IN-61
133988
P.15

NASA Contractor Report 189724

ICASE Report No. 92-56

ICASE

A GENERAL PURPOSE SUBROUTINE FOR FAST FOURIER TRANSFORM ON A DISTRIBUTED MEMORY PARALLEL MACHINE

A. Dubey
M. Zubair
C. E. Grosch

NASA Contract Nos. NAS1-18605 and NAS1-19480
November 1992

Institute for Computer Applications in Science and Engineering
NASA Langley Research Center
Hampton, Virginia 23681-0001

Operated by the Universities Space Research Association

NASA
National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665-5225

A GENERAL PURPOSE SUBROUTINE FOR FAST FOURIER TRANSFORM ON A DISTRIBUTED MEMORY PARALLEL MACHINE [1]

A. Dubey, M. Zubair and C.E. Grosch
Department of Computer Science
Old Dominion University
Norfolk, VA

ABSTRACT

One issue which is central in developing a general purpose FFT subroutine on a distributed memory parallel machine is the data distribution. It is possible that different users would like to use the FFT routine with different data distributions. Thus there is a need to design FFT schemes on distributed memory parallel machines which can support a variety of data distributions. In this paper we present an FFT implementation on a distributed memory parallel machine which works for a number of data distributions commonly encountered in scientific applications. We have also addressed the problem of rearranging the data after computing the FFT. We have evaluated the performance of our implementation on a distributed memory parallel machine Intel iPSC/860.

[1]Research for the last two authors was supported in part by the National Aeronautics and Space Administration under NASA Contract Nos. NAS1-18605 and NAS1-19480 while the authors were in residence at the Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hampton, VA 23681-0001.

1 Introduction

Fourier transform plays an important role in mathematical and numerical analysis. Some of the applications which use the Fourier transform are: digital filtering, auto and cross correlation, solution of partial differential equations etc. The fast Fourier transform algorithm (FFT) computes the transform of an n-component sequence of complex numbers in $O(n \log n)$ time. The implementation of fast Fourier transform on parallel machines has been well studied, for example [6, 3, 4, 7, 5, 8, 11]. However, there has been very little effort in investigating implementations which can provide a basis for a general purpose subroutines on distributed memory machines.

One issue which is central in developing a general purpose subroutine on a distributed memory parallel machine is the data distribution [2]. It is possible that different users wish to use the FFT routine with different data distributions. Typically, users determine their data distribution based on the over all application requirements, which could be different for different users. Thus there is a need to design FFT schemes on distributed memory parallel machines which can support a variety of data distributions.

There are two possible approaches to this problem. The first one is to design an FFT subroutine for a specific data distribution which gives optimal performance, along with a set of basic communication subroutines to convert a user data distribution to the specific data distribution. This approach has the problem of rearranging the user data initially which is quite costly on distributed memory parallel machines. The second approach is to design an FFT routine which works well for arbitrary data distributions. We have designed a scheme which takes the second approach, but only supports a set of common data distributions as opposed to arbitrary data distributions. A common set of data distributions, referred to as block scattered distributions, has been identified by Walker and Dongarra [14] as very useful for distributed memory parallel machines. Block scattered distributions encompass the two most common data distributions; the linear data distribution and the scattered data distribution. For a one dimensional data set, a block scattered distribution is specified by the block size. The data is divided into a set of equal sized blocks. A block j is mapped to node $(j \bmod p)$, where p is the number of nodes. For example, two data distributions for a one dimensional array of 16 data values on a 4 node machine with two different block sizes are shown in Figure 1(a) and 1(b).

In this paper we give an FFT implementation on a distributed memory parallel machine for block scattered distributions of data with different block sizes. We have also addressed the problem of rearranging the data after computing FFT on the same machine. The motivation for rearrangement comes from problems such as solution of partial differential equations using spectral techniques which require the final data distribution to be identical to the initial one. Finally, we evaluated the performance of our implementation on the Intel iPSC/860.

The rest of the paper is organized as follows. In the second section we discuss the issues involved in designing a parallel FFT algorithm. In the third section we propose a parallel FFT algorithm for variable block size. The fourth section describes the rearrangement problem. In the fifth section there is a discussion of the experimental results and in the sixth section we give conclusions.

2 Fourier Transform

The DFT, $X(k)$, of an N-point sequence $x(r)$ is defined as,

$$X(k) = \sum_{r=0}^{N-1} x(r)e^{-j2\pi rk/N}, \quad 0 \le k < N-1, \tag{1}$$

where $j = \sqrt{-1}$. We assume, for convenience, that N is a power of 2.

