measured for use in this model.

To measure the time taken by computational operations like addition, multiplication, etc., and also to measure the overhead due to programming language constructs like 'for' loop, a pseudocode was written. This code was used to measure the time for addition, multiplication, etc., on some dummy variables initialized arbitrarily. These operations were done for a large number of times, typically 100000, by putting it in a 'for' loop and the time taken was measured. The time taken by an empty for loop was also measured for the same number of iterations, and was subtracted from the earlier measurement to determine the time taken by arithmetic operations. The second measurement was also used to determine the time taken by each iteration of the 'for' loop which involves incrementing the loop index and checking the limiting condition. The results obtained by these measurements are as follows:

Integer Addition Time = t_{ai} = 0.1515μs.
Integer Multiplication Time = t_{mi} = 0.2016μs.
Floating Point Addition Time = t_{af} = 0.4525μs.
Floating Point Multiplication Time = t_{mf} = 0.4014μs.
Overhead due to each iteration of 'for' loop = t_i = 0.9031μs.
Overhead due to 'if' statement = 0.4500μs.

These parameters have been used later to estimate the time for FFT algorithm implemented on the nCUBE.

The measurement of parameters for modeling the communication behavior on the nCUBE system was also done using pseudocode to simulate various communication pattern and by measuring the time taken by such communication. This measurement was performed for various system sizes and message length. Some typical measurements are shown in Table 1 and Table 2.

TABLE - 1

Cube Dimension = 3.

All times in millisec.

Bytes	One to one	All to all	Broadcast from 0
16	0.173720	1.522000	0.361000
32	0.183745	1.562000	0.365000
64	0.202775	1.651000	0.411000
128	0.233915	1.974000	0.508000
256	0.313495	2.423000	0.676000
512	0.456465	3.447000	1.039000
1024	0.756490	5.479000	1.760000
2048	1.333915	9.559000	3.204000
4096	2.501190	17.716000	6.092000
8192	4.839785	34.103000	11.868000
16384	9.508040	66.768000	23.421000

TABLE - 2

Number of bytes transferred = 4096.

All times in millisec.

Dimension	One to one	All to all	Broadcast from 0
1	2.495915	2.556000	2.312000
2	2.501730	7.631000	4.204000
3	2.501190	17.716000	6.092000
4	2.507355	37.999000	7.985000
5	2.503770	79.735000	9.888000
6	2.510035	162.271000	11.771000

Based on these measurements the communication time on nCUBE can be modeled as

$$T_{comm} = \alpha + k\beta \tag{1}$$

where,

α = Setup time = 165μs.

β = Transmission time per byte = 0.571μs.

k = Number of bytes to be transmitted.

This is a very simple model and is applicable for one to one communication between the processors. This model is also applicable for the case when the interprocessor communication is such that there is no contention for the channels on the binary cube network. It should be noted that because of the cut-through routing technique used in the nCUBE system, the distance between the nodes in the network does not have any significant effect on the communication time as long as network is contention free. This is evident from the times in the second column of Table 2 which lists the time taken to transfer data between farthest processors in the hypercube.

In case of broadcasting on the nCUBE system, it takes about $\log_2 P$ times one to one communication time. This is so because the broadcasting is done by spawning a $\log_2 P$ level tree in the hypercube. For all to all communication, the time taken is proportional to the number of processor in the cube. A fact which is not apparent from the tables above and should be noted here is that the broadcast time from any node other than 0 in the cube takes longer time. This is because broadcast from any node other than 0 is done first by sending the data to node 0 and then broadcasting it from there. This was confirmed by the nCUBE Corp. and they hope to eliminate this discrepancy in the later versions of their system software.

6 Analysis of FFT Algorithms on nCUBE System

To estimate the computation and communication time for the FFT algorithms on the nCUBE system based on the model proposed in the previous section, let,

$N = 2^n$ = Number of data elements

$P = 2^p$ = Number of processors

b = Number of bytes per data element

c = Number of bytes per integer data

t_a = Complex addition time = $2t_{af} = 0.905\mu$s.

t_m = Complex multiplication time = $4t_{mf} + 2t_{af} = 2.5106\mu$s.