FFT Algorithm

There are two distinct classes of FFT algorithms; namely decimation in time, and decimation in frequency. We have used a decimation in frequency FFT algorithm and it is described here briefly. (For details one can refer to [1]). To begin with, the N-point sequence $x(r)$ is divided into two halves, $x_1(r)$ and $x_2(r)$ so that the the transformed sequence can be written as

$$X(2k) = \sum_{r=0}^{(N/2-1)} [x_1(r) + x_2(r)]{\omega_N}^{2kr} \tag{2}$$

$$X(2k+1) = \sum_{r=0}^{(N/2-1)} [x_1(r) - x_2(r)]\omega_N^r{\omega_N}^{2kr}, \; k = 0, 1..N/2-1. \tag{3}$$

where $omega_N^r = e^{-j2\pi r/N}$.

These equations represent two $N/2-$point DFT's of sequences $[x_1(r) + x_2(r)]$ and $[x_1(r) - x_2(r)]\omega_N^r$. The process is then repeatedly applied to the two subsequences. An example of an eight point decimation in frequency FFT algorithm is shown in Figure 2(a). The fundamental unit of computation that we use in our algorithm is a butterfly (see Figure 2(b)).

2.1 Parallel Implementation

A typical implementation of FFT on a distributed memory machine results in a sequence of butterflies at each node interspersed with internode communication. Depending upon the initial distribution, data for some of the butterflies is available locally and for others, off-node data are required. There are two approaches for computing butterflies which need off-node data. The first approach splits a butterfly between two nodes, and in the second approach a complete butterfly is computed on a node. The parallelism in the later case is achieved by distributing different butterflies on different nodes. For example, consider a simple case of computing two butterflies on a two node machine as shown in Figure 3(a). Notice that both butterflies need off-node data. The two approaches are illustrated in Figure 3(b) and Figure 3(c) respectively. It is obvious from these figures that the first approach has certain disadvantages. These are:

High communication volume. The first approach requires twice the inter-node communication volume as compared to the second approach.

Unbalanced computational load . The first approach results in additional computation on some of the nodes. For example, the multiplication by ω in the computation of both the butterflies of Figure 3(b) is done on node P_1, unlike the second approach (See Figure 3(c).)

Extra storage. The first approach requires twice the storage of the second approach.

For these reasons our parallel implementation of the FFT algorithm is based on the second approach.

3 Algorithm

As stated earlier, the main feature of our FFT scheme is that it works for block scattered data distribution with variable block sizes. That is, the same algorithm can be used for different data distributions without any initial rearrangement of the data. The algorithm consists of three phases: the first and the third phase compute butterflies for which the data is locally available, and the second phase computes butterflies for which off-node data is required. As a result, internode communication is required only during the second phase. Depending upon the block size the work distribution for the first and third phases will differ. In the extreme cases one of these two phases will not be executed. For a block size of one we need to execute only the first two phases, while for the block consisting of all the data on a node only the last two phases are executed. For all other block sizes all three phases of the algorithm are executed. Given the number of processors, the amount of work in the second phase remains constant for all block sizes. When all the data on a node forms a single block, it must be treated as a special case in phase 2 of the algorithm. This is so because to compute butterflies needing off-node data the block must be divided into two sub-blocks, which does not happen for other block sizes.

An FFT algorithm with data size N has $\log_2(N)$ distinct stages of computation. Each of these stages compute $N/2$ butterflies. In our FFT scheme, as in most other parallel FFT schemes, all the nodes participate in computing a stage by operating on different data points. We describe the algorithm by considering an N point FFT on a p-node machine, with $n = N/p$ as the number of data points mapped per node and b as the block size. The distribution of work in the three phases is as follows

(i) The first $\log_2(n/b)$ stages are computed in the first phase. The butterflies in these stages require data available locally from different blocks.

(ii) The next $\log_2(p)$ stages are computed in the second phase. The butterflies need off-node data, hence internode communication takes place.

(iii) The last $log_2(b)$ stages constitute phase three. The butterflies in phase three are again computed with local data, but from within a block.

The algorithm has a computational kernel **dftstep** which is common to all the three phases. The kernel is common to all nodes and computes all the butterflies of a stage mapped onto a node. It assumes that the ω's values have been precomputed and arranged so that they are available in the right order as needed. A FORTRAN call to the kernel can be made as follows:

$$call\ dftstep(a, w, offset, groups, dist, wincr)$$

where the arguments are:
a Array of input sequence.
w Array of trigonometric coefficients ω.
offset The distance between the two elements of a butterfly.
groups The number of similar sets of butterflies.
dist The distance between two groups.
wincr The stride for **w**.
The kernel **dftstep** essentially computes $n/2$ butterflies. How these butterflies are formed is determined by the three arguments **offset, groups** and **dist.** For example consider the data in p_0 in Figure 1(a). If a stage requires that the butterflies be formed by combining x_0 with x_8 and x_1 with x_9 then a call to dftstep will have $offset = 2$, $groups = 1$ and any value in $dist$. On the other hand, if a stage requires combination of x_0 with x_1 and x_8 with x_9 then the values of these parameters will be $offset = 1$, $groups = 2$ and $dist = 2$.

The pseudo code given below describes the FFT algorithm using 'dftstep'. { This code is executed on each node }

```
begin{phase 1}
    offset = $n/2$ {distance between two points of a butterfly }
    groups = 1 {only one subgroup in the first stage }
    wincr = 1 { stride for $\omega$ }
    for $i = 1$ to $\log_2(n/b)$ do
        dftstep(a,w,offset,groups,offset*2,wincr)
        offset = offset/2
        groups = groups*2
    end for
end {phase 1}
begin {phase 2 }
    offset = $n/2$
    groups = offset/b
        for $i = 1$ to $\log_2(p)$ do
        {negh is node with $i^{th}$ bit differing}
        {exchange half the data with negh }
        negh = $mynode \oplus 2^{d-i+1}$
        exchange(a(k),negh) {if bit $i = 0$ then k = 1, else k = n/2+1 }
        dftstep(a,w,offset,groups,b,wincr)
        if($b = n$) then {special case}
            negh = $mynode \oplus 2^{d}$
            exchange(a(k),negh)
        end if
    end for
end {phase 2 }
begin {phase 3 }
    for $i = 1$ to $\log_2(b)$ do
        groups = groups*2
        dftstep(a,w,offset,groups,offset*2,wincr)
        offset = offset/2
        wincr = 2*wincr
    end for
end {phase 3 }
```

The call to procedure **exchange** initiates a send first and then posts a receive for incoming data. This protocol is followed to ensure concurrent communication [10]. The working of the three phases is shown in Figure 4 with the initial data distribution of Figure 1(a). In this problem a total of 4 stages are required. The first phase is computed in stage one of the algorithm. The butterflies in this stage are formed by the corresponding elements of the two blocks. The second phase is computed in stages 2 and 3 which require exchanges of data. The fourth stage forms the third phase, which is computed by combining the data within a block.

4 Rearrangement

In general FFT algorithms generate the resultant sequence in an order different from the original one. As a result, the data is in the the wrong node. Also all the data to be sent to one node may not be contiguous. In circuit switch or message passing environments, sending one data at a time to a node is extremely expensive due to the overheads [9]. Hence it is desirable that all the data destined for one particular node be collected in one place and sent together. In addition, at the destination node the data from different nodes may need to be interspersed. In view of these problems we require the following steps in the process of rearrangement at each node.

(i) For each data point on a node determine the destination. The destination includes the node number, the block number and the displacement within the block.

(ii) Collect all the data destined for one node in one set to avoid multiple sends.

(iii) Send the collected data to the appropriate nodes and receive data being sent by other nodes.

(iv) Place each data item in its destination block with the correct displacement.

The data rearrangement problem in the worst case is equivalent to the complete exchange problem [13, 12] (see Figure 5(a)). That is each node needs to send data to all the other nodes in the machine. However, that is not always the case. For example, the FFT shown in Figure 5(b) requires data exchange between only a subset of nodes. Each node sends data to and receives data from only two out of four nodes. This situation usually arises when the number of blocks mapped per node is of the same order as the number of of data points per block.

5 Experimental results

We evaluated the performance of our implementation on Intel iPSC/860 for different block sizes and for different data sizes. The Intel iPSC/860 is a distributed memory machine which can have up to 128 nodes. Internode communication is done through a hypercube interconnection network. We carried out all of our experiments on 64 nodes of a 128 nodes machine. The results for different block sizes are summarized in Figure 6. The two curves in Figure 6 represent performance per node in Mflops for the complete code and for the FFT section alone. Recall that the complete code consists of the FFT computation followed by the data rearrangement. These results indicate that while the performance of FFT section varies by small amounts, the overall performance shows more noticeable differences. Both curves tend to peak in the middle. The variation in the over all performance is due to the data rearrangement. Depending on the block size, the data rearrangement may or may not be a complete exchange problem. Also, observe from Figure 7 that the data rearrangement time forms a large enough fraction of the total execution time to have influence on the overall performance.