$t_B = t_m + 2t_a$ = Time to compute one butterfly in FFT.

$q = n \bmod (n - p)$

6.1 Cooley - Tukey FFT Algorithm

A data flow diagram of Cooley-Tukey algorithm [13] is shown in Fig 1 for $N = 32$. Mapping of this algorithm on multiprocessors is straight forward. The bit-reversed data is equally divided among available number of processors. Each processor has N/P data elements. All the processors perform first $(n-p)$ stages of FFT on its own data, concurrently. After $(n-p)$ stages, every processor need to get data from some other processor during the rest of p stages. In every communication step, all the data from a processor is transmitted to some other processor. The communication pattern in the Cooley-Tukey FFT algorithm is very suitable for a binary cube, since the communication will always be among neighboring processors.

Now, the computation time for N-point Cooley-Tukey FFT algorithm using P processors as obtained from counting the major operations in the code, can be written as

$$T_{comp} = \frac{N}{2P}(n-p)t_B + \frac{N}{P}p(t_m + t_a) + C_1\frac{N}{P}n \qquad (2)$$

The first term on the right hand side represents the computation time for the first $(n-p)$ stages of computation. Each processor computes $N/2P$ butterflies. The second term takes into account the last p stages of computation. During these stages each processor computes only half portion of each butterfly. The final term on the right hand side takes care of time spent on other computations like bit - reversal, index computation, 'for' loop, etc., which is proportional to nN/P. Value of C_1 was found to be about 8μs.

The communication time for Cooley-Tukey FFT on nCUBE is estimated using the model proposed in the previous section.

$$T_{comm} = p\left(\alpha + \frac{N}{P}b\beta\right) \qquad (3)$$

This is because during last p stages of computation, all the data from each of the processors needs to be transferred to some other processor. All these data transfers are always to a neighboring processor on a hypercube structure because of the communication pattern in Cooley-Tukey FFT. Thus, in all the p stages each processor transfers N/P data to its neighboring processor.

The expression for communication time presented above does not take into account the communication required to bring the data in bit-reverse order as is required for the Cooley-Tukey FFT algorithm. The time taken by the communication required to achieve this reordering of data from the initial distribution (assuming that data is distributed such that processor i has data from index

$i * N/P$ to $(i+1) * N/P - 1)$ is

$$\begin{aligned} T_{rev} &= (P-1)\left(\alpha + \frac{N}{P^2} b\beta\right), \text{ if } \quad p \leq n/2 \\ &= \frac{N}{P}(\alpha + b\beta), \text{ otherwise} \end{aligned} \quad (4)$$

In the first case where $p \leq n/2$, i.e., $P \leq \sqrt{N}$, each processor needs to communicate to all the other $(P-1)$ processors sending N/P^2 data to each of them. This communication pattern is called total exchange and takes $\Theta(P)$ time to complete on the hypercube [3]. In the second case the data elements per processor is less than the number of processors, so, each processor communicates to at the most N/P other processors, sending one data to each which can be properly scheduled to avoid conflicts on the network, thus the time as shown above.

Therefore, the total communication time for the Cooley-Tukey FFT algorithm on the hypercube is estimated from eqn. (3) and (4) as

$$T_{comm} = p\left(\alpha + \frac{N}{P} b\beta\right) + T_{rev} \quad (5)$$

and the total execution time of N point Cooley-Tukey FFT algorithm on a hypercube using P processors can be estimated from eqn. (2) and (5) as

$$T_{ctfft} = T_{comp} + T_{comm} \quad (6)$$

6.2 Jeon - Reeves FFT Algorithm

The data flow diagram of Jeon-Reeves FFT algorithm [7] is shown in Fig 2 for $N = 32$ and $P = 4$. This algorithm was also mapped on the hypercube similar to Cooley-Tukey algorithm by equally diving the bit-reversed data among all processors. Each processor has N/P data elements and computes $N/2P$ butterflies in every stage of FFT. Thus the computation time of N-point Jeon-Reeves FFT algorithm using P processors is given by

$$T_{comp} = \frac{N}{2P} n t_B + C_2 \frac{N}{P} n \quad (7)$$

The final term on the right hand side takes care of time spent on other computations like bit - reversal, index computation, 'for' loop, etc., which is proportional to nN/P. Value of C_2 was found to be about 9μs.