The reason for the slight variation in the FFT section performance lies in the relative distribution of work between the three phases of the algorithm. In Figure 8 we have plotted the effect of block size on relative distribution of work in the three phases of the FFT section. As expected, the fraction of time taken by the first phase is maximum for the smallest block size and steadily decreases as the block size increases. The third phase exhibits a reverse trend. The region where both these phases have approximately equal work is also the region which shows higher performance in Figure 6. Also notice from Figure 8 that the fraction of time used in the second phase remains almost constant for all block sizes. The the second phase takes more time than the other two since it also involves internode communication.

The effect of data size on the communication and computation time of the fft section of the code is shown in Figure 9. To plot this we picked the best performance for every data size. With very small data sizes almost the entire time is taken up by the communication. As the data sizes increase, computation starts taking larger fractions of the execution time. The computation fraction tends to saturate when the data sizes become sufficiently large. To explain the reason for this behavior we consider the communication characteristics of the machine

In Intel iPSC/860 the cost of communication is determined by the expression

$$t_{comm} = 164 + 0.398\alpha + 29.9\beta$$

where α is the number of bytes in the message and β is the distance between two nodes [12]. The first term in the equation is the setup overhead. As α increases for fixed β, the fraction of total time used in setup decreases. Notice from Figure 9 that the saturation occurs for $\alpha = 2^{11}$. The value of β is 1. For this data size the contribution from the overhead term in the expression is about 5%.

6 Conclusions

In this paper we gave an FFT implementation on a distributed memory parallel machine which works for a number of data distributions commonly encountered in scientific applications.

We evaluated the performance of our implementation on the Intel iPSC/860. The results of our experiments indicate that the variation in block sizes has more effect on the performance of the rearrangement section than on the FFT section. For certain block sizes the data rearrangement cost is significantly lower than for others. In the FFT section, the variation in performance is not significant enough to make the choice of block size a critical issue. On a 64 node machine we obtained a peak performance of 203 Mflops, that is 3.17 Mflops per node. (This figure does not include the initialization costs which are incurred only once for a given size FFT) If we include the data rearrangement, the performance decreases to 2.78 Mflops per node.

We believe that the data rearrangement can be made more efficient. At present we have adapted one of the complete exchange schemes. It is not clear whether it is the best option for the data rearrangement problems arising in FFT or other similar algorithms. Also, the performance can be further improved (expected to be about 25%) by using an optimized (at assembly level) version of **dftstep.**

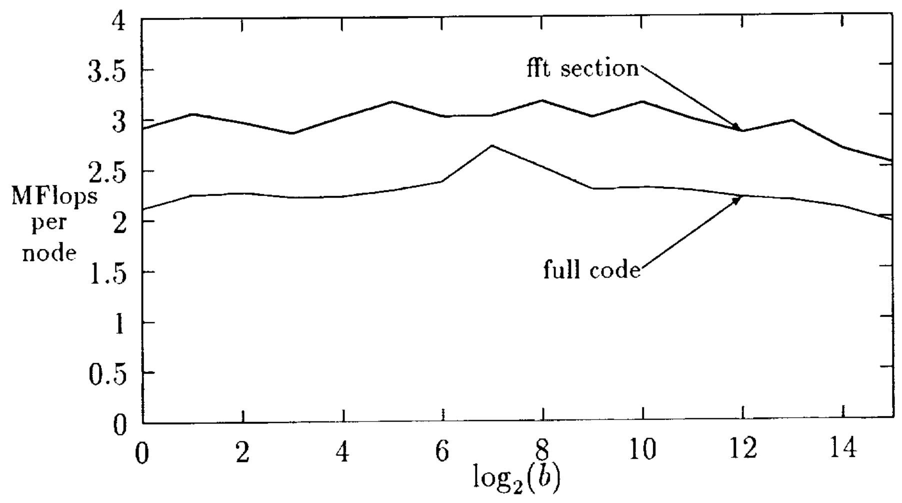

Figure 6. The variation in performance as blocksize is increased.