All the processors perform first $(n-p)$ stages of FFT on its own data, concurrently. After $(n-p)$ stages, interprocessor communication is required. The interprocessor communication is done differently from the Cooley-Tukey algorithm such that for another $(n-p)$ stages no communication is required.

The communication pattern is a p-inverse shuffle, i.e., the destination index is obtained by right rotating binary representation of source index p times. This communication is repeated after every $(n-p)$ stages, so there are $\lfloor n/(n-p) \rfloor$ such communication stages. The communication pattern of p-inverse shuffle is similar to that of the bit-reversal and takes same amount of time. So, the total communication time for the Jeon-Reeves algorithm on hypercube can be estimated as

$$T_{comm} = \left(\left\lfloor \frac{n}{n-p} \right\rfloor + 1 \right) T_{rev} + A \tag{8}$$

where, $A = 2^q\{\alpha + (N/(P2^q))b\beta\}$, if $q \neq 0$; else $A = 0$. One extra bit-reversal time has been added to take into account the initial bit-reversal time. The term A takes care of the time required to bring the data in proper order after n stages of computation since if n is not a multiple of $(n-p)$, the data will not be in proper order at the end.

Thus, the total execution time of N point Jeon-Reeves FFT algorithm on a hypercube using P processors can be estimated from eqn. (7) and (8) as

$$T_{pfft} = T_{comp} + T_{comm} \tag{9}$$

6.3 Hypercube FFT Algorithm

The data flow diagram of hypercube FFT [14] is shown in Fig 3 for $N = 16$ and $P = 2$. No bit-reversal is required for this algorithm and the normally ordered input data is equally divided among the processors. The expression for computation time of this algorithm is similar to that of Jeon-Reeves algorithm as given below.

$$T_{comp} = \frac{N}{2P} n t_B + C_3 \frac{N}{P} n \tag{10}$$

The final term on the right hand side takes care of time spent on other computations such as index computation, 'for' loop, etc., which is proportional to nN/P. Value of C_3 was found to be about 8μs.

The communication in this algorithm is always between neighboring processors and every time half of the data in the processor is exchanged. The number of communication stages, as shown in [14], is $(2p+1)$ if $p \leq \lceil n/2 \rceil$, or $(\lceil n/2 \rceil + p + 1)$ if $p > \lceil n/2 \rceil$. We also keep a record of the original data index which facilitates fetching multiplication factor from a precomputed table in the butterfly computation. This index also needs to be moved along with the data. Taking all this into account, the communication time for hypercube

FFT algorithm can be expressed as

$$
\begin{aligned}
T_{comm} &= (2p+1)\left(2\alpha + \frac{N}{2P}(b+c)\beta\right), \text{ if } p \leq \lceil n/2 \rceil \\
&= (\lceil n/2 \rceil + p + 1)\left(2\alpha + \frac{N}{2P}(b+c)\beta\right), \text{ if } p > \lceil n/2 \rceil \qquad (11)
\end{aligned}
$$

Thus, the total execution time of N point hypercube FFT algorithm on a hypercube using P processors can be estimated from eqn. (10) and (11) as

$$T_{pfft} = T_{comp} + T_{comm} \qquad (12)$$

7 Experimental Results

The total execution time and interprocessor communication time for the three algorithms discussed in the previous section are shown in Fig 5 to Fig 10 with respect to logarithm of the number of processors. These times were obtained by implementing all the three FFT algorithms on nCUBE system and measuring the respective time for 1024 - point FFT on different hypercube sizes.