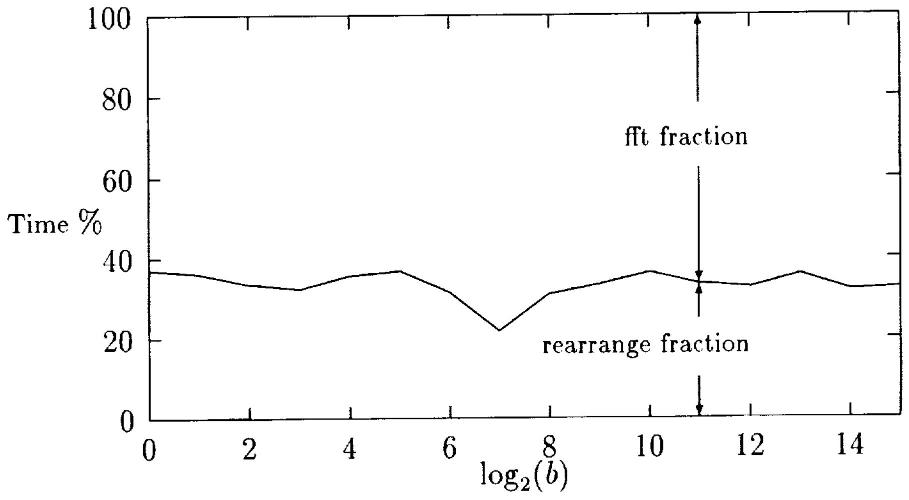

Figure 7. The fraction of time taken by FFT and Rearrangement.

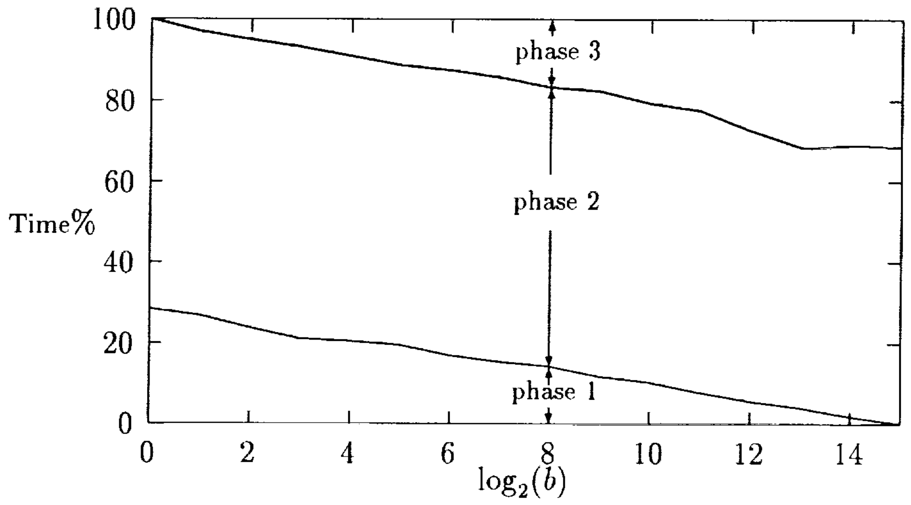

Figure 8. The distribution of work in the three phases with different blocksizes.

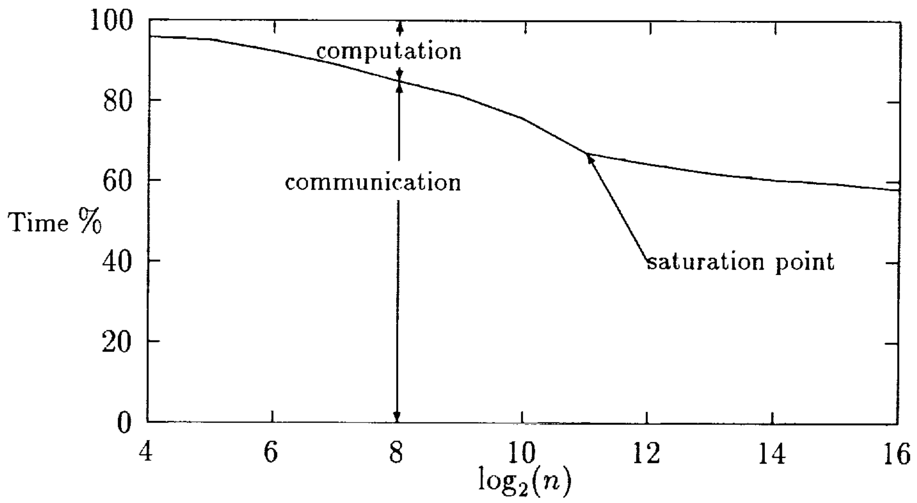

Figure 9. The relative contributions of computation and communication in the FFT section as the datasize is varied.