Fig 5 shows the total execution time of Cooley-Tukey FFT algorithm on 1024 points with respect to dimension of the hypercube. As the dimension increases, the execution time keeps on decreasing as expected. Fig 6 shows the communication time for the same algorithm. As we can see that the communication time is maximum for the case where $p = n/2$. This is mainly because of the communication required to rearrange the data in bit-reversed order before the computation. From the first expression of the communication time due to bit-reversal in Eqn (4) we see that the setup time increases linearly with the number of processors while the transmission time decreases. We transmit data in packets of N/P^2 during bit reversal and as we approach $p = n/2$ the packet size decreases to 1. Since the setup time is much larger than the time required to transmit a data item of 32 bits, the communication time shoots up near $p = n/2$. A similar characteristics is observed in case of Jeon-Reeves algorithm as well because of the same reason, as shown in Fig 8. The total execution time of Jeon-Reevs algorithm is shown in Fig 7 which shows that it decreases as the number of processors increase except near $p = n/2$, where a big increase in communication time offsets the gain in computation time.

The total execution time and communication time for the hypercube FFT algorithm is shown in Fig 9 and Fig 10, respectively. The communication characteristic of this algorithm seems awkward but one important thing which must be noticed here is that the actual communication time obtained from the measurements are considerably less than the expected communication time

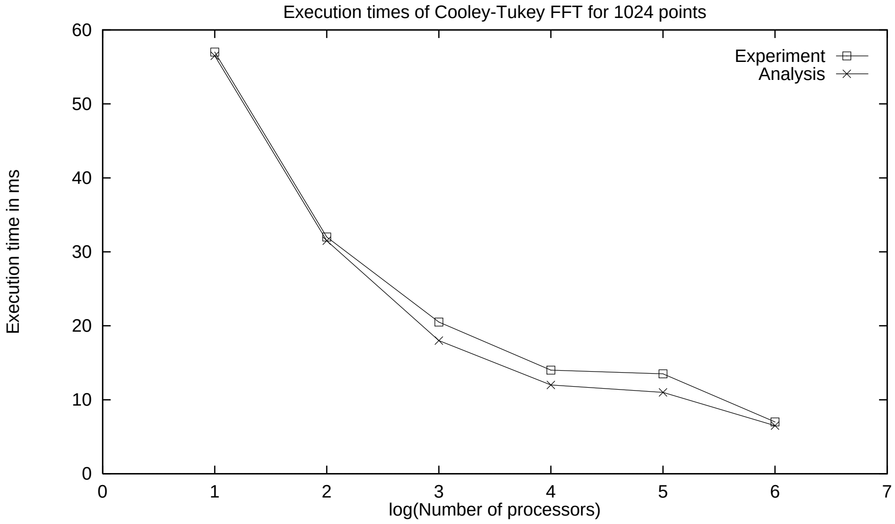

Figure 5: Execution time of Cooley-Tukey FFT for different number of processors.

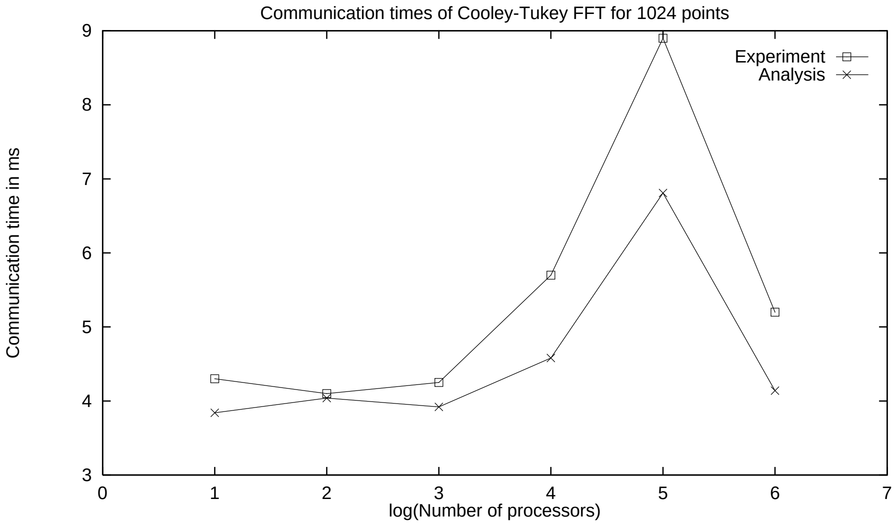

Figure 6: Communication time of Cooley-Tukey FFT for different number of processors.

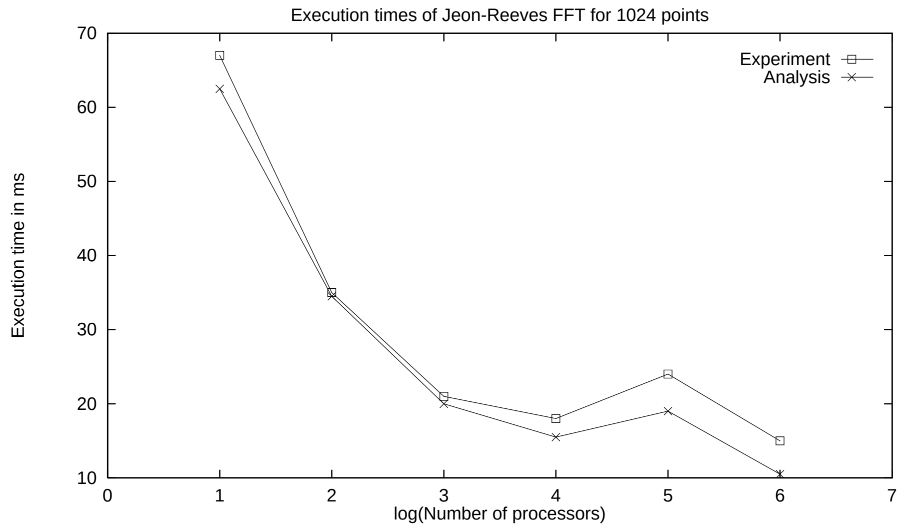

Figure 7: Execution time of Jeon-Reeves FFT for different number of processors.

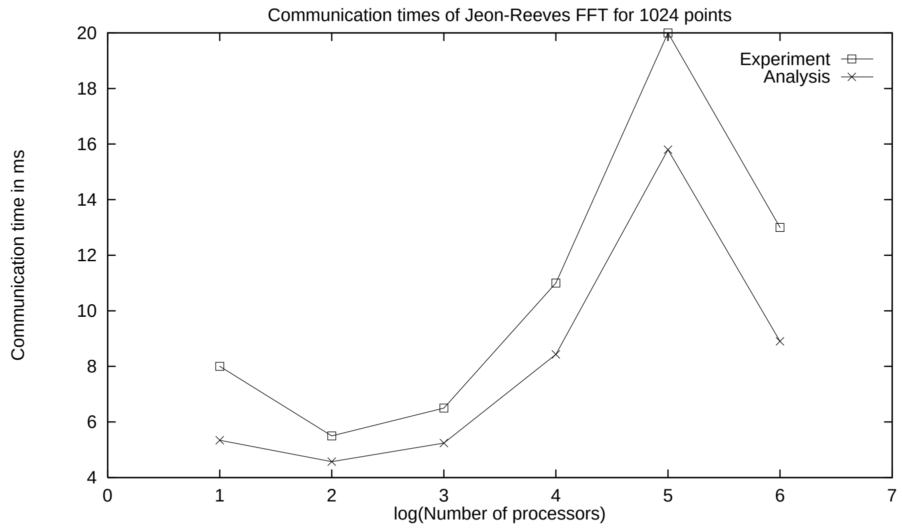

Figure 8: Communication time of Jeon-Reeves FFT for different number of processors.

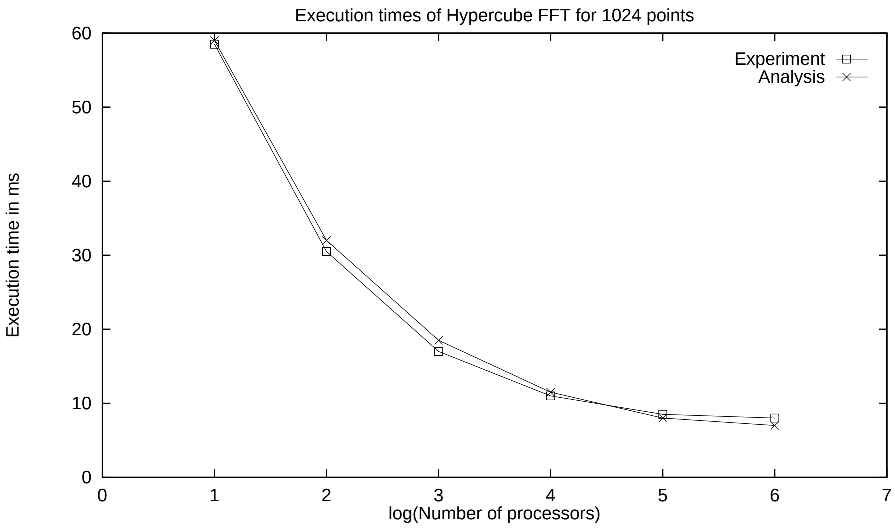

Figure 9: Execution time of hypercube FFT for different number of processors.

based on our model. The reason for this is because every processor transmits two packets, one for data and another for index, to one of its neighbor in every communication step. We can save some time by first transmitting both packets before reading them instead of transmitting and reading first and then doing the same for second packet. This way the setup time for the second packet overlaps with the transmission time of the first packet and we save some time. This saving in time disappears when the packet size becomes very small as we increase the number of processors.

8 Conclusion

Three different FFT algorithms were implemented on a hypercube multicomputer and their performance in terms of computation and communication time was observed. We also developed an analytical model for computation and communication on such a system and analyzed the performance of FFT algorithms based on this model. Fig 5 to Fig 10 shows the comparison of measured execution time and communication time with their estimated values based on the analytical model. As we can see that they match within a tolerable limit.

The interprocessor communication time is a major factor affecting the performance of these algorithms on the hypercube multicomputer. Figs 11, 12 and 13 show the comparison of communication times based on the analytical model for three different problem sizes for these algorithms. From these comparisons it can be concluded that none of the FFT algorithm perform uniformly well

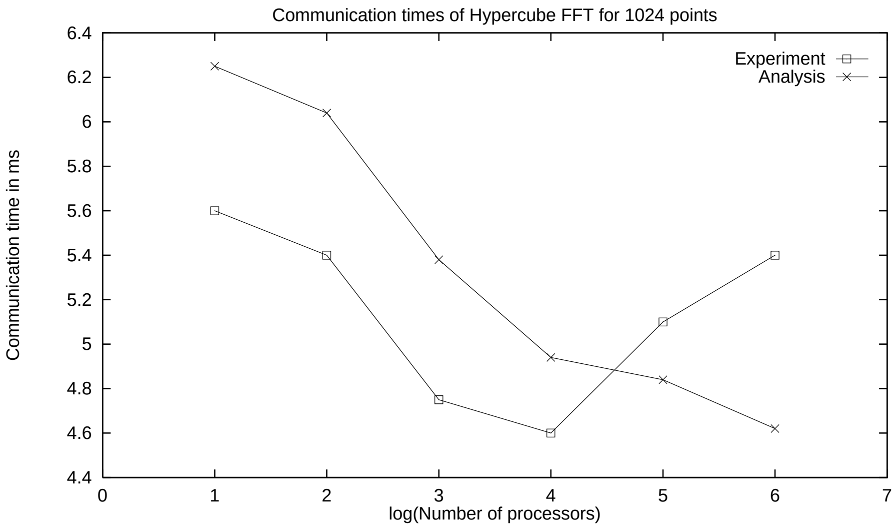

Figure 10: Communication time of hypercube FFT for different number of processors.

throughout the range of number of data points and number of processors in a hypercube multicomputers. We see that the communication time for Cooley-Tukey algorithm is usually less than that of hypercube FFT, except in the vicinity of $p = n/2$, where hypercube FFT performs better than the Cooley-Tukey FFT. It must be noted here that the analytical model does not take into account the time saved by transmitting both the packets before we start reading them in the hypercube FFT. This effect could not be investigated extensively because of the limited memory available with the nCUBE nodes and thus large problem size could not be executed. Our conjecture is that hypercube FFT will have less communication time only in the vicinity of $p = n/2$ and for other cases Cooley-Tukey FFT will have less communication time. We also see that for some problem sizes Jeon-Reeves FFT performs better than other algorithms, while in some other region its performance becomes worst among all. Thus it is very difficult to say that a particular FFT algorithm will perform better than other FFT algorithms on hypercube for all cases. However, in most of the cases hypercube FFT performs fairly well.

References

[1] A. Averbuch, et al., "A parallel FFT on an MIMD machine," *Proc. of 1989 Intl. Conf. Parallel Processing*, 1989.

[2] D. H. Bailey, "A high performance FFT algorithm for vector supercomputers," *International Journal of Supercomputer Applications*, vol. 2, no.

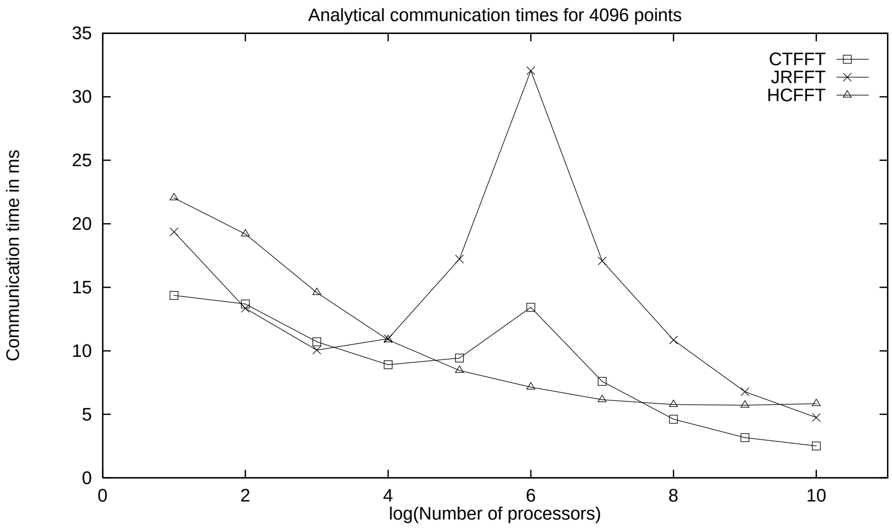

Figure 11: Communication time of FFT algorithms for different number of processors.

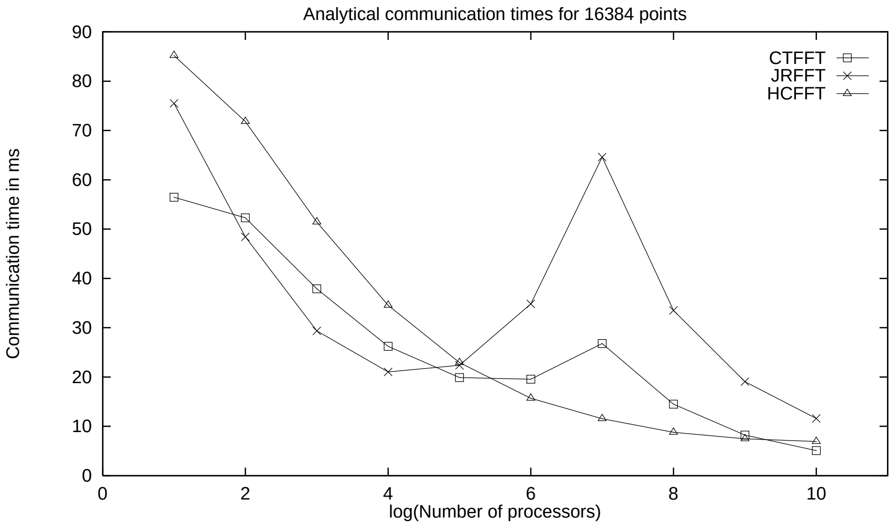

Figure 12: Communication time of FFT algorithms for different number of processors.

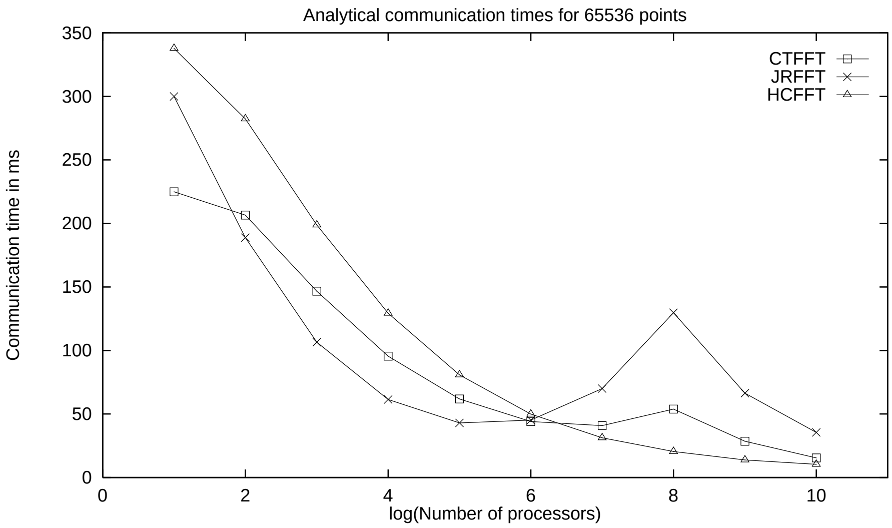

Figure 13: Communication time of FFT algorithms for different number of processors.

1, 1988, pp. 82-87.

[3] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation : Numerical Methods*, Prentice-Hall, Inc., 1989.

[4] L. N. Bhuyan and D. P. Agrawal, "Performance analysis of FFT algorithm on multiprocessor systems," *IEEE Trans. Software Engg.*, vol. SE-9, no. 4, July 1983, pp. 512-521.

[5] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Mathematics of Computation*, vol. 19, Apr. 1965, pp. 297-301.

[6] S. Horiguchi and T. Nakada, "Performance evaluation of parallel fast Fourier transform on a multiprocessor workstation," *Journal of Parallel and Distributed Computing*, vol. 12, 1991, pp. 158-163.

[7] C. H. Jeon and A. P. Reeves, "The fast Fourier transform on a multicluster computer architecture," *Proc. 19th Annual Hawaii Intl. Conf. on System Sciences*, 1986, pp. 103-110.

[8] C. R. Jesshope, "The implementation of fast radix 2 transforms on array processors," *IEEE Trans. Comp.*, vol. C-29, no. 1, Jan 1980, pp. 20-27.

[9] A. Kumar and L. N. Bhuyan, "Parallel FFT Algorithms for Cache Based Shared Memory Multiprocessors," *Proc. of 1993 Int'l Conference on Parallel Processing*, Aug. 1993, vol. III, pp. 23–27.

[10] *nCUBE2 Processor Manual, r.2.0*, nCUBE Corp., Beaverton,OR.

[11] A. Norton and A. J. Silberger, "Parallelization and performance analysis of Cooley-Tukey FFT algorithm for shared-memory architectures," *IEEE Trans. Comp.*, vol. C-36, no. 5, 1987, pp. 581-591.

[12] M. C. Pease, "An adaptation of the fast Fourier transform for parallel processing," *Journal of the ACM*, vol. 15, no. 2, 1968, pp. 252-264.

[13] P. N. Swarztrauber, "FFT algorithms for vector computers," *Parallel Computing*, vol. 1, 1984, pp. 45-63.

[14] P. N. Swarztrauber, "Multiprocessor FFTs," *Parallel Computing*, vol. 5, 1987, pp. 197-210.