References

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

[2] C.L. Lawson, R.J. Hanson, and D.R. Kincaid, and F.T. Krogh, "Basic linear algebra subprogram for FORTRAN usage," *ACM Trans. Math. Sotw. 5*, 3, 1979, pp308-323.

[3] R.W. Hockney and C.R. Jesshope, *Parallel Computers*, Adam Higler, 1981.

[4] L.H. Jamieson,P.T. Mueller, and H.J. Siegel, "FFT Algorithm for SIMD Parallel Processing Systems," *Journal of Parallel and Distributed Computing*, vol.3, 1986, pp.48-71.

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

[6] D.H. Bailey, "A High-Performance Fast Fourier Transform Algorithm for the CRAY-2", *The Journal of Supercomputing*, vol.1, 1987, pp.43-60.

[7] R.A. Kamin III and G.B. Adams III, "Fast Fourier Transform Algorithm Design and Tradeoffs on the CM-2," in *Proceedings of the Conference on Scientific Applications on The Connection Machine*, September 1988, pp.134-160.

[8] R.M. Chamberlain, "Gray Codes, Fast Fourier Transforms and Hypercubes," *Parallel Computing* 6 (1988), pp.225-233.

[9] Thomas Schmiermund and Steven R. Seidel, "A Communication Model for the Intel iPSC/2," Computer Science Technical Report CS-TR 90-02, Michigan Tech, Univ, April 1990.

[10] Steven R. Seidel, Ming-Horng Lee and Shivi Fotedar "Concurrent Bidirectional Communication on the Intel iPSC/860 and iPSC/2," Computer Science Technical Report CS-TR 90-06, Michigan Tech, Univ, November 1990.

[11] D.H. Bailey, and P.O. Frederickson, "Performance Results for Two of the NAS Parallel Benchmarks," in Supercomputing '91. (pp.166-173).

[12] Shahid H. Bokhari, "Complete Exchange on the iPSC/860," ICASE Report No. 91-4, NASA Contractor Report No. 187498.

[13] Shahid H. Bokhari, "Multiphase Complete Exchange on a Circuit Switched Hypercube," (ICASE Report No. 91-5), Proceedings of the 1991 International Conference on Parallel Processing, pp. 525-, 1991.

[14] D. W. Walker and J. Dongarra, "Design issues for a scalable library algebra subroutines," Draft Preprint, Oak Ridge National Laboratory October 1991.

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE November 1992	3. REPORT TYPE AND DATES COVERED Contractor Report

4. TITLE AND SUBTITLE A GENERAL PURPOSE SUBROUTINE FOR FAST FOURIER TRANSFORM ON A DISTRIBUTED MEMORY PARALLEL MACHINE	5. FUNDING NUMBERS C NAS1-18605 C NAS1-19480 WU 505-90-52-01
6. AUTHOR(S) A. Dubey, M. Zubair, and C.E. Grosch	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Institute for Computer Applications in Science and Engineering Mail Stop 132C, NASA Langley Research Center Hampton, VA 23681-0001	8. PERFORMING ORGANIZATION REPORT NUMBER ICASE Report No. 92-56
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) National Aeronautics and Space Administration Langley Research Center Hampton, VA 23681-0001	10. SPONSORING/MONITORING AGENCY REPORT NUMBER NASA CR-189724 ICASE Report No. 92-56

11. SUPPLEMENTARY NOTES
Langley Technical Monitor: Michael F. Card
Final Report

Submitted to International Conf. on Mathematical Modelling & Scientific Comp., Dec. 7-11, 1992, Bangalore

12a. DISTRIBUTION/AVAILABILITY STATEMENT Unclassified - Unlimited Subject Category 61	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

One issue which is central in developing a general purpose FFT subroutine on a distributed memory parallel machine is the data distribution. It is possible that different users would like to use the FFT routine with different data distributions. Thus there is a need to design FFT schemes on distributed memory parallel machines which can support a variety of data distributions. In this paper we present an FFT implementation on a distributed memory parallel machine which works for a number of data distributions commonly encountered in scientific applications. We have also addressed the problem of rearranging the data after computing the FFT. We have evaluated the performance of our implementation on a distributed memory parallel machine Intel iPSC/860.

14. SUBJECT TERMS distributed memory parallel machine, Fourier transform	15. NUMBER OF PAGES 14
	16. PRICE CODE A03

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT