A PARALLEL IMPLEMENTATION OF FAST FOURIER TRANSFORM ALGORITHM

A MASTER'S THESIS

in

Computer Engineering

Middle East Technical University

by

Şadan Tuğtekin

April 1988

Approval of the Graduate School of Natural and Applied Sciences.

Professor Dr. Halim DOĞRUSÖZ

Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Professor Dr. Ziya AKTAŞ

Chairman of the Department

We certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Associate Prof. Dr. Ayşe Kiper

Supervisor

Examining Committee in Charge:

Professor Dr. Ziya Aktaş (Chairman) .....

Associate Prof. Dr. Ayşe Kiper .....

Associate Prof. Dr. Bülent Özgüç .....

Associate Prof. Dr. Mürşit Eskicioğlu .....

Assistant Prof. Dr. Mehmet Tolun .....

ABSTRACT

A PARALLEL IMPLEMENTATION OF

FAST FOURIER TRANSFORM ALGORITHM

TUĞTEKİN, Şadan

M.S. in Computer Engineering

Supervisor: Assoc. Prof. Dr. Ayşe Kiper

April 1988, 141 pages

Parallel implementations of the fast Fourier transform algorithms (FFT) on multiprocessor systems are investigated and simulated with concurrent programming.

Two parallel FFT implementations are considered for simulation on a VAXcluster system which consists of two interconnected VAX 11/780 computers. These are the Bergland's parallel FFT algorithm on a parallel element processing ensemble, and the decimation-in-time radix-2 FFT algorithm on a loosely coupled multiprocessor system. A set of simulation programs is developed for each implementation, in VAX FORTRAN language. Furthermore, each simulation is tested for a set of time domain waveforms and results are discussed in terms of complexity, execution time, and efficiency.

Key words: FFT algorithm, parallel implementation, simulation, concurrent programming, execution time.

ÖZET

HIZLI FOURIER DÖNÜŞÜM YÖNTEMİNİN PARALEL UYGULANIŞI

TUĞTEKİN, Şadan

Yüksek Lisans Tezi, Bilgisayar Müh. Böl.

Tez Yöneticisi: Doç. Dr. Ayşe Kiper

Nisan 1988, 141 sayfa

Bu çalışmada hızlı Fourier dönüşüm (HFD) yönteminin çoklu işletim birimine sahip bilgisayarlardaki paralel uygulanışı araştırılmış ve eşzamanlı programlama ile benzetimi yapılmıştır.

Bu uygulamalardan iki tanesi, VAXcluster denilen, iki VAX 11/780 bilgisayarın karşılıklı bağlanmasından oluşan sistemde, benzetim yapmak için ele alınmıştır. Bunlardan biri Bergland'ın paralel HFD yönteminin, paralel eleman işletim bütünü, denilen bilgisayardaki uygulanışı ve diğeride zamanda-parçalama taban-2 HFD yönteminin, bir gevşek bağlantılı çoklu işletim birimi, sistemindeki uygulanışıdır. Her iki uygulama şekli için, VAX FORTRAN dilinde benzetim programları geliştirilmiştir. Ayrıca, her benzetim çeşitli dalga şekilleri için test edilmiş ve sonuçlar karmaşıklık, çalışma zamanı ve verim yönünden tartışılmıştır.

Anahtar kelimeler: HFD yöntemi, paralel uygulanış, benzetim, eşzamanlı programlama, çalışma zamanı.

ACKNOWLEDGEMENTS

It is my pleasant duty to express my sincere gratitude to Assoc. Prof. Dr. Ayşe Kiper for her appreciable supervision and valuable comments.

I would also like to thank Assoc. Prof. Dr. Atila Gönder, Manager of the Computer Department of ASELSAN Inc., for providing access to the necessary instruments and software. Acknowledgements are also due to all staff of the Computer Department of ASELSAN Inc., for their patience and encouragement throughout the project.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1.

INTRODUCTION

Fourier analysis is not a new subject. It has been around since 1800s when J.B.J. Fourier developed the initial concepts and theory. Since then numerous papers and books dealing with Fourier theory have been published and the Fourier transform has found its way into signal processing area.

Both the Fourier series and the Fourier integral transform are the Fourier analysis tools used for investigating the frequency spectrum of a time domain waveform with a set of conditions. The discrete Fourier transform (DFT) is a third type of Fourier transform, and it is defined only for discrete values over a finite time interval.

The term "fast Fourier transform" or "FFT" refers to a family of algorithms that can perform the DFT calculation with much less computational effort than the straightforward approach.

The FFT algorithms are based on decomposing the DFT calculation of a sequence of length N, into successively smaller DFTs. Two well-known types of FFT algorithms, "decimation-in-time" and "decimation-in-frequency", are derived from this decomposition.

Since its general introduction in 1965, the FFT has been a commonly used algorithm for evaluating the DFT.

Today, the FFT algorithm forms the core of digital signal processing. It provides an efficient method of performing the operations; convolution, correlation, and Fourier analysis, to emphasize a basis for radar signal processing, sonar signal processing, seismic signal processing, speech analysis, pattern recognition, image processing and the other branches of digital signal processing.

Since the FFT algorithm is basic for implementing all of these operations, the extent to which they can be done in real-time is often limited by the rate at which the FFT algorithm can be executed. High performance requirements, for real-time, led to the design of special-purpose FFT processors or the parallel implementations on multiprocessor systems.

Although a considerable amount of work has been done in the design of special-purpose FFT processors, very few researchers have studied the performance of the FFT algorithm on a general-purpose multiprocessor system.

The goal of this work is to investigate the parallel FFT implementations and to realize a few of them by simulating with concurrent programming on an available loosely coupled two-node VAXcluster system. Two different FFT algorithms, namely, Bergland's parallel FFT algorithm and decimation-in-time radix-2 FFT algorithm have been chosen for this purpose.

The first one; Bergland's parallel FFT algorithm, has been simulated by using only one node of the VAXcluster

system, and the second one; decimation-in-time radix-2 FFT algorithm, has been simulated by using both nodes of the system.

This thesis contains seven chapters and an appendix. Chapter 2 presents an overview for the discrete Fourier transform and fast Fourier transform algorithm to emphasize a basis for the chosen parallel FFT algorithms, given in Chapter 4. Chapter 3 gives a survey on FFT implementations in two main sections. First section describes the initial implementations and the second section introduces the recent developments for the FFT algorithm. Chapter 4 is about parallel FFT algorithms, namely, Bergland's parallel FFT algorithm and decimation-in-time radix-2 FFT algorithm, and contains a detailed information for their implementations on a parallel element processing ensemble and a shared bus computer, respectively. Chapter 5 describes the simulations of the algorithms, introduced in Chapter 4, in a VAXcluster environment by giving an information about the written simulation programs. Some numerical results for the FFT simulation software of the familiar time domain waveforms: exp(-x), sin x, and cos x, are given in Chapter 6. These results are obtained from the sample runs of the simulation programs. In addition, total execution times of the simulations are also discussed this chapter.

Chapter 7 gives a summary for the simulations explained in Chapter 5, and concludes the study with some comments for future implementations of the FFT algorithm. Finally, Appendix-A contains the source listings of the simulation programs.

CHAPTER 2.

FAST FOURIER TRANSFORM ALGORITHM

The fast Fourier transform is a computational tool which facilitates the signal processing by means of digital computers. It is simply a method for efficiently computing the discrete Fourier transform.

From a digital viewpoint, the FFT realizes the exact transformations of the given discrete values. Either a continuous periodic signal or a continuous nonperiodic signal can be sampled to provide the discrete values. These discrete values can then be transformed to frequency domain by the FFT. Thus, the FFT provides the correct frequency domain information for the sampled version of a time domain waveform.

Since the FFT is a method for computing the DFT, it is appropriate to begin with a brief review of the DFT.

2.1. DISCRETE FOURIER TRANSFORM

The discrete Fourier transform is a third type of Fourier transform and related to the Fourier series and Fourier integral transform.

The Fourier integral transform is also called as continuous Fourier transform or just as Fourier transform. It is one of the most important mathematical tools used in

signal processing both from theoretical and practical viewpoints.

The discrete Fourier transform is derived from the continuous Fourier transform and applied only to the sampled versions of the periodic or nonperiodic time domain waveforms. The common form of the N-point DFT and its inverse are

$$A(p) = \sum_{q=0}^{N-1} w^{pq}\, a(q), \qquad p = 0,\ldots,N-1, \tag{2.1}$$

$$a(q) = N^{-1} \sum_{p=0}^{N-1} w^{-pq}\, A(p), \qquad q = 0,\ldots,N-1, \tag{2.2}$$

where $w = w_N = e^{i2\pi/N}$, and $i = (-1)^{1/2}$. A(p) represents the p th coefficient of DFT and a(q) denotes the q th sample of the time series which consists of N samples. The a(q)'s can be complex numbers and the A(p)'s are almost always complex.

This definition of the DFT is not uniform in the literature. Some authors use A(p) / N as the DFT coefficients, others use A(p) / $N^{1/2}$, still others use a negative exponent in the weighting function w.

A continuous waveform can be sampled to provide the input discrete values of DFT. In practice, these samples can be easily obtained by using an analog-to-digital converter. There are two basic concepts in the analog-to-digital conversion of a continuous waveform. They are

- windowing: taking a part of the waveform over a finite time interval,
- sampling : finding the function values of the windowed waveform between specified time intervals.

As an example, the result of an analog-to-digital conversion of a cosine wave is shown in Fig.2.1. In this conversion, 0.01 sec is used as a sampling time interval over the window, from 0.00 to 0.31 sec. There are 32 equally spaced samples in the window. The function values of the samples are stored in the a(q)'s and used as input data for the DFT in Eq.(2.1).

After the DFT computation on these samples by using an FFT algorithm, the corresponding DFT coefficients, A(p)'s, contain the frequency components of the given cosine wave. The result is shown in Fig.2.2. In the same manner, the inverse DFT in Eq.(2.2) can be used for transforming the A(p)'s, back to the initial discrete samples, a(q)'s.

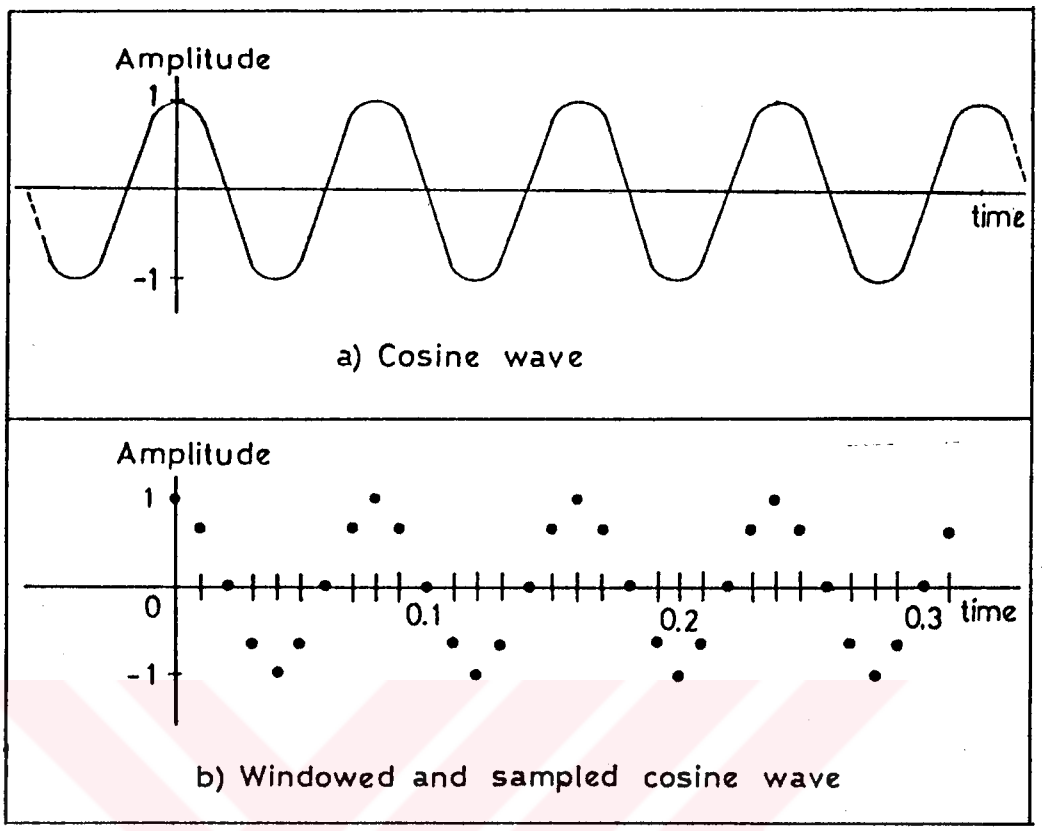


Figure 2.1 Analog-to-digital conversion of a cosine wave.

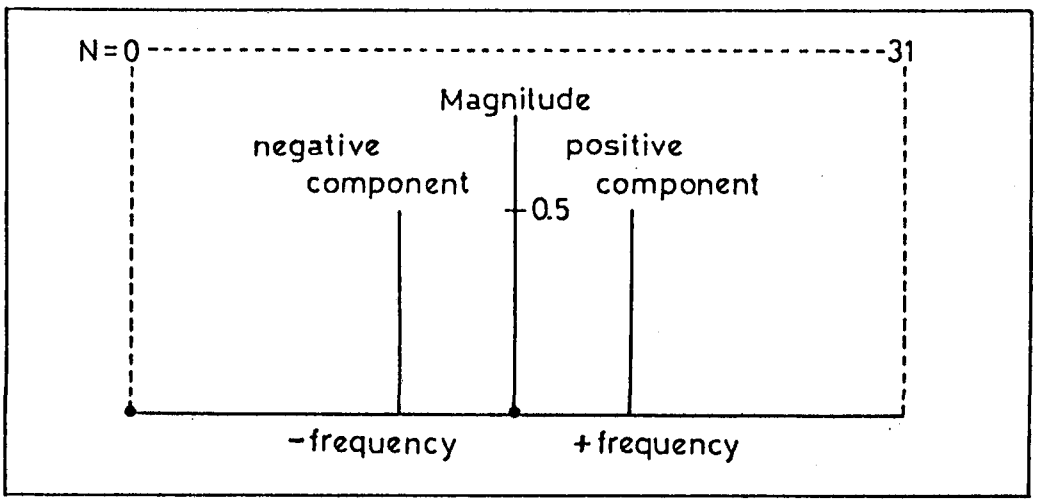


Figure 2.2 Frequency components of the cosine wave.

2.2. DERIVATION OF FFT ALGORITHM

After the brief review of the DFT in previous section, an example derivation of the FFT algorithm is given in the following.

A straightforward calculation of the DFT in Eq.(2.1) would require N^2 operations where "operation" means a complex multiplication followed by a complex addition.

The FFT algorithm described here is the original Cooley-Tukey algorithm [14], and yields the result in approximately $N.\log_2 N$ operations. To derive the algorithm, suppose that the number of discrete complex data points, N, is a composite number such that $N = r_1.r_2$.

Then, the indices in Eq.(2.1) can be expressed as

$$p = p_1 r_1 + p_0, \quad p_0 = 0,\ldots,r_1-1, \quad p_1 = 0,\ldots,r_2-1, \tag{2.3}$$

$$q = q_1 r_2 + q_0, \quad q_0 = 0,\ldots,r_2-1, \quad q_1 = 0,\ldots,r_1-1, \tag{2.4}$$

and then, one can write

$$A(p_1,p_0) = \sum_{q_0} \sum_{q_1} w^{pq}\, a(q_1,q_0), \tag{2.5}$$

where

$$w^{pq} = w^{p(q_1 r_2 + q_0)}$$

$$= w^{pq_1 r_2}\, w^{pq_0} \tag{2.6}$$

Note that, this decomposition of the time sequence index "q" has been used in the original Cooley-Tukey [14] FFT algorithm and called as "decimation-in-time". The decomposition of the frequency spectrum index "p" has been used instead of "q" in the Sande-Tukey version [20] and called as "decimation-in-frequency". While the methods differ, the two algorithms produce the same final results. It is just a different path.

Justifying the first term of Eq.(2.6)

$$
\begin{aligned}
w^{pq_1r_2} &= w^{(p_1r_1+p_0)q_1r_2} \\
&= w^{r_1r_2p_1q_1}\, w^{p_0q_1r_2} \\
&= (w^N)^{p_1q_1}\, w^{p_0q_1r_2} \\
&= w^{p_0q_1r_2} \qquad (2.7)
\end{aligned}
$$

since

$$
\begin{aligned}
w^N &= e^{i2\pi N/N} = e^{i2\pi} \qquad (2.8) \\
&= \cos(2\pi) + i\sin(2\pi) \\
&= 1.
\end{aligned}
$$

Then,

$$
w^{pq} = w^{p_0q_1r_2}\, w^{pq_0} \qquad (2.9)
$$

and therefore, the Eq.(2.5) becomes

$$A(p_1,p_0) = \sum_{q_0} \sum_{q_1} a(q_1,q_0)\, w^{p_0 q_1 r_2}\, w^{p q_0}. \qquad (2.10)$$

The inner sum, over q_1, depends only on p_0 and q_0 and can be defined as an individual sum

$$a_1(p_0,q_0) = \sum_{q_1} w^{p_0 q_1 r_2}\, a(q_1,q_0). \qquad (2.11)$$

The result, then can be written

$$A(p_1,p_0) = \sum_{q_0} w^{(p_1 r_1 + p_0) q_0}\, a_1(p_0,q_0). \qquad (2.12)$$

The steps Eq.(2.11) and Eq.(2.12) forms the original Cooley-Tukey [14] FFT algorithm for $N = r_1.r_2$. If we are able to choose N to be a highly composite number, then we can make very real gains. This is the key idea of the FFT algorithm. That is, if N is a highly composite number such that

$$N = r_1.r_2 \ldots r_m,$$

then the transformation can be decomposed into m stages with N/r_i sub-transformations of size r_i within each stage. Here, if each r_i equals to r, then N becomes r^m and the resulting FFT algorithm takes a prefix name as "radix-r".

After Cooley-Tukey [14], most subsequent authors have directed attention to the special case of $N = 2^m$. Explanation and programming are simpler for $N = 2^m$ than for general case, and the restricted choice of values of N is adequate for a majority of applications. In addition, in the case of $N = 2^m$, the reduced number of operations, $N.\log_2 N$, can be obtained exactly. The general form of the original Cooley-Tukey [14] FFT algorithm for $N = 2^m$ is stated in Section 4.2.2 of Chapter 4.

There are also a number of other algorithms, offshoots of the Cooley-Tukey algorithm (see [20],[24], and [25]), that are lumped under the term FFT, and any algorithm that provides the $N.\log_2 N$ advantage is generally referred to as an FFT.

CHAPTER 3.

PAST RESEARCHES RELATED WITH FFT

An efficient algorithm for the computation of discrete Fourier transform was first reported by J.W. Cooley and J.W. Tukey [14] in 1965. Since the algorithm was required much less computational effort than the direct methods, it was called as the "fast Fourier transform". The fast Fourier transform algorithm has produced major changes in the computational techniques of digital signal processing. The FFT algorithm has a long and interesting history that has been summarized by Cooley, Lewis and Welch [15].

A software implementation of N-point FFT algorithm on a general-purpose serial machine reduce the N^2 major operations of DFT computation to $N.\log_2 N$. Even greater gains can be realized by either a special-purpose FFT processor or a parallel implementation on a multiprocessor system.

In most applications of digital signal processing, a processing rate slower than real-time would overload the system with input data or lead to worthless results. The other applications involve off-line processing where the volume of data makes processing impractical unless a dedicated machine is used. Because of these inherent real-time constraints of digital signal processing problems, a considerable amount of work has been done in

the design of special-purpose FFT processors. But unfortunately, very few researchers have studied the performance of the FFT algorithm on a general-purpose multiprocessor system. However, since the FFT is a highly parallel algorithm, the real-time digital signal processing is an area where parallel computers can be used particularly advantageously.

In this chapter, the developments in the implementation of FFT algorithm are divided into two main sections. In the first section, several papers concerning the subjects; extension hardware units, special-purpose FFT processors, and first FFT programs on general-purpose computers are referenced. In the second section, the recent developments, such as machine oriented algorithms with their implementations on special-purpose FFT processors, improvements in FFT algorithm, implementations for these improved forms of FFT, vector computer FFT implementations, radix-2 FFT butterfly processors, and ideal machine architectures for parallel FFT implementations will be mentioned.

3.1. INITIAL FFT IMPLEMENTATIONS

In this section, we try to summarize the first developments interested in the initial implementations of the FFT algorithm. These hardware related developments began with the capability enhancements of general-purpose computers for digital signal processing. The use of

extension hardware units, such as additional processors, extension memory units, and auxiliary storage devices, for the avaliable host computers has been discussed by many authors in the late 1960s. Furthermore, many FFT programs coded in ALGOL or FORTRAN have been implemented on general-purpose computers. Meanwhile, also many scientists and companies have studied the design of special-purpose FFT processors. We now refer to several papers that are concerned with the above subjects.

3.1.1 Extension Hardware Units for Host Computers

Gold, Lebow, McHugh, and Rader [22] introduced the architecture of a fast digital processor (FDP). It was a general-purpose digital attachment to a UNIVAC 1219 computer. The main purpose of the FDP was to enhance the capability of the UNIVAC computer for performance of digital signal processing operations. This was accomplished by a structure which allows for a degree of parallel processing, pipeline methods, and the use of high speed circuits. The FDP has been constructed at the M.I.T. Lincoln laboratory.

Wesley [43] proposed an associative memory as a parallel processing unit for the FFT. He derived the formulas for the number of memory operations required to execute the algorithm for various number of data points.

He showed that a 1024 word by 64 bit memory with an operation time of 100 ns could execute a 1024 point transform in 8.4 ms.

Singleton [37] and Brenner [11] described two different methods for computing the FFT of the large volume of data using external storage devices. In addition, Singleton has coded his method in ALGOL of Burroughs B5500 computer.

3.1.2 FFT Programs

Bergland [1],[2], Singleton [38], and Brenner [10], presented several FORTRAN programs for the various forms of the FFT algorithm.

Bergland [1] gave the total number of complex multiplications for the optimized FFT programs in radices 2,4,8, and 16, and pointed out that radix-8 with provision for an additional factor of 4 or 2, is a good choice for an efficient FFT program when N is a power of two. In addition, Bergland [2], wrote a radix-8 FFT program for the real-valued time series depending upon the results of his previous paper.

Programs written by Singleton [38] and Brenner [10], were for two different mixed radix FFT algorithms,

described mathematically in their papers. The major difference of Singleton's program was that an efficient method to arrange the results in ascending order. The number of complex multiplications was reduced in both programs by similar methods. However, Singleton additionally was used Bergland's [1] results for this reduction. These optimized programs have been implemented on a number of computers, such as CDC 6400, CDC 6600, IBM 360/67, and Burroughs B5500, and Singleton's program was found significantly faster.

3.1.3 Special-Purpose FFT Processors

Pease [32] presented a work which was concerned with the development of a variation of the FFT algorithm that was better adapted to parallel processing in a special-purpose machine. His primary objective was to see what capabilities would be needed in such a machine. He has concluded his study in another paper and proposed a valuable machine architecture for the parallel FFT implementation.

Bergland [3] prepared a list of features that serves to characterize an FFT processor. Based on this list, information was requested from 20 companies that each one has at least one FFT processor. The published table in the paper represents a compilation of the returns. After

this surveying, Bergland [4] introduced the problems associated with implementing the FFT algorithm in hardware and provided a frame of reference for characterizing specific implementations. He also described many of the design options applicable to an FFT processor, and gave a brief comparison of several machine organizations.

Bergland and Wilson [5] presented an FFT algorithm for an unstructred, parallel ensemble of computing elements with global control. The procedure has made efficient use of a fixed-size memory and has minimized the data transmission between computing elements. In this paper, some practical considerations of the tradeoffs between element utilization and gain of computing speed via parallelism were also included.

Groginsky and Works [27] described a novel structure for an FFT processor using Bergland's [4] alternative organizations for FFT processors. The machine was composed of nine processing modules, a synchronizer, an A/D converter, and a display. The pipeline structure was used to implement a real-time FFT machine to provide spectral analysis for wide-band radars. This machine was capable of processing simultaneously eight channels of input data at a sample rate of 16000 complex samples per second per channel with a transform length of 512 complex samples. This pipelined organization of the FFT processor has been constructed and used for radar signal processing.

Corinthios [16] introduced an arbitrary-radix FFT algorithm and design of its implementing signal processing machine. The algorithm, yields an implementation with a level of parallelism proportional to the radix-r factorization of the DFT, allowed 100 percent utilization of the arithmetic unit, and yields properly ordered Fourier coefficients without the need for pre- or postordering of data.

3.2. RECENT DEVELOPMENTS

Over the past few years summarized in the previous section, a large body of knowledge has been generated on the subject of FFT algorithms. While most of this work has been concerned with hardware, a substantial information has also been made available for programming.

FFT programs, like Bergland [1],[2], Singleton [38], and Brenner [10], naturally have to be conditioned by the architecture of the particular serial computer. On the other hand, the designer of a special-purpose FFT device has the freedom to match the hardware design and the FFT algorithm. This freedom is of particular advantage in designing high-speed systems. We refer here a few examples of such systems.

Several scientists, Good [25], Winograd [45], Sudhakar, Agarwal, and Roy [40], and Swartzrauber [41], have concerned only with the mathematical and

computational aspects of the FFT algorithm. The consequences of these interests are the improved forms of the FFT algorithm which are more flexible and general than the original in some special application fields. Papers related with these particular improvements and their subsequent implementations, Bergland [6], Chu-Burrus [13], and Chow, Vranesic, and Yen [12], are included in this section.

After the parallel and vector computers became operational, a few researchers, such as Korn-Lambiotte [28], and Fornberg [19], have studied the implementation of the FFT algorithm on these systems. In this section, two FFT implementations on the CDC STAR-100 vector computer are mentioned.

More recently, a number of single chip radix-2 butterfly processors has been fabricated using the very large scale integration (VLSI) architecture. The related papers, White [44], Mactaggart-Jack [30],[31], are also included in this section.

Finally, a number of proposed machine architectures, like Pease [33], Stone [39], is mentioned. These machines have a high degree of parallelism in the numerical solutions of the time consuming problems, such as FFT, partial differential equations, polynomial evaluation, and large matrix operations.

3.2.1 Special-Purpose FFT Processors

Gold and Biallly [23] reviewed the parallelism in fast Fourier transform algorithm. They derived a diagrammatic representation for mixed radix and highest radix FFT algorithms. Using this representation, they have explored the classes of FFT hardware from the point of view of speed, parallelism, radix number and type of memory.

Gottlieb and Lorenzo [26] prensented an algorithm that introduced two degrees of parallelism into the implementation of FFT processors. They also described a serial vector multiplier that was ideally suited to the implementation of a general radix arithmetic unit. Two different machine organizations, four arithmetic unit radix-2 and single arithmetic unit radix-4 were compared in terms of speed, complexity, and cost efficiency.

Despain [18] presented a new, especially attractive FFT computer architecture using the compensated and the modified CORDIC iterations. In the CORDIC iterations, the number of operations was found as a function of transform method and radix representation. Using these representations, he also examined several hardware configurations for cost, speed, and complexity tradeoffs.

Corinthios, Smith and Yen [17] introduced the organization and functional design of a parallel radix-4

FFT computer for real-time signal processing of wide-band signals. The constructed special-purpose FFT processor performed a real-time processing of signals at a rate of up to 1.6 million samples per second. The computation time of the FFT of 1024 samples was equal to 0.64 ms.

3.2.2 Improvements in FFT Algorithm

Good's [25] main purpose was to show as clearly as possible the mathematical relationship between the two basic fast methods, Cooley-Tukey [14] and Good [24], used for the calculation of DFTs and to generalize one of the methods a little further. He has also developed a prime-factor algorithm for decomposing the DFT. His method can be applied to all those linear transformations whose matrices are expressible as direct methods.

Winograd [45] described new algorithms using the results of his own previous study for computing the DFT. In his previous study, he has investigated the minimum number of multiplications needed to obtain the coefficients of the product of two (n-1) st degree polynomials in mod (n th degree polynomial). These new algorithms used about the same number of additions as the algorithm proposed by Cooley and Tukey [14] but only about 20 percent of the number of multiplications which their algorithm required.

Sudhakar, Agarwal, and Roy [40] proposed an algoritm for computing the Fourier transform. The new algorithm was more general than existing ones. It was retained the order of computation at FFT levels, but allowed the flexibility of choosing arbitrary frequencies for a uniformly sampled signal. The method was based on the fact that the Fourier transform of a finite length sequence at any arbitrary frequency can be expressed as a weighted sum of its DFT coefficients. The computational aspects of the algorithm and its behavior with typical signals have been critically examined.

Swarztrauber [41] proposed a number of new algorithms for symmetric FFTs that were somewhat more efficient and general than existing ones. Note that the symmetric FFTs are used extensively in the solution of boundary-value problems in partial differential equations. The algorithms prensented in this study have been obtained by modifying the FFT itself and they have given satisfactory results with more less computational times because of the eliminated steps from original algorithm.

3.2.3 Implementations for the Improved Forms of FFT

Bergland [6] described an FFT algorithm for the parallel FFT implementation on a parallel element processing ensemble (PEPE), previously described by

Githen [21]. Several methods of implementing the FFT algorithm that permit parallel operations were also described by Bergland in [4]. An additional parallel implementation was proposed by Bergland and Wilson in [5]. Bergland's new parallel FFT algorithm combined the components of Bergland and Wilson's algorithm with a variation of the Good's [25] prime-factor algorithm. The resulting algorithm computes an N-point DFT in a parallel manner when N can be expressed as the product of two integers R and S which are relatively prime. Bergland have concluded that the algorithm is most efficient when R is considerably smaller than S.

Chu and Burrus [13] developed a time-efficient prime factor Fourier transform (PFFT) algorithm for calculating the DFT. First, Good's [25] prime-factor algorithm was used for decomposing the DFT into multiple short prime DFT's. Next, a method of Winograd [45] based on the index permulation proposed by Rader [34] was used to convert the short DFT's into convolution. When programmed on a Z80 microprocessor, the algorithm becomes 2-20 times faster than conventional algorithms. The approach has also made it possible to add a simple external logic to a microprocessor system to further increase the speed.

Chow, Vranesic, and Yen [12] showed that previous experience of Chu and Burrus [13] in implementing the PFFT is much more difficult to do than the FFT because of its

complicated structure. They also described a general architecture for implementing PFFT machines using distributed arithmetic and lookup tables in ROM's to perform the computations. It was found to be much simpler and more modular than a design which uses multipliers and adders. The proposed architecture was also suitable for computing other linear functions.

3.2.4 Vector Computer FFT Implementations

Korn and Lambiotte [28] presented two algorithms for performing an FFT on a vector computer. The first one was an adaptation and extension of an algorithm proposed by Pease [32] and second one was a variant of the original Cooley-Tukey [14] FFT algorithm. These two algorithms have been compared on the CDC STAR-100 vector computer and the relative merits of both algorithms have been shown to depend upon whether only a few or many independent transforms are desired.

Fornberg [19] improved the results in the previous paper of Korn and Lambiotte [28] by considering the relative hardware speeds between different machine instructions. He also showed how the use of a different algorithm and trigonometric tables lead to more than three times faster execution times. But, the times for large transforms increase only about 39 percent if the tables are eliminated in order to save storage.

3.2.5 Radix-2 FFT Butterfly Processors

White [44] developed a very simple computational requirement for the FFT butterfly. Based upon the distributed arithmetic and an algebraic substitution an architecture has been developed. This architecture can be used effectively with a serial interface to very efficiently mechanize the radix-2 FFT butterfly computation for both decimation-in-time and decimation-in-frequency configuration.

Mactaggart and Jack [30] described a two-dimensional pipelined FFT butterfly processor using a distributed arithmetic algorithm. An 8-bit processor which contains approximately 8000 transistors has been fabricated based on this architecture.

Mactaggart and Jack [31] again, developed another arithmetic processor chip for radix-2 FFT butterfly computation. But, in this case, it has a modified architecture with 16-bit design and higher clocking rate. They also described in detail that the distributed arithmetic techniques can be applied in parallel-data arithmetic computations to achieve highly regular and efficient VLSI structures on silicon.

3.2.6 Ideal Machine Architectures for Parallel FFT Implementations

Pease's [33] study explored the possibility of using a large-scale array of microprocessors as a computational facility for the execution of massive numerical computations with a high degree of parallelism. The array studied was called as the "indirect binary n-cube array". He has used a network of switching nodes to obtain a high degree of flexibility of the connections among the microprocessors. The use of the array for radix-2 FFT, partial differential equations, and matrix multiplication, have been considered. Finally, he has concluded that the indirect binary n-cube is very useful for a wide range of important problems, and thus attractive candidate for the design of a large-scale array of microprocessors.

Stone [39] proposed an interconnection pattern for a parallel processor that was called as the "perfect shuffle". He has defined the perfect shuffle such that given a vector of elements, the perfect shuffle of this vector is a permutation of the elements that are identical to a perfect shuffle of a deck of cards. Elements of the first half of the vector are interlaced with elements of the second half in the perfect shuffle of the vector. He has also indicated by a series of examples that the perfect shuffle is an important interconnection pattern for a parallel processor. The examples include the FFT, polynomial evaluation, sorting and matrix transposition.

Bhuyan and Agrawal [7] considered a decimation-in-time radix-2 FFT algorithm for implementation in multiprocessors with shared bus, multistage interconnection network [8],[29],[33],[39], and in mesh connected computers [9],[39]. Results have been derived for data allocation, interprocessor communication, approximate computation time, and speedup of an N-point FFT on any P available processing elements. A performance comparison has also been carried out mainly between a shared bus multiprocesssor and a computer with multistage interconnection network.

CHAPTER 4.

PARALLEL FFT IMPLEMENTATIONS

In this chapter, we describe two different parallel FFT algorithms with their original implementations. The first one is the Bergland's [6] parallel FFT algorithm on a parallel element processing ensemble (PEPE) [21], and the second one is the decimation-in-time radix-2 FFT algorithm on a loosely coupled multiprocessor system [7]. The Bergland's parallel FFT algorithm is easy to simulate with concurrent programming on a serial machine, such as VAX 11/780. However, the decimation-in-time radix-2 FFT algorithm is adequate for a majority of signal processing applications, and also has an applicable structure for the available VAXcluster system. The simulated implementations of these two algorithms emphasize the core of this study.

4.1. BERGLAND'S PARALLEL FFT ALGORITHM

In 1972, Bergland derived algorithms for parallel FFT implementation. In the basic form of Bergland's parallel FFT algorithm, the first assumption is that the number of discrete complex data points, N, is factorable as

$N = R \times S$ where R and S are relatively prime. Additionally, one can always assume that R is an odd number and S is a power of two.

By using this factorization, the algorithm segments the computations of an N-term FFT in an odd number of identical parallel operations which can be performed independently and concurrently.

A type of multiprocessor system; parallel element processing ensemble, was proposed for implementing the Bergland's parallel FFT algorithm. The architecture of this computer system is similar to single instruction multiple data (SIMD) machines. It consists of one central control processor (CP) and R processing elements (PE). These processing elements can parallelly perform S-point transforms on the data stored in their own memories, under the general control of the control processor. In order to implement the algorithm efficiently, the number of processing elements, R, should be considerably smaller than S. The value of R can be 3,5, or 7, for efficient implementations.

4.1.1 The Algorithm

Assume that N is a product of two factors, R and S. Then, redefining p and q of the Eq.(2.1) using Good's prime-factor algorithm [25], we obtain

$$q = S.q_1 + R.q_2, \qquad \text{mod } N,\ 0 \leqslant q < N,$$

$$q_1 = q.S_R \ (\text{mod } R), \qquad \text{mod } R,\ 0 \leqslant q_1 < R,$$

$$q_2 = q.R_S \ (\text{mod } S), \qquad \text{mod } S,\ 0 \leqslant q_2 < S,$$

and

$$p = S.S_R.p_1 + R.R_S.p_2, \qquad \text{mod } N,\ 0 \leqslant p < N,$$

$$p_1 = p \ (\text{mod } R), \qquad \text{mod } R,\ 0 \leqslant p_1 < R,$$

$$p_2 = p \ (\text{mod } S), \qquad \text{mod } S,\ 0 \leqslant p_2 < S,$$

where S_R and R_S are the solutions of

$$S.S_R = 1 \ (\text{mod } R), \qquad S_R < R,$$

$$R.R_S = 1 \ (\text{mod } S), \qquad R_S < S,$$

respectively.

If we rewrite the Eq.(2.1) with these new definitions for p and q, and applying the decimation-in-frequency method (that is, decomposing p), it becomes

(i) Compute the DFT on R points

$$B(p_1,q_2) = \sum_{q_1=0}^{R-1} w_R^{p_1 q_1} \, a(q_1,q_2) \qquad (4.1)$$

(ii) Compute the DFT on S points

$$A(p_1,p_2) = \sum_{q_2=0}^{S-1} w_S^{p_2 q_2} \, B(p_1,q_2) \qquad (4.2)$$

The Bergland's parallel FFT algorithm consists of mainly these two steps. An example is given in the next section to illustrate the process of computing the N-term FFT.

4.1.2 Use of the Algorithm

Let N = 12 = R x S = 3 x 4. The N-term series of discrete complex data points is a(0),a(1),a(2),...,a(11). This N-term series of the time domain data will be transformed to the frequency domain, that is, to their corresponding Fourier coefficients A(0),A(1),A(2),..,A(11) by applying the steps of the Bergland's parallel FFT algorithm.

Using Good's notation, we have

$$q = 4q_1 + 3q_2, \qquad \text{mod } 12,\ 0 \leqslant q < 12,$$
$$q_1 = q.S_R \ (\text{mod } 3), \qquad \text{mod } 3,\ 0 \leqslant q_1 < 3,$$
$$q_2 = q.R_S \ (\text{mod } 4), \qquad \text{mod } 4,\ 0 \leqslant q_2 < 3,$$

and

$$p = 4S_R.p_1 + 3R_S.p_2, \qquad \text{mod } 12,\ 0 \leqslant p < 12,$$
$$p_1 = p \ (\text{mod } 3), \qquad \text{mod } 3,\ 0 \leqslant p_1 < 3,$$
$$p_2 = p \ (\text{mod } 4), \qquad \text{mod } 4,\ 0 \leqslant p_2 < 4,$$

where

$$S_R = 1,4,7,... < 3, \qquad S_R = 1,$$
$$R_S = 3,7,11,... < 4, \qquad R_S = 3.$$

If we expand the Eq.(4.1), we have S, R-point transforms

$$
\begin{aligned}
&B(0,0) = w_3^{(0)(0)}\, a(0,0) + w_3^{(0)(1)}\, a(1,0) + w_3^{(0)(2)}\, a(2,0),\\
&B(1,0) = w_3^{(1)(0)}\, a(0,0) + w_3^{(1)(1)}\, a(1,0) + w_3^{(1)(2)}\, a(2,0),\\
&B(2,0) = w_3^{(2)(0)}\, a(0,0) + w_3^{(2)(1)}\, a(1,0) + w_3^{(2)(2)}\, a(2,0),\\
&\text{----------}\\
&B(0,1) = w_3^{(0)(0)}\, a(0,1) + w_3^{(0)(1)}\, a(1,1) + w_3^{(0)(2)}\, a(2,1),\\
&B(1,1) = w_3^{(1)(0)}\, a(0,1) + w_3^{(1)(1)}\, a(1,1) + w_3^{(1)(2)}\, a(2,1),\\
&B(2,1) = w_3^{(2)(0)}\, a(0,1) + w_3^{(2)(1)}\, a(1,1) + w_3^{(2)(2)}\, a(2,1),\\
&\text{----------}\\
&B(0,2) = w_3^{(0)(0)}\, a(0,2) + w_3^{(0)(1)}\, a(1,2) + w_3^{(0)(2)}\, a(2,2),\\
&B(1,2) = w_3^{(1)(0)}\, a(0,2) + w_3^{(1)(1)}\, a(1,2) + w_3^{(1)(2)}\, a(2,2),\\
&B(2,2) = w_3^{(2)(0)}\, a(0,2) + w_3^{(2)(1)}\, a(1,2) + w_3^{(2)(2)}\, a(2,2),\\
&\text{----------}\\
&B(0,3) = w_3^{(0)(0)}\, a(0,3) + w_3^{(0)(1)}\, a(1,3) + w_3^{(0)(2)}\, a(2,3),\\
&B(1,3) = w_3^{(1)(0)}\, a(0,3) + w_3^{(1)(1)}\, a(1,3) + w_3^{(1)(2)}\, a(2,3),\\
&B(2,3) = w_3^{(2)(0)}\, a(0,3) + w_3^{(2)(1)}\, a(1,3) + w_3^{(2)(2)}\, a(2,3).\\
&\text{----------}
\end{aligned}
$$

(4.3)*

Since R = 3, we have three processing elements, PE_0, PE_1, PE_2. These processor numbers are written at the left side column of the following equations to show which equation is calculated by which processing element.

*It is a number for the equations between in dashed lines.

Since $q = 4q_1 + 3q_2$, mod 12, $0 \leq q < 12$, we can rewrite the equations in (4.3) as

PE#

PE_0 $\quad B(0,0) = w_3^0\, a(0) + w_3^0\, a(4) + w_3^0\, a(8),$

PE_1 $\quad B(1,0) = w_3^0\, a(0) + w_3^1\, a(4) + w_3^2\, a(8),$

PE_2 $\quad B(2,0) = w_3^0\, a(0) + w_3^2\, a(4) + w_3^4\, a(8),$

PE_0 $\quad B(0,1) = w_3^0\, a(3) + w_3^0\, a(7) + w_3^0\, a(11),$

PE_1 $\quad B(1,1) = w_3^0\, a(3) + w_3^1\, a(7) + w_3^2\, a(11),$

PE_2 $\quad B(2,1) = w_3^0\, a(3) + w_3^2\, a(7) + w_3^4\, a(11),$

PE_0 $\quad B(0,2) = w_3^0\, a(6) + w_3^0\, a(10) + w_3^0\, a(2),$

PE_1 $\quad B(1,2) = w_3^0\, a(6) + w_3^1\, a(10) + w_3^2\, a(2),$

PE_2 $\quad B(2,2) = w_3^0\, a(6) + w_3^2\, a(10) + w_3^4\, a(2),$

PE_0 $\quad B(0,3) = w_3^0\, a(9) + w_3^0\, a(1) + w_3^0\, a(5),$

PE_1 $\quad B(1,3) = w_3^0\, a(9) + w_3^1\, a(1) + w_3^2\, a(5),$

PE_2 $\quad B(2,3) = w_3^0\, a(9) + w_3^2\, a(1) + w_3^4\, a(5).$

(4.4)

Thus, the Eq.(4.1) gives

a(0)		B(0,0)	a(3)		B(0,1)
a(4)	are used to	B(1,0)	a(7)	are used to	B(1,1)
a(8)	compute	B(2,0)	a(11)	compute	B(2,1)
a(6)		B(0,2)	a(9)		B(0,3)
a(10)	are used to	B(1,2)	a(1)	are used to	B(1,3)
a(2)	compute	B(2,2)	a(5)	compute	B(2,3)

Similary, if we expand the Eq.(4.2) and using the relation $p = 4p_1 + 9p_2$, mod 12, $0 \leqslant p < 12$, we have R, S-point transforms

PE#

PE_0

$A(0)=A(0,0)=w_4^0\ B(0,0) + w_4^0\ B(0,1) + w_4^0\ B(0,2) + w_4^0\ B(0,3),$

$A(9)=A(0,1)=w_4^0\ B(0,0) + w_4^1\ B(0,1) + w_4^2\ B(0,2) + w_4^3\ B(0,3),$

$A(6)=A(0,2)=w_4^0\ B(0,0) + w_4^2\ B(0,1) + w_4^4\ B(0,2) + w_4^6\ B(0,3),$

$A(3)=A(0,3)=w_4^0\ B(0,0) + w_4^3\ B(0,1) + w_4^6\ B(0,2) + w_4^9\ B(0,3),$

PE_1

$A(4)=A(1,0)=w_4^0\ B(1,0) + w_4^0\ B(1,1) + w_4^0\ B(1,2) + w_4^0\ B(1,3),$

$A(1)=A(1,1)=w_4^0\ B(1,0) + w_4^1\ B(1,1) + w_4^2\ B(1,2) + w_4^3\ B(1,3),$

$A(10)=A(1,2)=w_4^0\ B(1,0) + w_4^2\ B(1,1) + w_4^4\ B(1,2) + w_4^6\ B(1,3),$

$A(7)=A(1,3)=w_4^0\ B(1,0) + w_4^3\ B(1,1) + w_4^6\ B(1,2) + w_4^9\ B(1,3),$

PE_2

$A(8)=A(2,0)=w_4^0\ B(2,0) + w_4^0\ B(2,1) + w_4^0\ B(2,2) + w_4^0\ B(2,3),$

$A(5)=A(2,1)=w_4^0\ B(2,0) + w_4^1\ B(2,1) + w_4^2\ B(2,2) + w_4^3\ B(2,3),$

$A(2)=A(2,2)=w_4^0\ B(2,0) + w_4^2\ B(2,1) + w_4^4\ B(2,2) + w_4^6\ B(2,3),$

$A(11)=A(2,3)=w_4^0\ B(2,0) + w_4^3\ B(2,1) + w_4^6\ B(2,2) + w_4^9\ B(2,3).$

(4.5)

Thus, the Eq.(4.2) gives

B(0,0)		A(0)	B(1,0)		A(4)
B(0,1)	are used to	A(9)	B(1,1)	are used to	A(1)
B(0,2)	compute	A(6)	B(1,2)	compute	A(10)
B(0,3)		A(3)	B(1,3)		A(7)

B(2,0)		A(8)
B(2,1)	are used to	A(5)
B(2,2)	compute	A(2)
B(2,3)		A(11)

The summations have been expanded briefly in Eq.(4.4) and Eq.(4.5). The example implementation can now be considered in more detail on the proposed hardware.

The Bergland's parallel FFT algorithm for N = 12 = R x S = 3 x 4 on a PEPE multiprocessor system is illustrated in Fig.4.1. The details of the implementation can be explained by using this configuration.

Each processing element consists of an arithmetic processor and a memory module. The communications between the central control processor and the PE's are achieved through the communication network. There is no data transfer between PE's, and they can only execute the instructions coming from the control processor.

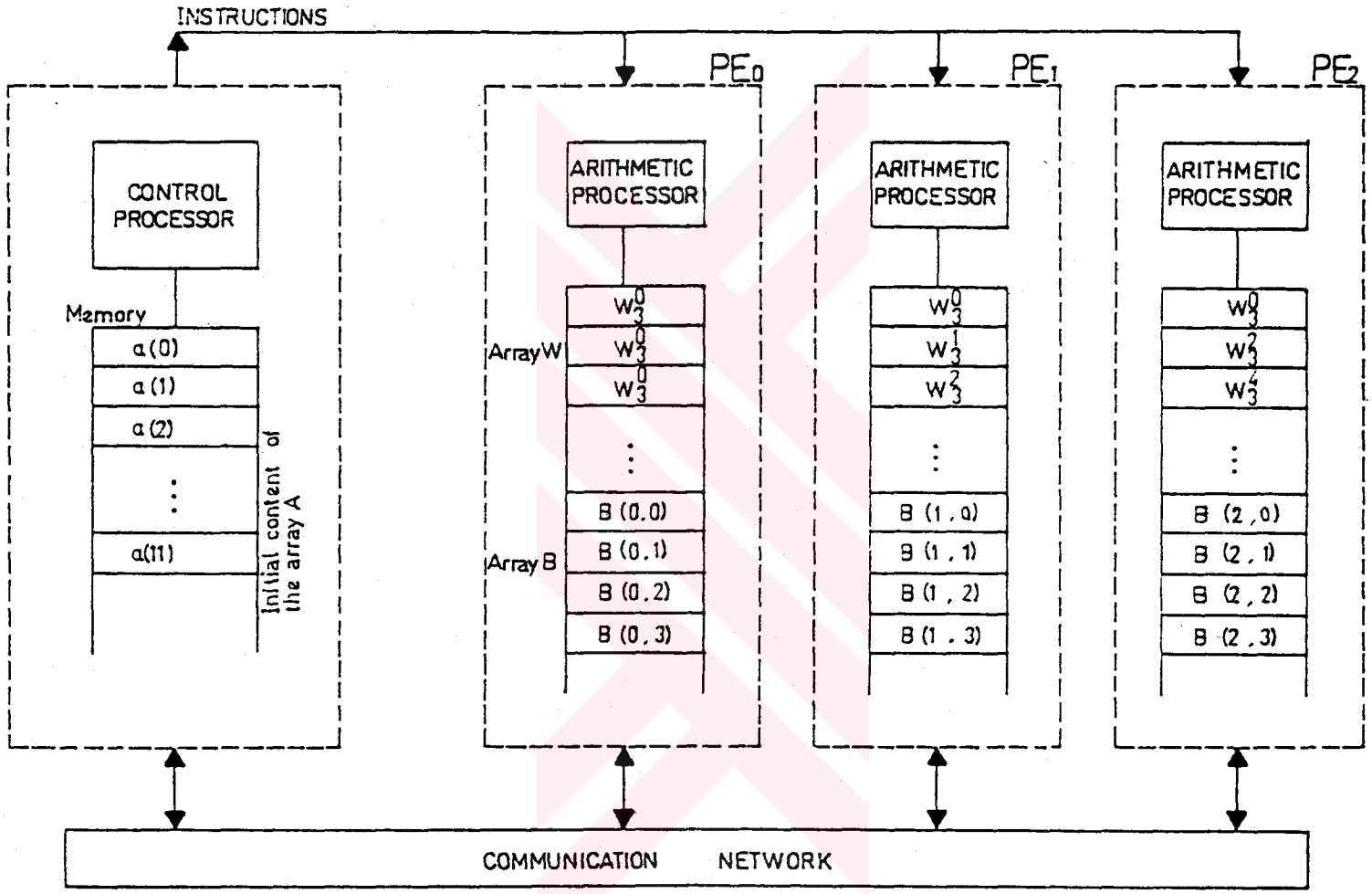


Figure 4.1 Bergland's parallel FFT algorithm on a parallel element processing ensemble.

The parallel implementation of the algorithm works as follows.

(1) The N-term series of the time domain data a(0),a(1),a(2),...,a(11) are initially stored in the array A of control processor's memory. At the end of the computations they will be replaced by their corresponding Fourier coefficients, that is A(0),A(1),A(2),...,A(11).

(2) This N-term series is prensented sequentially by the control processor using the communication network, to the entire ensemble of R processing elements. This is simply a broadcast operation.

(3) Each of the sums of Eq.(4.1) are accumulated during this time in the array B of each processing element's memory. For example, the first processor accumulates B(0,0),B(0,1),B(0,2), and B(0,3), of the S, R-point transforms indicated by Eq.(4.1), the second processor accumulates B(1,0),B(1,1),B(1,2), and B(1,3) and the third processor accumulates B(2,0), B(2,1),B(2,2), and B(2,3).

- Each processing element accumulates these coefficients by simply accepting every sample a(q) as it is broadcast to all of the processors, then multiplying it by the appropriate power of w, stored in its w

array, and finally adding the resulting product to the correct location in its array B. This step is done by using the equations in (4.4). The left side column of (4.4) shows the numbers of the processing elements.

(4) When all the input terms have been processed, the processor's memories have accumulated the intermediate results as depicted in Fig.4.1. These intermediate results, that is the set of B(i,j) terms can then be transformed independently and concurrently. Note that these R, S-point parallel transforms are shown in the equations (4.5) and they can be evaluated via any standard FFT algorithm.

(5) The last step is to read out the evaluated Fourier coefficients of the N-term series, in ascending order. This is done by noting the correspondence between the A(p) notation and the $A(p_1,p_2)$ notation where $p_1 = p \pmod{R}$ and $p_2 = p \pmod{S}$.

In the example, for $N = 12 = R \times S = 3 \times 4$, results of the computations will be stored in the array B of each processing element's memory, as shown in Fig.4.2.

p_1: PE number

p_2: address in array B	0	1	2
0	A(0)	A(4)	A(8)
1	A(9)	A(1)	A(5)
2	A(6)	A(10)	A(2)
3	A(3)	A(7)	A(11)

Figure 4.2 Location of the A(p)'s where p is a function of p_1 and p_2.

Reading the A(p) terms out, in ascending frequency order, is done by forming a (mod R) counter which identifies the value of p_1 and a (mod S) counter which identifies the value of p_2, that is

$k_1 = 0 \pmod 3$
$k_2 = 0 \pmod 4$ ⟶ count = 0 ⟶ A(0),

$k_1 = 1 \pmod 3$
$k_2 = 1 \pmod 4$ ⟶ count = 1 ⟶ A(1),

.
.
.

$k_1 = 11 \pmod 3 = 2$
$k_2 = 11 \pmod 4 = 3$ ⟶ count = 11 ⟶ A(11).

4.2. DECIMATION-IN-TIME RADIX-2 FFT ALGORITHM

In the Bergland's case, the original derivation of the fast Fourier transform algorithm is modified for parallel implementation. However, the fast Fourier transform, as it is, is a highly parallel algorithm, so there is no need for exploring further parallelism in it. In the following sections, the original form of the decimation-in-time radix-2 FFT algorithm is considered on a loosely coupled multiprocessor system.

If the factorization of the discrete Fourier transform formula is carried out using factor 2 as a base or a radix, then the derived FFT algorithm takes a prefix-name, "radix-2". Furthermore, in the same factorization of the DFT, if the time sequence index "q" is decomposed, then the resulting algorithm is also called as "decimation-in-time".

Since it has many distinct advantages in its computation, the radix-2 FFT algorithm is the most popular and developed one among other FFT algorithms.

4.2.1 The Computer System

The general architecture of a loosely coupled multiprocessor system is depicted in Fig.4.3.

As in the PEPE system used for implementing the Bergland's algorithm, each processing element consists of

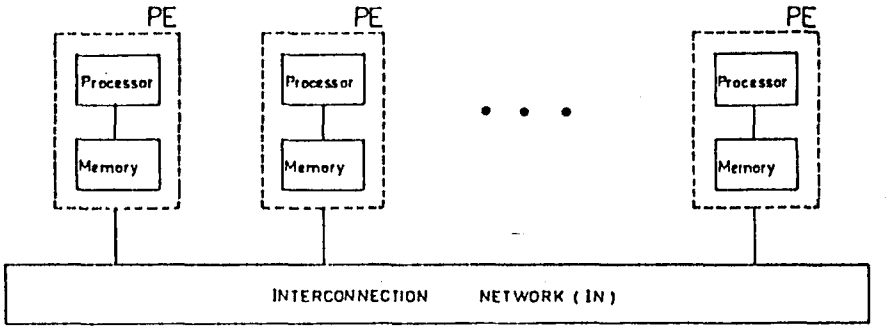


Figure 4.3 A loosely coupled multiprocessor.

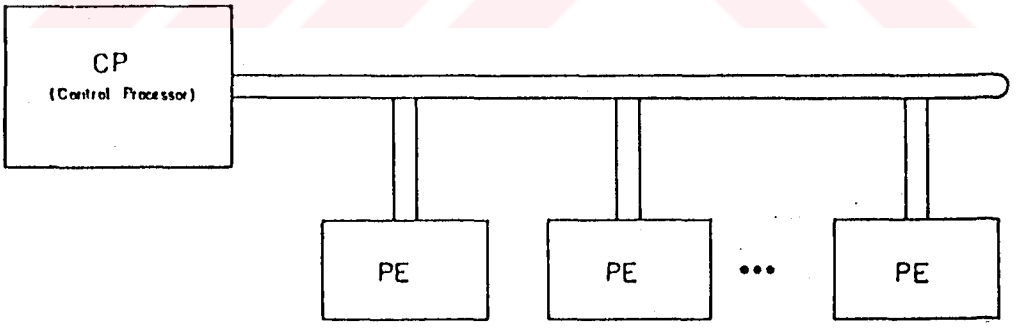


Figure 4.4 A shared bus computer.

a processor and a memory module. However, in a loosely coupled multiprocessor system the PE's are intellegent enough to execute their own programs and the communication between the processing elements are achieved through IN. We additionally assume that there is a central control processor that supervises the operation and switches the IN for interprocessor communications.

A type of loosely coupled multiprocessor system, "computer with shared bus", is illustrated in Fig.4.4. This computer system is specifically considered here to implement the decimation-in-time radix-2 FFT algorithm.

In the computer with shared bus, there are P processing elements where $P = 2^k$. Each PE has its own local memory and the bus operates in a time multiplexed mode to transfer data from one PE to another. The central control processor enables connection between two PE's. Once the connection is established, the data transfer begins with a block of data.

4.2.2 The Algorithm

The general form of the decimation-in-time radix-2 FFT algorithm is stated as follows.

When $N = 2^m$, p and q of the Eq.(2.1) can be represented in binary form (the name "radix-2") as

$$p = 2^{m-1}p_{m-1} + 2^{m-2}p_{m-2} + \ldots + p_0, \tag{4.6}$$

$$q = 2^{m-1}q_{m-1} + 2^{m-2}q_{m-2} + \ldots + q_0. \tag{4.7}$$

Using this representation, we can rewrite the Eq.(2.1) as

$$A(p_{m-1}, p_{m-2}, \ldots, p_0) = \sum_{q_0} \sum_{q_1} \cdots \sum_{q_{m-1}} w^{pq}\, a(q_{m-1}, q_{m-2}, \ldots, q_0). \tag{4.8}$$

Note that the single summation in Eq.(2.1) must now be replaced by m summations in order to enumerate all the bits of the binary representation of q. The term w^{pq} in Eq.(4.8) is

$$w^{pq} = w^{(2^{m-1}p_{m-1}+2^{m-2}p_{m-2}+\ldots+p_0)(2^{m-1}q_{m-1}+2^{m-2}q_{m-2}+\ldots+q_0)}. \tag{4.9}$$

Here, if we decompose q, that is,

$$w^{pq} = w^{p(2^{m-1}q_{m-1})}\, w^{p(2^{m-2}q_{m-2})} \cdots w^{pq_0}, \tag{4.10}$$

then the algorithm is called as "decimation-in-time", whereas if we decompose p

$$w^{pq} = w^{(2^{m-1}p_{m-1})q}\, w^{(2^{m-2}p_{m-2})q} \cdots w^{p_0 q}, \tag{4.11}$$

it is called as "decimation-in-frequency".

Performing each of the summations in Eq.(4.8) seperately and labeling the stages of intermediate results, we obtain

$$a_1(p_0,q_{m-2},\ldots,q_0) = \sum_{q_{m-1}} w^{2^{m-1}(p_0 q_{m-1})}\, a_0(q_{m-1},\ldots,q_0),$$

$$a_2(p_0,p_1,q_{m-3},\ldots,q_0) =$$

$$\sum_{q_{m-2}} w^{(2p_1+p_0)2^{m-2}q_{m-2}}\, a_1(p_0,q_{m-2},\ldots,q_0),$$

$$\vdots \qquad (4.12)$$

$$a_m(p_0,p_1,\ldots,p_{m-1}) =$$

$$\sum_{q_0} w^{(2^{m-1}p_{m-1}+2^{m-2}p_{m-2}+\ldots+p_0)q_0}\, a_{m-1}(p_0,p_1,\ldots,q_0),$$

$$A(p_{m-1},p_{m-2},\ldots,p_0) = a_m(p_0,p_1,\ldots,p_{m-1}).$$

This set of iterative equations represents the original formulation of the decimation-in-time radix-2 FFT algorithm. The final results $a_m(p_0,p_1,\ldots,p_{m-1})$ which are obtained from the last summation of Eq.(4.12), are in bit-reversed order with respect to the desired values

$A(p_{m-1},p_{m-2},\ldots,p_0)$. This is simply called as "scrambling" that results from the FFT algorithm. The final step in computing the FFT is to unscramble the results as shown in the last equation of Eq.(4.12).

However, if the input data is given as in scrambled order, that is in bit-reversed order, then the output data of the FFT does not need any unscrambling operation, because, in this case the FFT gives the results in desired natural order.

$$a_1(q_0,\ldots,q_{m-2},p_0) = \sum_{q_{m-1}} w^{2^{m-1}(p_0 q_{m-1})}\, a_0(q_0,\ldots,q_{m-1}),$$

$$a_2(q_0,\ldots,q_{m-3},p_1,p_0) =$$

$$\sum_{q_{m-2}} w^{(2p_1+p_0)2^{m-2}q_{m-2}}\, a_1(q_0,\ldots,q_{m-2},p_0),$$

$$\vdots \qquad (4.13)$$

$$a_m(p_{m-1},\ldots,p_1,p_0) =$$

$$\sum_{q_0} w^{(2^{m-1}p_{m-1}+2^{m-2}p_{m-2}+\ldots+p_0)q_0}\, a_{m-1}(q_0,\ldots,p_1,p_0).$$

This second method, that is, bit-reversed input data and natural ordered results is considered in the following sections.

4.2.3 Use of the Algorithm

In this section, an example is given to illustrate the process of computing the N-term FFT. Let $N = 16 = 2^m = 2^4$. The N-term series containing the bit-reversed input data is

$$a(0), a(1), a(2), \ldots, a(15),$$

and the corresponding Fourier coefficients containing the natural ordered results are

$$A(0), A(1), A(2), \ldots, A(15).$$

For $N = 2^4$, p and q is represented as 4-bit binary numbers,

$$p = 2^3 p_3 + 2^2 p_2 + 2 p_1 + p_0,$$
$$q = 2^3 q_3 + 2^2 q_2 + 2 q_1 + q_0.$$

Using Eq.(4.13), we obtain following equations

$$a_1(q_0, q_1, q_2, p_0) = \sum_{q_3} w_2^{p_0 q_3} \, a_0(q_0, q_1, q_2, q_3),$$

$$a_2(q_0, q_1, p_1, p_0) = \sum_{q_2} w_4^{(2p_1 + p_0) q_2} \, a_1(q_0, q_1, q_2, p_0),$$

$$a_3(q_0,p_2,p_1,p_0) = \sum_{q_1} w_8^{(4p_2+2p_1+p_0)q_1}\, a_2(q_0,q_1,p_1,p_0),$$

$$a_4(p_3,p_2,p_1,p_0) = \sum_{q_0} w^{(8p_3+4p_2+2p_1+p_0)q_0}\, a_3(q_0,p_2,p_1,p_0).$$

(4.14)

Because of the binary nature of the parameters q_3,q_2,q_1, and q_0 each of the sums in Eq.(4.14) consists of two terms only. For example, consider the first equation of Eq.(4.14)

$$a_1(q_0,q_1,q_2,p_0)$$

$$= \sum_{q_3} w_2^{p_0q_3}\, a_0(q_0,q_1,q_2,q_3),$$

$$= w_2^{(p_0)(0)}\, a_0(q_0,q_1,q_2,0) + w_2^{(p_0)(1)}\, a_0(q_0,q_1,q_2,1),$$

$$= w_2^0\, a_0(q_0,q_1,q_2,0) + w_2^{p_0}\, a_0(q_0,q_1,q_2,1).$$

Furthermore, since p_0 takes values 0 or 1 only, the computations of the two terms

$$a_1(q_0,q_1,q_2,0) = w_2^0\, a_0(q_0,q_1,q_2,0) + w_2^0\, a_0(q_0,q_1,q_2,1),$$

and

$$a_1(q_0,q_1,q_2,1) = w_2^0\, a_0(q_0,q_1,q_2,0) + w_2^1\, a_0(q_0,q_1,q_2,1),$$

can be diagrammed as shown in Fig.4.5.

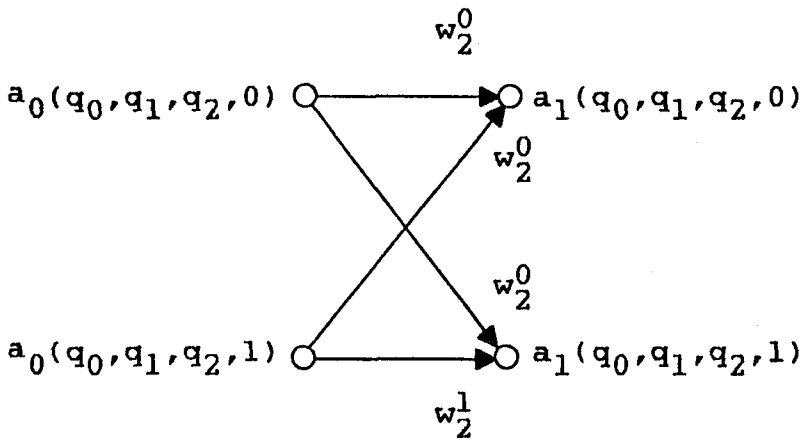


Figure 4.5 Butterfly computation of decimation-in-time radix-2 FFT algorithm.

This computational diagram is called as the "butterfly" computation of radix-2 FFT algorithm in Eq.(4.13). Similar diagrams can be drawn for each of the sums in Eq.(4.14).

4.2.4 Parallel Implementation of the Algorithm

In a decimation-in-time N-point radix-2 FFT algorithm, m stages of computations are required with N/2 butterfly computations per stage, where $m = \log_2 N$. With P number of PE's, N/2P butterfly computations are carried out in each PE per stage. The partitioning of a 16 point FFT with 2,4,

and 8, PE's are shown in Fig.4.6 (a), (b), and (c), respectively.

Note that at each stage of interprocessor communication, a PE keeps one datum of each butterfly in its memory for the next stage computation and sends the other data to some other PE.

The parallel implementation of the algorithm works as follows.

(1) The control processor sends the same amount of input data which is N/P to each of the P available PE's.
(2) Each PE computes N/2P butterflies per stage until (m-k) stages.
(3) PE_i sends N/2P data items to PE_j where $1 \leqslant i,j \leqslant P$.
(4) Each PE computes N/2P butterflies.
(5) Repeat from step (3) until m stages are complete.
(6) Each PE sends its resulting data back to control processor.

4.2.5 General Features of the Algorithm

In the decimation-in-time radix-2 FFT algorithm considered here, the inputs are bit-reversed and the outputs are ordered. We assume that the input data a(q) and output data A(p) are numbered from 0 to N-1 and the PE's are numbered from 1 to P where $P = 2^k$. The following results are obtained.

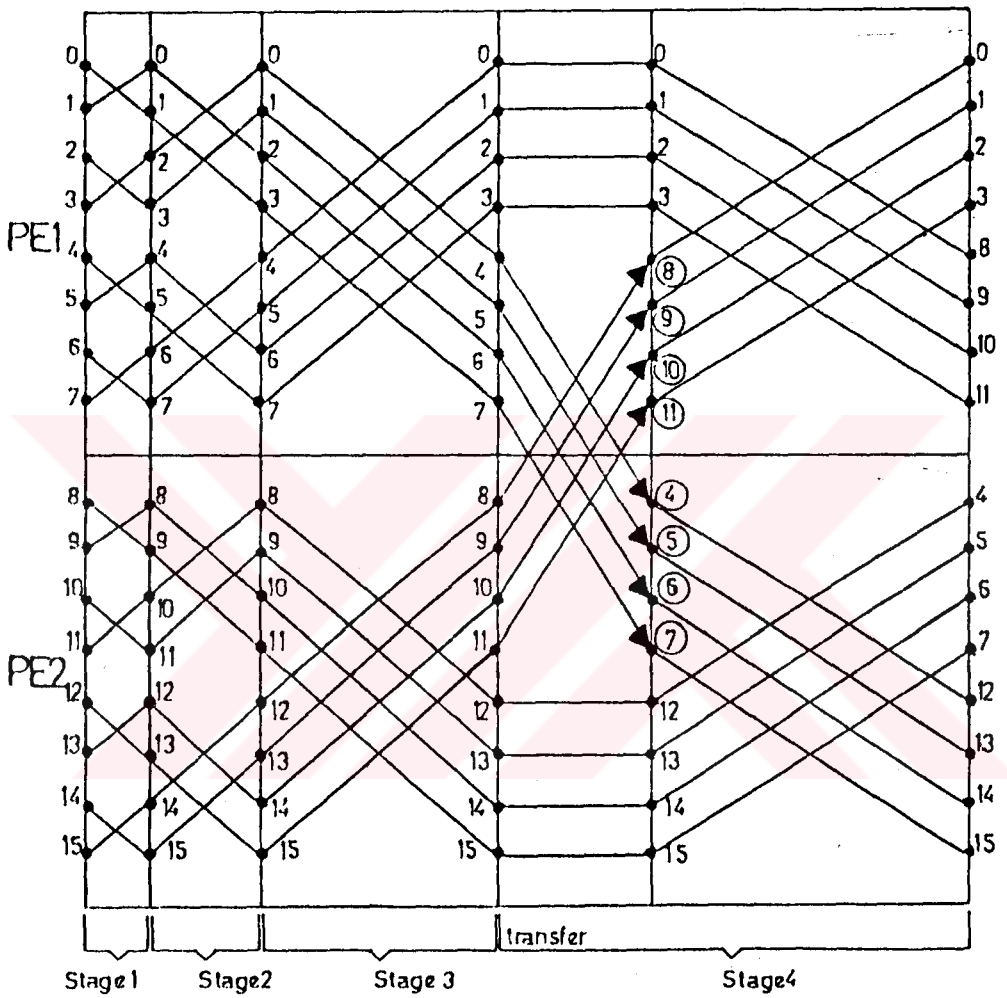


Figure 4.6(a) 16-point decimation-in-time radix-2 FFT computation with P = 2.

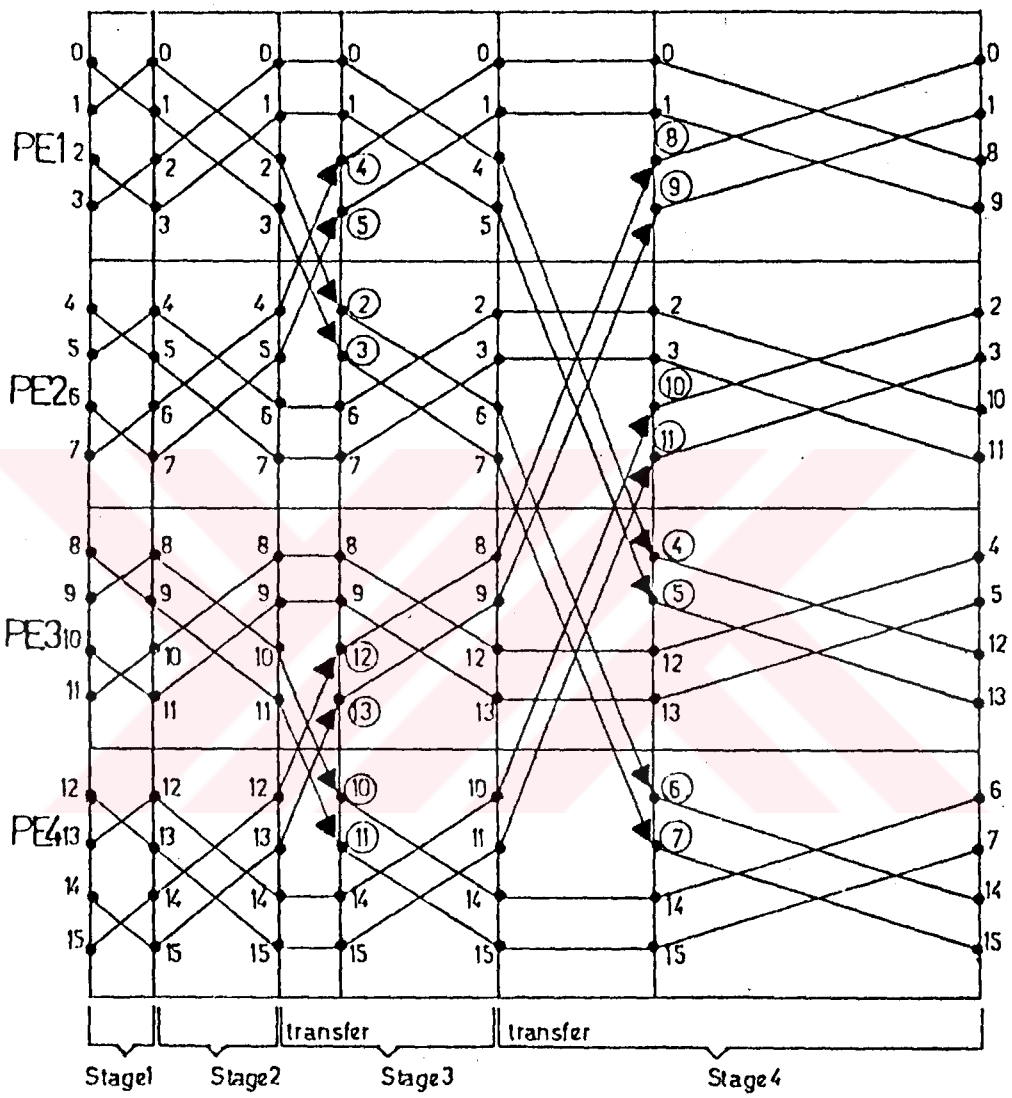


Figure 4.6(b) 16-point decimation-in-time radix-2 FFT computation with P = 4.

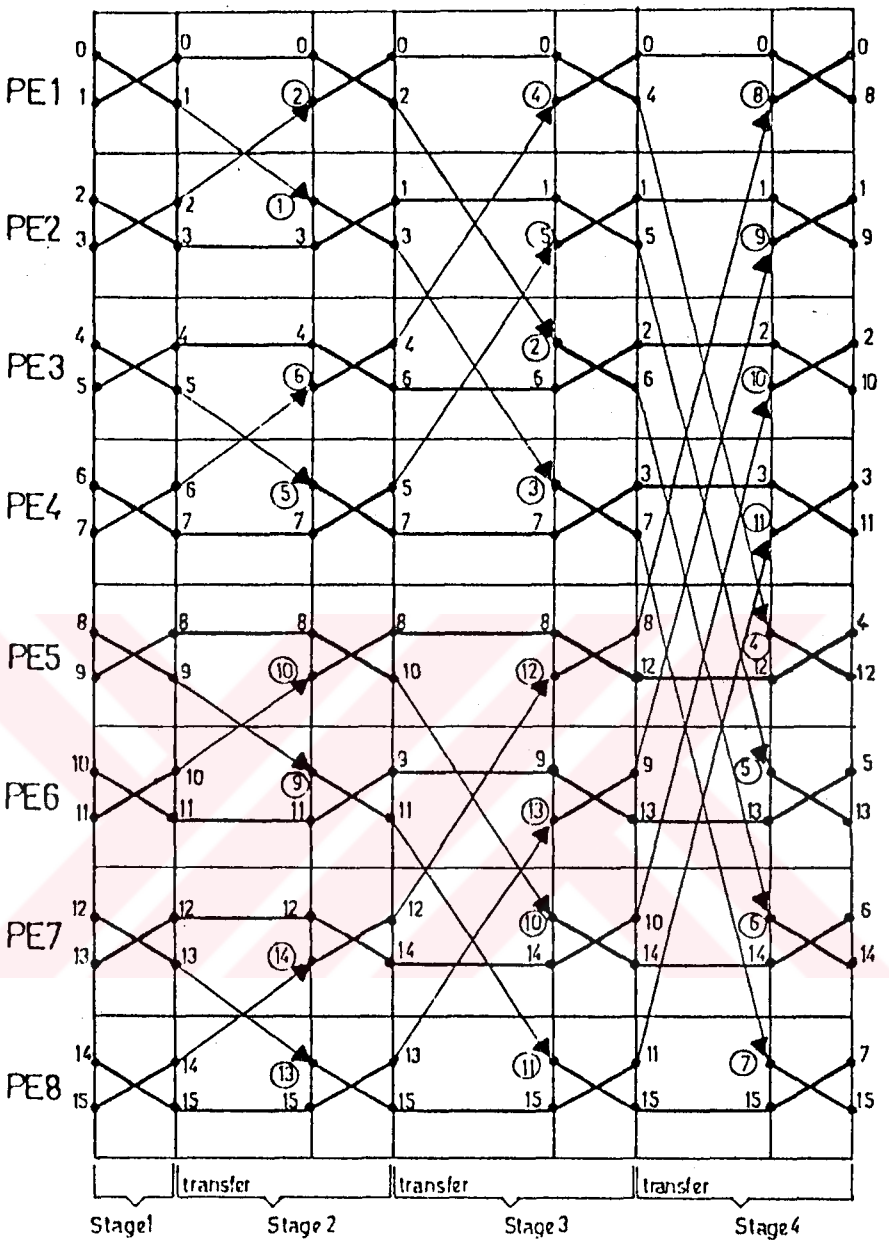


Figure 4.6(c) 16-point decimation-in-time radix-2 FFT computation with P = 8.

(1) For a decimation-in-time N-point radix-2 FFT algorithm with P available PE's, the number of butterfly computations per PE per stage is N/2P. Assuming each butterfly computation takes one unit of time, computation time per stage is N/2P. The rest of the butterflies in that stage are also simultaneously computed by other PE's. Hence, for m stages, the total time of butterfly computations is m(N/2P) units.

(2) The amount of data per PE is N/P. The i th PE

will contain input data ◀ (N/P)(i-1) to (N/P)i - 1

and produce output data ◀ (N/2P)(i-1) to (N/2P)i - 1 and (N/2P)(i-1) + (N/2) to ((N/2P)i - 1) + (N/2)

Note that these input-output sequences for N = 16 and P = 2,4, and 8, are illustrated in Fig.4.6(a),(b) and (c), respectively.

(3) There is no data transfer for stages s $\leq$ (m-k). After (m-k) stages, the PE's are grouped for data transfer with a difference of 1,2,4,8,... until m stages are

complete; so the difference grows by powers of two. For the s th stage of computation, the difference between PE_i and PE_j with respect to s is given by

$$|i-j| = 0 \qquad \text{for} \qquad 1 \leqslant s \leqslant (m-k),$$

and

$$|i-j| = P.2^{(s-1)} / N \qquad \text{for} \qquad (m-k) < s \leqslant m.$$

CHAPTER 5.

SIMULATED IMPLEMENTATIONS

In this chapter, we describe a set of FORTRAN programs that simulate the two different parallel FFT algorithms with concurrent programming. The source listings of the programs are given in Appendix-A.

This set of simulation programs has been implemented on an available VAXcluster system which consists of two interconnected VAX 11/780 computers. The Bergland's parallel FFT algorithm has been simulated by using only one node of the VAXcluster system and the decimation-in-time radix-2 FFT algorithm by using both nodes of the system.

Loosely coupled VAXcluster systems are configured using a new method of interconnecting or "clustering" VAX processors. Communications within the VAXcluster systems are handled through a central interconnect data channel which is called as the high speed Computer Interconnect or "CI".

In addition to the CI architecture, the available VAXcluster system has an ethernet which provides the accessing to Local Area Network (LAN). Ethernet combines with the CI and terminal servers to provide a total communication solution for VAXcluster systems. Two-node VAXcluster system interconnected by the CI and ethernet is shown in Fig.5.1.

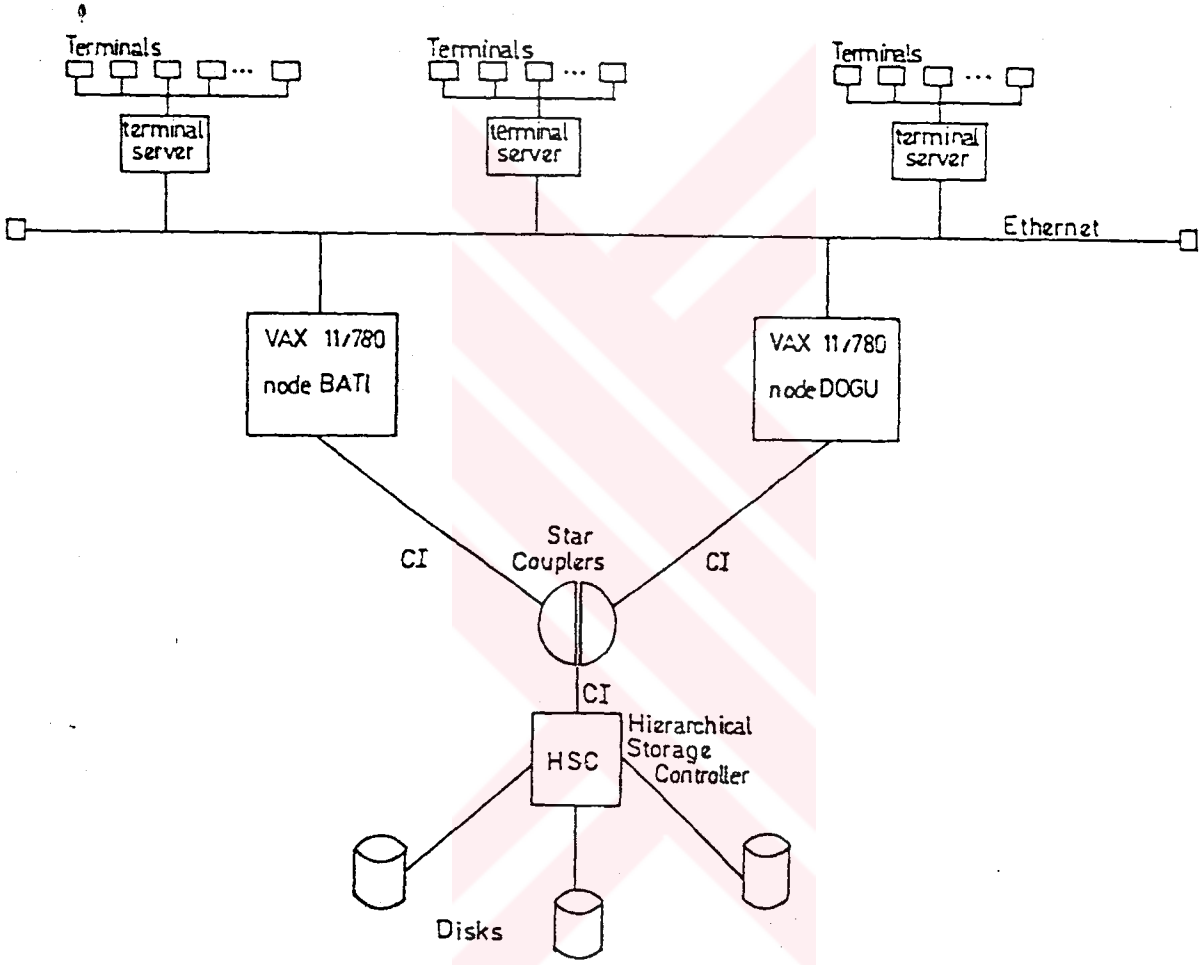


Figure 5.1 Two-node VAXcluster with ethernet.

The VAX/VMS operating system provides a high performance environment for concurrent multiuser time sharing operations and offers a set of very valuable routines for concurrent programming in implementing with high level languages. Both algorithms have been implemented by using these routines and some other programmed system services, such as interprocess communication services and intersystem communication services. The VAX/VMS manuals related with these subjects are referenced between in [46]-[53].

5.1. SIMULATION OF BERGLAND'S PARALLEL FFT ALGORITHM

The simulated implementation of Bergland's parallel FFT algorithm on a VAX 11/780 computer is illustrated in Fig.5.2. The following 5 programs are used for this simulation.

- CONTROL_PROCESSOR (a program executed in a parent process to simulate the central control processor)
  - GET_POINTS (a subprogram to generate input data)
  - PUT_RESULTS (a subprogram to output results)
- PROCESSING_ELEMENT (a program executed in each concurrent subprocess to simulate a PE)

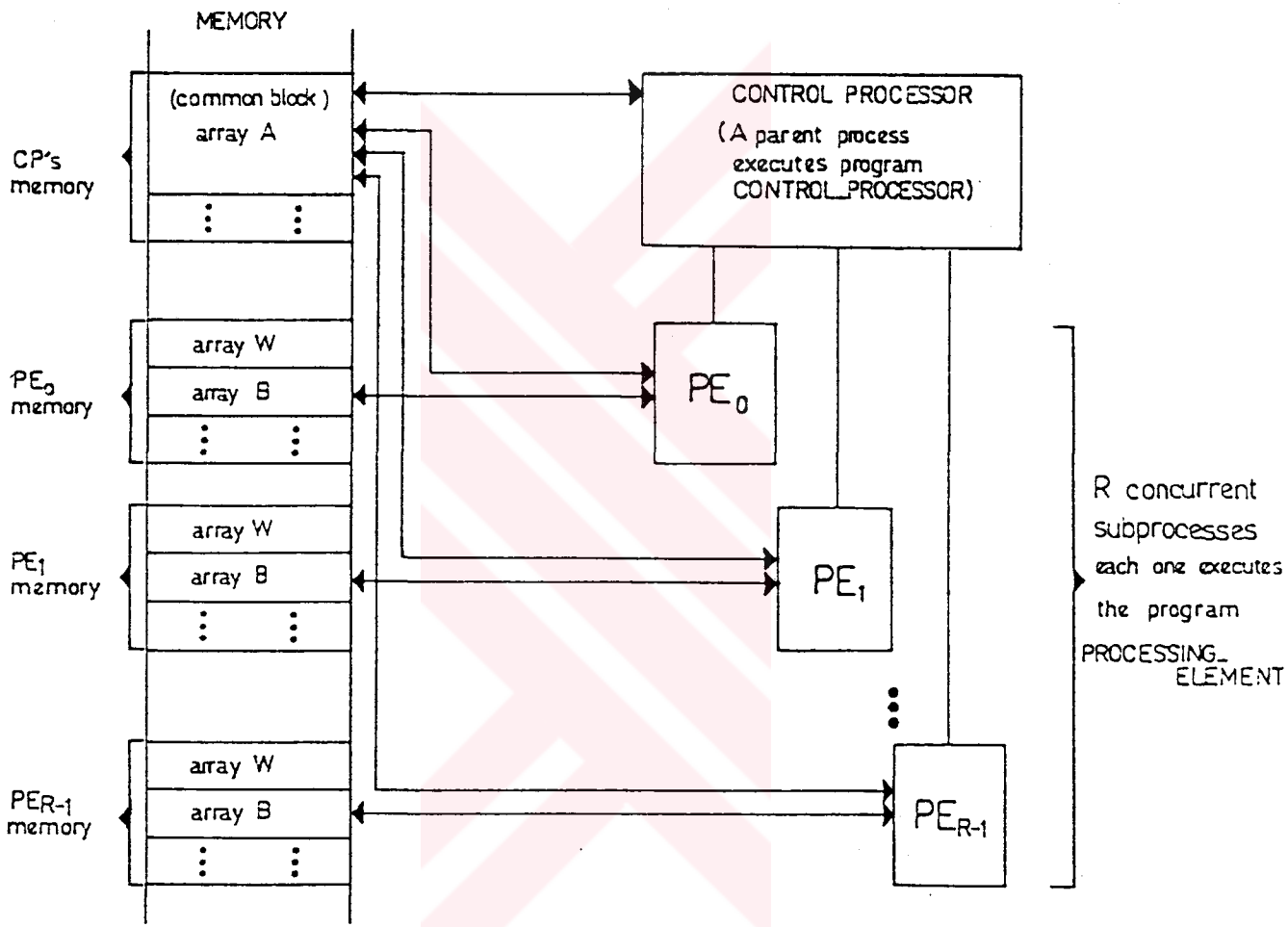


Figure 5.2 Simulation of Bergland's parallel FFT algorithm on a VAX 11/780.

. FFT4_2 (a subprogram to calculate S-point FFT)

A parent process executing the program CONTROL_PROCESSOR simulates the central control processor and sequentially creates R number of concurrent subprocesses (maximum 9). Each one of these concurrent subprocesses simulates a PE by executing the program PROCESSING_ELEMENT.

The subprogram GET_POINTS is called by the CONTROL_PROCESSOR and generates the input data using one of the time domain waveforms exp(-x), sin x, and cos x, and the step size in the interval $0 < h \leq 1$.

The subprogram PUT_RESULTS is also called by the CONTROL_PROCESSOR and has two choices as Screen and Disk to where the results will be output.

The subprogram FFT4_2 is called by each one of the R PROCESSING_ELEMENT's to perform the Eq.(4.2) independently and concurrently. It is an optimized radix-"4+2" standard FFT algorithm.

As shown in Fig.5.2, the communications between CONTROL_PROCESSOR and PROCESSING_ELEMENT's are achieved by using a previously installed common block. In the VAX/VMS operating system, we can use an installed common block for interprocess communication or for allowing two or more processes to access the same data simultaneously.

Note that, the desired common block is in the program file IN_PLACE.FOR (Appendix-A).

In addition, common event flags are used to synchronize the events between PROCESSING_ELEMENT's. This synchronization is especially required for communicating between CONTROL_PROCESSOR and PROCESSING_ELEMENT's using the installed common block.

5.2. SIMULATION OF DECIMATION-IN-TIME RADIX-2 FFT ALGORITHM

As mentioned previously, the decimation-in-time radix-2 FFT algorithm has been simulated by using both nodes of the available VAXcluster system (Fig.5.1). The following 5 programs are used for this simulation.

- CONTROL_PROCESSOR (a program executed in a parent process in local node to simulate the central control processor)
  - GET_POINTS (a subprogram to generate input data)
  - BIT_REVERSED (a subprogram to take the bit-reversed order of input data)
  - PUT_RESULTS (a subprogram to output results)
- PROCESSING_ELEMENT (a program executed in each local subprocess and remote process to simulate a PE)

Note that, the four programs have the same name with

different content by the simulation programs of the Bergland's algorithm.

Each of the two interconnected VAX 11/780 computers has a node name as "node BATI" and "node DOGU". In the simulation programs, node BATI is used as a local node and node DOGU as a remote node.

In each node, the same number of concurrent processes (maximum total is 8) are created sequentially by a parent process which executes the program CONTROL_PROCESSOR to simulate the central control processor. If a process is created in local node, it is called as local subprocess, whereas if it is created at remote node by establishing a logical link between two nodes, then it is called as remote process.

Each local subprocess or remote process executes the same program; PROCESSING_ELEMENT, to simulate a PE. In the following, the terms; local subprocess and remote process are used synonymously with local PROCESSING_ELEMENT and remote PROCESSING_ELEMENT, respectively.

All subprograms are called by the CONTROL_PROCESSOR. Tasks of the GET_POINTS and PUT_RESULTS are the same as in the Bergland's case. However, the BIT_REVERSED is used to take the bit-reversed order of input data generated by the GET_POINTS.

In this simulation, synchronous mailboxes are used instead of an installed common block for interprocess communication. Mailboxes are pseudo input-output devices located in memory, and synchronous mailbox means that read

or write to a mailbox, and then wait for the cooperating process to perform the opposite operation.

In the simulation programs, mailboxes are used both for communicating between PROCESSING_ELEMENT's in the same node (it doesn't matter the node is local or remote) and for CONTROL_PROCESSOR and PROCESSING_ELEMENT's in local node.

In addition to the mailboxes for interprocess communication, the logical links, established between the CONTROL_PROCESSOR in local node and the PROCESSING_ ELEMENT's at remote node while the CONTROL_PROCESSOR creating remote processes are used for intersystem communication.

The interprocess communications with the synchronous mailboxes and the intersystem communications with the logical links are shown in Fig.5.3 (a),(b), and (c), for P = 2,4, and 8, respectively. Functions of the mailboxes, shown in Fig.5.3, are as follows:

- The mailbox CP in local node is used to transfer the identification data of each local PROCESSING_ ELEMENT.
- The mailboxes, MBX0#, in local node are used for communicating between CONTROL_PROCESSOR and each local PROCESSING_ELEMENT.
- The other mailboxes, MBX##, both in local and remote node are used for communicating between PROCESSING_ELEMENT's in the same node.

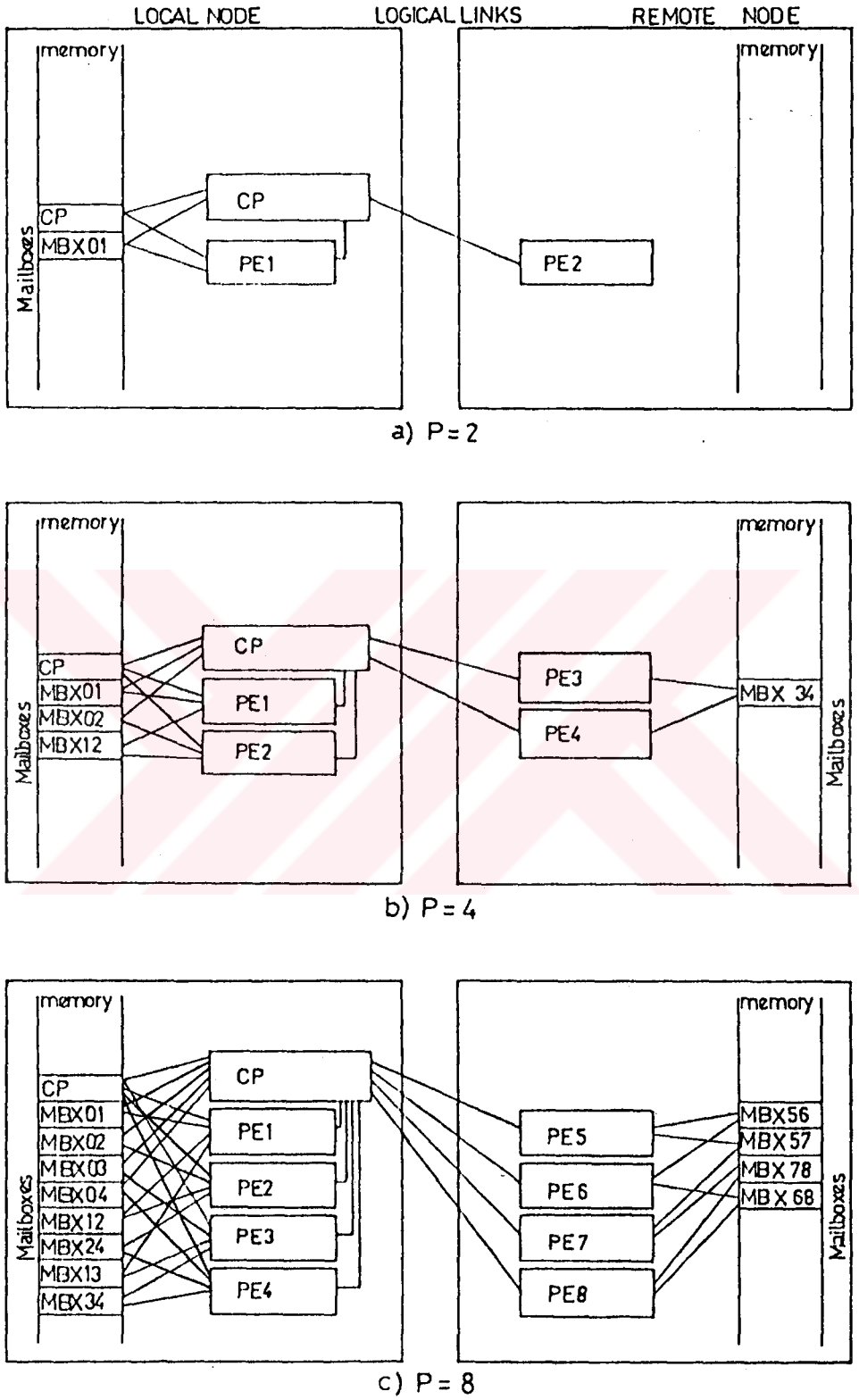


Figure 5.3 Interprocess and intersystem communications for the VAXcluster simulation of decimation-in-time radix-2 FFT algorithm.

The communications between a local PROCESSING_ELEMENT and a remote PROCESSING_ELEMENT is done as follows.

As it is seen in Section 4.2. that there is no data transfer for the stages s $\leq$ (m-k), and after (m-k) stage, there are k transfer stages in the computation. In our simulation, up to k th transfer stage, the data transfer between PROCESSING_ELEMENT's in the same node is done by using the mailboxes, MBX##, in that node. However, in the k th transfer stage, the data transfer between PROCESSING_ELEMENT's in different nodes is required. This synchronous intersystem data transfer is done as follows.

- As in the each tranfer stage, the PROCESSING_ELEMENT(i) determines its communication partner PROCESSING_ELEMENT(j) also in the k th transfer stage.
- Assume that PROCESSING_ELEMENT(i) is a local subprocess and PROCESSING_ELEMENT(j) is a remote process. After the step; determining communication partner, local PROCESSING_ELEMENT(i) knows its partner remote PROCESSING_ELEMENT(j), and also remote PROCESSING_ELEMENT(j) knows local PROCESSING_ELEMENT(i).
- Local PROCESSING_ELEMENT(i) puts the data to the mailbox MBX0i and waits for the CONTROL_PROCESSOR to read out the data from that mailbox.
- CONTROL_PROCESSOR gets the data from MBX0i, and then PROCESSING_ELEMENT(i) wakes up to continue.

- CONTROL_PROCESSOR outputs the data to remote PROCESSING_ELEMENT(j) synchronously over the established logical link between CONTROL_PROCESSOR and PROCESSING_ELEMENT(j).
- Finally, a reverse operation begins with this last step to transfer the data from PROCESSING_ELEMENT(j) to PROCESSING_ELEMENT(i).

CHAPTER 6.

RESULTS AND DISCUSSIONS

A complex multiplication, requiring four real multiplications and two real additions, is a relatively slow operation on most computers. The fact that the FFT of N points requires $N.\log_2 N$ complex multiplications while the straightforward approach requires N^2, providing a clear choice of algorithms for serial computers.

The number of complex multiplications still remains important for parallel computers, but several other factors, such as the data transfer time between processing elements, becomes equally significant. In a highly parallel algorithm like the FFT, the computation time in various multiprocessor organizations remains the same while the communication overhead due to interprocessor data transfer is extremely important. It decides the actual performance of the algorithm, since the total time required to move data to the appropriate processing elements can be larger than the computation time, on certain multiprocessor architectures.

In the following sections, the execution times both for the simulation of Bergland's parallel FFT algorithm and the simulation of decimation-in-time radix-2 FFT algorithm are considered for various values of N, in different number of processing elements. Note that, the execution time is a total of computation time and the data transfer time of the specified simulation.

Note also that, the execution times given in the following sections for both simulations can be obtained by using anyone of the time domain waveforms: exp(-x), sin x, and cos x, as input data. Because, the content of input data does not change the execution time of the simulation.

6.1. BERGLAND'S PARALLEL FFT ALGORITHM

The total number of complex multiplications that must be performed in sequence using this algorithm is the sum of the operations required in evaluating Eq.(4.1) and Eq.(4.2), that is,

$$N - S = R \times S - S = (R - 1)S, \tag{6.1}$$

and

$$(\log_2 S / 3 - 1)S + 1, \tag{6.2}$$

respectively. Thus the algorithm requires that a total of

$$(R + \log_2 S / 3 - 2)S + 1, \tag{6.3}$$

complex multiplications must be performed sequentially per execution.

The simulation of Bergland's parallel FFT algorithm on a VAX 11/780 computer runs without requiring any data transfer, since the necessary data both for CONTROL_PROCESSOR and PROCESSING_ELEMENT's is stored in a common

memory area. Thus, the execution times given in this section, contain only the computation time of the simulation.

Some numerical results for the FFT simulation software of the familiar time domain waveforms: exp(-x), sin x, and cos x, with the sampling interval $0 < h \leq 1$ (starting from origin), are listed in Table 6.1, Table 6.2, and Table 6.3, respectively.

These tables are obtained from the sample runs of the simulation programs and contain the following information:

. values of R and S,

. used time domain waveform,

. sampling interval value,

and

. the related FFT results.

Furthermore, the execution times of the simulation for various values of R and S are tabulated in Table 6.4. The left most column of Table 6.4 shows the tested values of S (2 to 4096) and the number of complex multiplications required to compute these S-point transforms. Each value of R (1 to 9) occupies a heading column that covers the following information:

(i) : number of complex multiplications in evaluating the Eq.(4.1),

Total : total number of complex multiplications,

N : total number of discrete complex data points, R x S,

Time : execution time.

R = 3, S = 4 ⟶ N = 12		
Waveform = exp(-x)		
Sampling interval = 0.1		
FFT Results		
Index	Real part	Imaginary part
0	7.34328318	0.00000000
1	0.60122949	1.25703716
2	0.41870633	0.59918678
3	0.38422716	0.34766316
4	0.37265784	0.20105702
5	0.36810848	0.09337187
6	0.36685848	0.00000000
7	0.36810845	-0.09337187
8	0.37265757	-0.20105749
9	0.38422716	-0.34766316
10	0.41870651	-0.59918672
11	0.60122973	-1.25703716

Table 6.1 12-point FFT of the function exp(-x) with h = 0.1.

R = 5, S = 8 ⟶ N = 40		
Waveform = sin x		
Sampling interval = 0.01		
FFT Results		
Index	Real part	Imaginary part
0	7.69912624	0.00000000
1	-0.22689843	-2.48410320
2	-0.20278139	-1.23060215
3	-0.19833173	-0.81139416
4	-0.19677617	-0.59941018
5	-0.19605695	-0.47014979
6	-0.19566670	-0.38218462
7	-0.19543199	-0.31776550
8	-0.19527966	-0.26801360
9	-0.19517718	-0.22798851
10	-0.19510402	-0.19471884
11	-0.19505043	-0.16630450
12	-0.19501071	-0.14146982
13	-0.19498037	-0.11932194
14	-0.19495772	-0.09921227
15	-0.19494024	-0.08065359
16	-0.19492714	-0.06326666
17	-0.19491792	-0.04674694
18	-0.19491161	-0.03083992
19	-0.19490775	-0.01532447
20	-0.19490623	0.00000000
21	-0.19490772	0.01532435
22	-0.19491138	0.03083962
23	-0.19491751	0.04674646
24	-0.19492806	0.06326592
25	-0.19494022	0.08065359
26	-0.19495773	0.09921230
27	-0.19498064	0.11932202
28	-0.19501078	0.14146984
29	-0.19505058	0.16630462
30	-0.19510399	0.19471884
31	-0.19517712	0.22798863
32	-0.19528036	0.26801334
33	-0.19543235	0.31776580
34	-0.19566695	0.38218459
35	-0.19605693	0.47014979
36	-0.19677649	0.59941012
37	-0.19833215	0.81139427
38	-0.20278190	1.23060215
39	-0.22689927	2.48410320

Table 6.2 40-point FFT of the function sin x with h = 0.01.

R = 7, S = 8 ⟶ N = 56		
Waveform = cos x		
Sampling interval = 0.001		
FFT Results		
Index	Real part	Imaginary part
00	55.97151947	0.00000000
01	-0.00366738	0.01395816
02	-0.00033231	0.00695640
03	0.00028503	0.00461329
04	0.00050139	0.00343409
05	0.00060120	0.00272039
06	0.00065541	0.00223988
07	0.00068834	0.00189255
08	0.00071108	0.00162822
09	0.00072393	0.00141813
10	0.00073442	0.00124735
11	0.00074197	0.00110454
12	0.00074747	0.00098264
13	0.00075243	0.00087696
14	0.00075722	0.00078392
15	0.00075855	0.00070077
16	0.00076745	0.00062698
17	0.00076325	0.00055606
18	0.00076416	0.00049251
19	0.00076561	0.00043322
20	0.00076652	0.00037748
21	0.00076792	0.00032471
22	0.00076783	0.00027436
23	0.00076857	0.00022566
24	0.00076747	0.00018635
25	0.00076917	0.00013328
26	0.00076972	0.00008829
27	0.00076905	0.00004429
28	0.00076866	0.00000000
29	0.00076916	-0.00004370
30	0.00076954	-0.00008827
31	0.00076926	-0.00013316
32	0.00076640	-0.00018993
33	0.00076846	-0.00022561
34	0.00076783	-0.00027436
35	0.00076792	-0.00032471
36	0.00076652	-0.00037771
37	0.00076565	-0.00043325
38	0.00076416	-0.00049251
39	0.00076317	-0.00055617
40	0.00074029	-0.00063264
41	0.00075867	-0.00070017
42	0.00075722	-0.00078392
43	0.00075231	-0.00087636
44	0.00074761	-0.00098270
45	0.00074188	-0.00110466
46	0.00073442	-0.00124735
47	0.00072349	-0.00141810
48	0.00072205	-0.00162512
49	0.00068834	-0.00189255
50	0.00065541	-0.00223988
51	0.00060108	-0.00272044
52	0.00050128	-0.00343409
53	0.00028512	-0.00461317
54	-0.00033225	-0.00695641
55	-0.00366750	-0.01395757

Table 6.3 56-point FFT of the function cos x with h = 0.001.

		R = 1			R = 3				R = 5				R = 7				R = 9			
S	(ii)	Total	N	Time	(i)	Total	N	Time	(i)	Total	N	Time	(i)	Total	N	Time	(i)	Total	N	Time
2	0	0	2	0	4	4	6	0	8	8	10	0	12	12	14	10	16	16	18	10
4	0	0	4	0	8	8	12	0	16	16	20	10	24	24	28	10	32	32	36	20
8	1	1	8	10	16	17	24	10	32	33	40	20	48	49	56	20	64	65	72	30
16	1	1	16	10	32	33	48	10	64	65	80	20	96	97	112	30	128	129	144	40
32	1	1	32	20	64	65	96	20	128	129	160	30	192	193	224	40	256	257	288	60
64	65	65	64	40	128	193	192	30	256	321	320	30	384	449	448	50	512	577	576	70
128	129	129	128	40	256	385	384	50	512	641	640	60	768	897	896	70	1024	1153	1152	80
256	257	257	256	70	512	769	768	90	1024	1281	1280	110	1536	1793	1792	150	2048	2305	2304	190
512	1025	1025	512	160	1024	2049	1536	180	2048	3073	2560	220	3072	4097	3584	310	4096	5121	4608	390
1024	2049	2049	1024	320	2048	4097	3072	390	4096	6145	5120	470	6144	8193	7168	670	8192	10241	9216	790
2048	4097	4097	2048	610	4096	8193	6144	810	8192	12289	10240	980	12288	16385	14336	1330	16384	20481	18432	1650
4096	12289	12289	4096	1320	8192	20481	12288	1740	16384	28673	20480	2080	24576	36385	28672	2770	32768	45057	36384	-

Table 6.4 Execution times for the VAX 11/780 simulation of Bergland's parallel FFT algorithm.

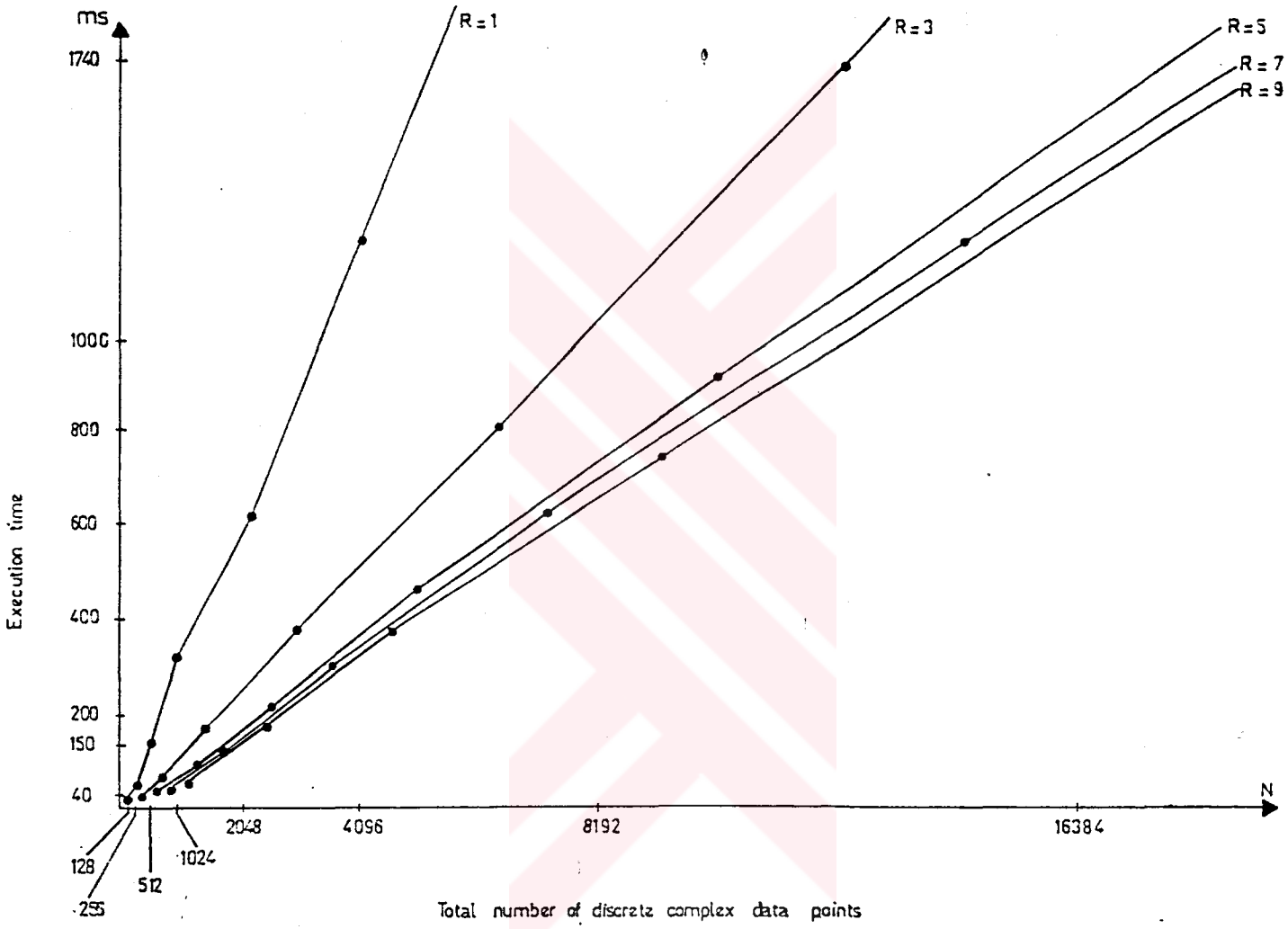


Figure 6.1 Execution times of VAX 11/780 simulation as a function of N.

The execution times are CPU times, which are measured with 10 milliseconds (0.010 second) resolution. The times measured on successive runs rarely differ by more than 10 to 20 milliseconds. Furthermore, the program has been run for N = 2,4,6,8,10, and 12, and yields a timing result of 0 or 10 milliseconds and for N = 36384 = R x S = 9 x 4096 exceeds the size of input array A (program CONTROL_ PROCESSOR in Appendix-A).

The Bergland's parallel FFT algorithm is the most efficient when R is considerably smaller than S, as mentioned in Chapter 4. It can be seen from Table 6.4 such that, given a number of identical processors that can each efficiently perform 1024 point transforms, these processors can then be interconnected to perform 3 x 1024 = 3072 point transform, 5 x 1024 = 5120 point transforms, and 7 x 1024 = 7168 point transforms, with the corresponding times 390, 470, and 670 milliseconds, respectively. Thus, while the number of discrete complex data points increases substantially, the execution time increases moderately.

However, as R increases, the N - S operations, that is, the number of complex multiplications in evaluating Eq.(4.1), take an increasing percentage of the execution time of the algorithm. The execution times for various values of R and S are plotted in Fig.6.1, as a function of N by using Table 6.4.

It can be seen from Fig.6.1 that for systems with more than five to seven processors the algorithm becomes time-inefficient.

6.2. DECIMATION-IN-TIME RADIX-2 FFT ALGORITHM

Some numerical results for the FFT simulation software of the time domain waveforms: exp(-x), sin x, and cos x, are given in Table 6.5, Table 6.6, and Table 6.7, respectively.

These tables are obtained from the sample runs of the simulation programs and contain the similar information as in the case of Bergland's algorithm.

The performance of this algorithm is examined both for VAXcluster simulation and a suitable shared bus computer, in the following three subsections.

6.2.1 Shared Bus Computer Implementation

In order to examine the performance of decimation-in-time radix-2 FFT algorithm on a P-processor shared bus computer, the following times (machine constants) are needed.

u : unit transfer time
(The time to transfer one datum of a butterfly from PE_i to PE_j.)

v : switching overhead
(The time needed by the control processor to establish a connection between PE_i and PE_j.)

P = 2, N = 8		
Waveform: exp(-x)		
Sampling interval = 0.01		
FFT Results		
Index	Real part	Imaginary part
0	7.72687149	0.00000000
1	0.03975432	0.09279081
2	0.03882639	0.03843970
3	0.03866703	0.01592252
4	0.03863382	-0.00000034
5	0.03866697	-0.01592268
6	0.03882606	-0.03844004
7	0.03975395	-0.09279098

Table 6.5 8-point FFT of the function exp(-x) with h = 0.01.

P = 4, N = 16		
Waveform: sin x		
Sampling interval = 0.1		
FFT Results		
Index	Real part	Imaginary part
0	9.78363132	0.00000000
1	-1.22209954	-2.68908453
2	-0.67823285	-1.22753012
3	-0.58368760	-0.75408661
4	-0.55141896	-0.50229645
5	-0.53707695	-0.33515799
6	-0.52996945	-0.20762622
7	-0.52655953	-0.09967268
8	-0.52553844	-0.00000045
9	-0.52655989	0.09967232
10	-0.52996963	0.20762593
11	-0.53707719	0.33515775
12	-0.55141950	0.50229597
13	-0.58368814	0.75408649
14	-0.67823374	1.22752988
15	-1.22210109	2.68908477

Table 6.6 16-point FFT of the function sin x with h = 0.1.

P = 8, N = 32		
Waveform: cos x		
Sampling interval = 0.001		
FFT Results		
Index	Real part	Imaginary part
0	31.99479485	0.00000000
1	-0.00057574	0.00259894
2	0.00004677	0.00128684
3	0.00016131	0.00084369
4	0.00020237	0.00061760
5	0.00022001	0.00047892
6	0.00023037	0.00038269
7	0.00023619	0.00031183
8	0.00024055	0.00025536
9	0.00024264	0.00021006
10	0.00024464	0.00017106
11	0.00024588	0.00013679
12	0.00024685	0.00010595
13	0.00024716	0.00007757
14	0.00024770	0.00005097
15	0.00024791	0.00002516
16	0.00024796	-0.00000140
17	0.00024789	-0.00002534
18	0.00024763	-0.00005132
19	0.00024710	-0.00007775
20	0.00024656	-0.00010665
21	0.00024579	-0.00013696
22	0.00024440	-0.00017141
23	0.00024250	-0.00021023
24	0.00023915	-0.00025676
25	0.00023598	-0.00031201
26	0.00022984	-0.00038304
27	0.00021969	-0.00047909
28	0.00020068	-0.00061830
29	0.00016074	-0.00084387
30	0.00004502	-0.00128718
31	-0.00057751	-0.00259912

Table 6.7 32-point FFT of the function cos x with h = 0.001.

b : butterfly computation time

(The time for calculating a single butterfly.)

The total execution time of the decimation-in-time radix-2 FFT algorithm consists of: a) total data transfer time, b) total switching overhead, and c) total butterfly computation time.

a) Total data transfer time:

As mentioned previously in Chapter 4., in FFT calculations no data transfer is necessary for stages $s \leqslant (m-k)$. For stages $s > (m-k)$, each PE_i keeps one datum out of a butterfly for computation in the next stage and transfers the other data to PE_j. With N/2P number of butterflies per processor, data are transferred sequentially over the bus for all $1 \leqslant i,j \leqslant P$. Each PE takes transfer time (N/2P)u per stage. For P PE's and k transfer stages, the total data transfer time is

$$k(N/2)u.$$

Note that, the time required to transfer the initial data from control processor to PE's plus the output data from PE's to control processor has been excluded from this total.

b) Total switching overhead:

The control processor takes time v to establish a connection between PE's i and j. After PE_i sends N/2P data items to PE_j, a reverse transfer occurs over the same connection. With P/2 such connections and k stages of transfers, the total switching overhead is

$$k(P/2)v.$$

c) Total butterfly computation time:

The butterfly computation time in a single PE is

$$t = m(N/2)b,$$

and in P PE's is

$$m(N/2P)b.$$

Hence, the total time is

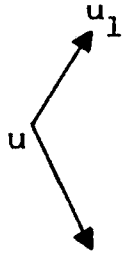
$$T = m(N/2P)b + k(N/2)u + k(P/2)v. \qquad (6.4)$$

and, the speed-up ratio is

$$S_p = t/T = P \,/\, (1 + P(k/m)(u/b) + (P^2/N)(k/m)(v/b)). \qquad (6.5)$$

6.2.2 VAXcluster Simulation

In this simulation, the machine constants u,v, and b, can be redefined as follows:



u_1 : The mailbox transfer time of an 8-byte complex number.

u_2 : The logical link transfer time of an 8-byte complex number.

v : The time needed to create a mailbox between two PROCESSING_ELEMENT's.

b : The charged CPU time for a butterfly computation.

In addition, the following two more times are needed to create the concurrent processes;

l : for a local subprocess,

r : for a remote process.

Hence, the total time for the simulated implementation is

$T = m(N/2P)b$	C	The butterfly computation time.
$+ (P/2)(1+r)$	S	The time needed to create local subprocesses and remote processes.
$+ (1+P/2)v$	S	The time needed to create mailboxes CP and MBX0# between CONTROL_PROCESSOR and local PROCESSING_ELEMENT's, and,
$+ (k-1)v$	C	MBX## between PROCESSING_ELEMENT's in the same node.
$+ (k-1)(N/P)u_1$	C	The data tranfer time using mailboxes, MBX##.
$+ (N/2)(u_1+u_2)$	S	The data transfer time using logical links and mailboxes, MBX0#.

(6.6)

where C and S show the operation type as Concurrent and Sequential, respectively.

Remarks:

- The second term, that is, the time needed to create local subprocesses and remote processes can be ignored by assuming that they are created previously (in the following, we ignore it).
- The time required to transfer the identification data of local PROCESSING_ELEMENTS's (using mailbox CP) and remote PROCESSING_ELEMENTS's (using logical links) has been worked out.

. As in the previous subsection, the transfer time of the initial and output data from/to CONTROL_PROCESSOR has been excluded from the total.

Therefore, the speed-up ratio is

$$P \,/\, (1 + (P/mNb)(2k+P)v + (1/mb)((2(k-1)+P)u_1 + Pu_2)). \tag{6.7}$$

Because of the absence of a definite klowledge of the machine constants u,v, and b, for the available VAXcluster system, several test programs have been written for an estimation. The following rough values have been obtained from these test programs

u_1 = 0.8 ms,

u

u_2 = 4 ms,

v = 10 ms,

b = 0.05 ms.

They have been calculated by taking a mean of the results obtained from the successive runs of the test programs. According to these values, the execution times of the VAXcluster simulation for various values of N and P are tabulated in Table 6.8 (all time values are in milliseconds).

Since both u_1 and u_2, and v are greater than b, the VAXcluster simulation of the algorithm is completely

N	P=1	P=2	P=4	P=8
4	0.2	29.7	-	-
8	0.6	39.5	61.1	-
16	1.6	59.2	82.4	112.4
32	4.0	98.8	125.2	155.2
64	9.6	178.4	211.2	241.2
128	22.4	338.4	384.0	414.0
256	51.2	660.0	731.2	761.2
512	115.2	1306.4	1428.8	1458.8
1024	256.0	2605.6	2830.4	2860.4
2048	563.2	5216.8	5646.4	5676.4
4096	1228.8	10464.8	11304.0	11334.0
8192	2662.4	21012.0	22670.4	22700.4
16384	5734.4	42208.8	45505.6	45535.6
32768	12288.0	84807.2	91380.8	91410.8

Table 6.8 Total execution times of decimation-in-time radix-2 FFT algorithm on VAXcluster.

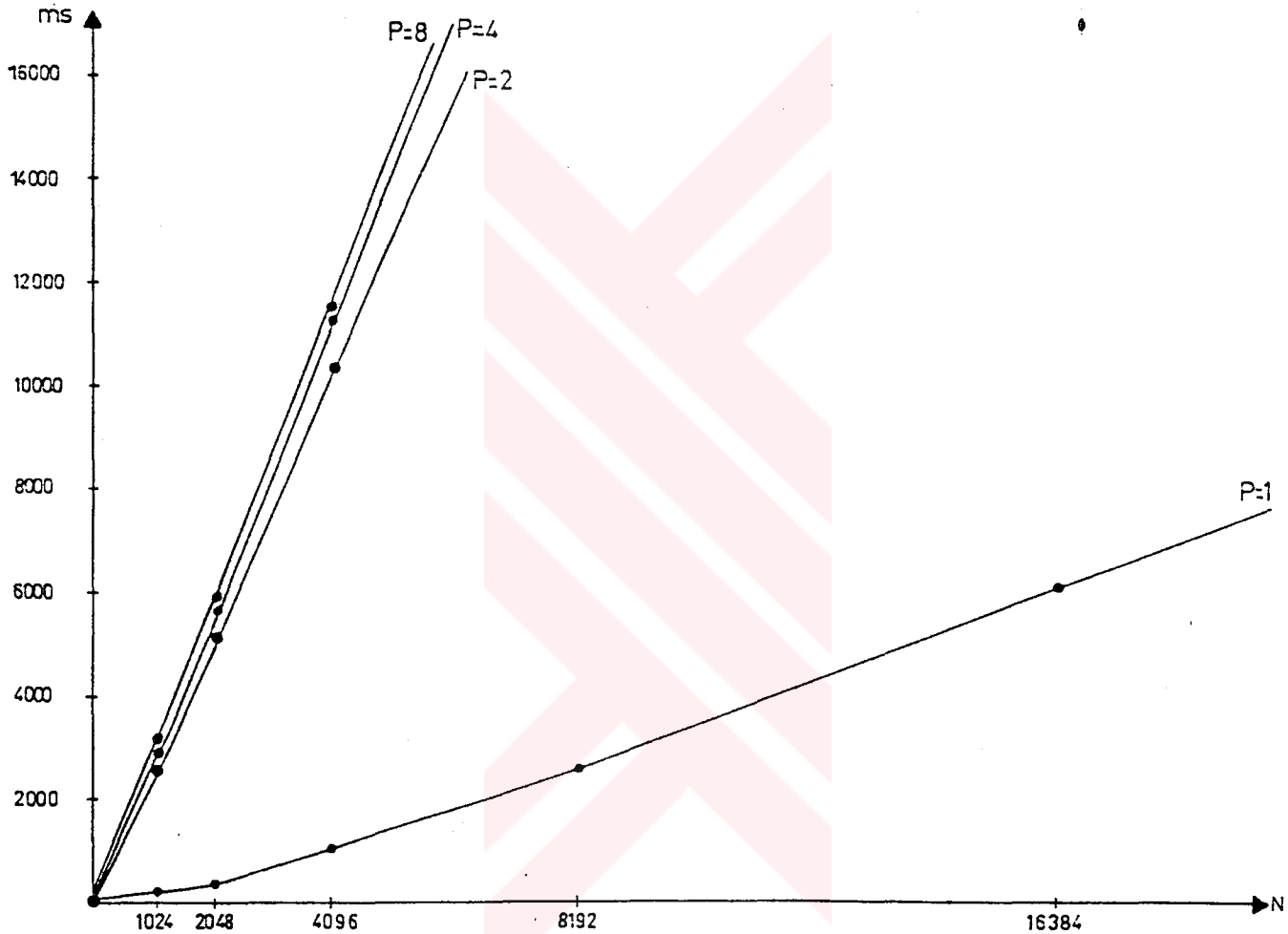


Figure 6.2 Total execution times of VAXcluster simulation for P = 2,4, and 8.

unsuitable. This result is depicted in Fig.6.2, by using the values in Table 6.8.

However, it is just a simulation and the result is dependent exactly on the VAXcluster architecture, and the available software tools.

6.2.3 General Results

In order to implement the algorithm efficiently, the first term of the Eq.(6.4),

$$m(N/2P)b,$$

must be the most dominant term on the result of the equation. To do this, the order of times u, v, and b, for the chosen machine must be as

$$v < u < b.$$

The desired results are obtained merely from this ideal ordering. For example, the results in Table 6.9 for a suitable shared bus computer that has the realistic values of b = 0.05, u = 0.001, and v = 0.0001 show that the speed-up ratio Eq.(6.5) of the algorithm closes to its perfect value, that is P. However, it is also seen from Table 6.9 that for larger values of P the speed-up ratio is smaller, although the time values u, v, and b, are in ideal order.

The results of the suitable shared bus computer are plotted in Fig.6.3, by using Table 6.9.

It is clear that the speed-up ratio, Eq.(6.5) depends heavily upon the factors u/b and v/b. As mentioned previously, these constants are machine dependent and vary from one configuration to another.

The speed-up Sp of the suitable shared bus computer are plotted in Fig.6.4, using the values in Table 6.9. As shown in this figure, the implementation of decimation-in-time radix-2 FFT algorithm on a shared bus computer closes to theoretical performance for $P \leqslant 16$, and gives much less speed-up for $P > 16$.

Finally, we can conclude that, when P and N are very large, the shared bus system is unsuitable because of the high congestion in the single bus.

N	P=1	P=2	Sp	P=4	Sp	P=8	Sp	P=16	Sp	P=32	Sp
4	0.2	0.1021	1.9598	-	-	-	-	-	-	-	-
8	0.6	0.3041	1.9730	0.1584	3.7879	-	-	-	-	-	-
16	1.6	0.8081	1.9800	0.4164	3.8425	0.2252	7.1048	-	-	-	-
32	4.0	2.0161	1.9840	1.0324	3.8745	0.5492	7.2833	0.3172	12.6103	-	-
64	9.6	4.8321	1.9867	2.4644	3.8955	1.2972	7.4006	0.7312	13.1291	0.4680	20.5128
128	22.4	11.2641	1.9886	5.7284	3.9103	2.9932	7.4836	1.6592	13.5005	1.0280	21.7899
256	51.2	25.7281	1.9900	13.0564	3.9214	6.7852	7.5458	3.7152	13.7812	2.2480	22.7758
512	115.2	57.8561	1.9911	29.3124	3.9301	15.1692	7.5943	8.2272	14.0023	4.8880	23.5679
1024	256.0	128.5121	1.9920	65.0244	3.9370	33.5372	7.6333	18.0512	14.1819	10.5680	24.2241
2048	563.2	282.6241	1.9928	142.8484	3.9426	73.4732	7.6654	39.2992	14.3311	22.7280	24.7800
4096	1228.8	616.4481	1.9934	311.2964	3.9474	159.7452	7.6922	84.9952	14.4573	48.6480	25.2590
8192	2662.4	1335.2961	1.9939	673.7924	3.9514	345.0892	7.7151	182.7872	14.5656	103.6880	25.6770
16384	5734.4	2875.3921	1.9943	1449.9844	3.9548	741.3772	7.7348	391.1712	14.6596	220.1680	26.0456
32768	12288.0	6160.3841	1.9947	3104.7684	3.9578	1585.1532	7.7519	833.5392	14.7420	465.9280	26.3732

Table 6.9 Total execution times of decimation-in-time radix-2 FFT algorithm on a suitable shared bus computer.

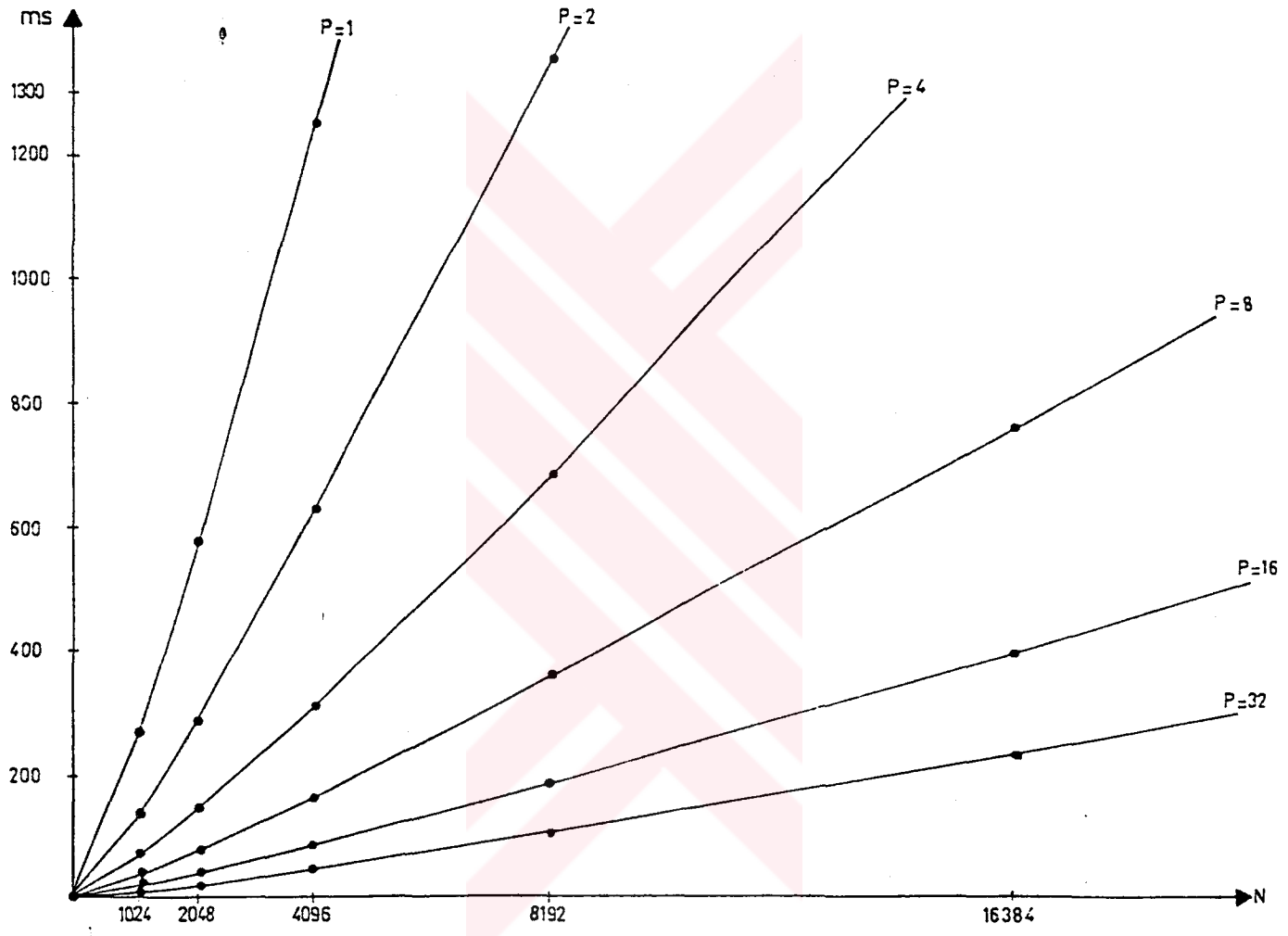


Figure 6.3 Total execution times of shared bus implementation for P = 2,4,8,16, and 32.

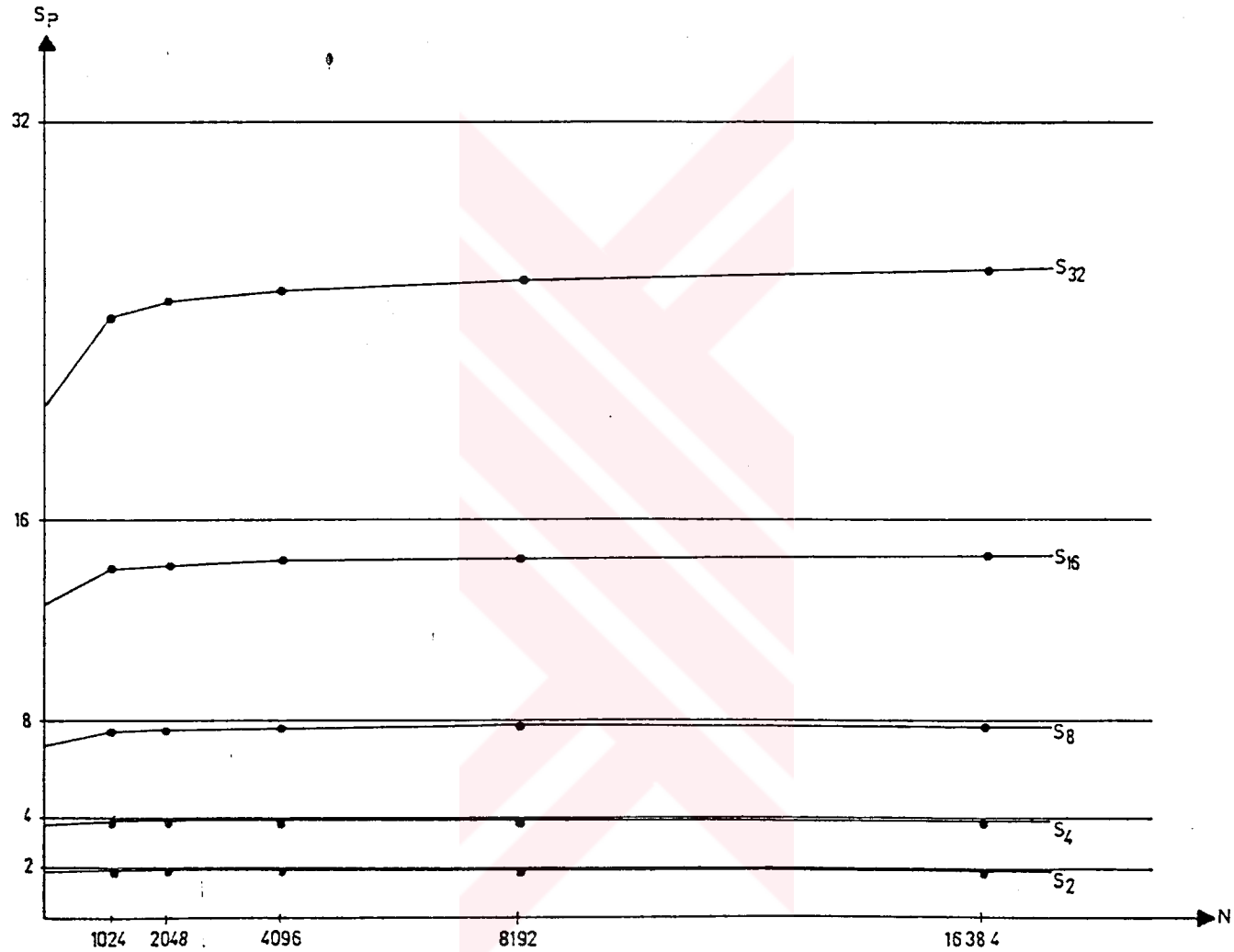


Figure 6.4 Speed-up of decimation-in-time radix-2 FFT algorithm on a suitable shared bus computer for P = 2,4,8,16, and 32.

CHAPTER 7.

SUMMARY AND CONCLUSIONS

In this study, the original implementations of two different FFT algorithms, namely, Bergland's parallel FFT algorithm on a PEPE system and decimation-in-time radix-2 FFT algorithm on a shared bus computer have been simulated with concurrent programming on an avaliable two-node VAXcluster system.

The simulation of Bergland's algorithm on a VAX 11/780 computer (one node of VAXcluster), gave good results that are very close to the results of its original implementation. An installed common block and the common event flags were used in this concurrent simulation both for communication and synchronization purposes between CONTROL_PROCESSOR and PROCESSING_ELEMENT's. The value of R can be 1,3,5,7, or 9, and the S, which is a power of two, is in the interval, $2 \leqslant S \leqslant 4096$. It was shown that the systems with more than five to seven processors the algorithm becomes time-inefficient.

The simulation of decimation-in-time radix-2 FFT algoritm on both nodes of VAXcluster gave bad results because of the large amount of time spent for data transfer between PROCESSING_ELEMENT's. Synchronous mailboxes and logical links were used for interprocess and intersystem communications in this concurrent simulation. The value of P can be 2,4, and 8, and the number of

discrete complex data points, N, is in the interval, $4 \leq N \leq 65536$. An approximate evaluation of speed-up has been done both for shared bus implementation and VAXcluster simulation of the algorithm. Finally, it was shown that the VAXcluster is completely unsuitable because of its architecture and shared bus implementation closes to theoretical performance for $P \leq 16$ and gives much less speed-up for $P > 16$.

As a result, there is an efficieny limit both for the number of processors and the number of data points in the parallel FFT implementations on multiprocessor systems, such as PEPE and shared bus computer.

It is obvious that these limits will be increased in the future architectures of multiprocessor systems. Although the computer; SIMD architecture with Multistage Interconnection Networks (MIN's) [8],[29],[33], and [39], is not yet commercially available, much research in the area indicates their potential advantage in various types of applications. The proposed systems like the Shuffle Exchange Network of Stone [39], and the Indirect Binary n-Cube Network of Pease [33], can be much suitable for future implementations of the FFT algorithm.

LIST OF REFERENCES

[1] Bergland, G.D., 1968. "A fast Fourier transform algorithm using base 8 iterations," Math. Comput., vol.22, pp.275-279.

[2] Bergland, G.D., 1969. "A radix-eight fast Fourier transform subroutine for real valued series," IEEE Trans. Audio Electroacoustics, vol.AU-17(2), pp.138-144.

[3] Bergland, G.D., 1969. "Fast Fourier transform hardware implementations - A survey," IEEE Trans. Audio Electroacoustics, vol.AU-17(2), pp.109-119.

[4] Bergland, G.D., 1969. "Fast Fourier transform hardware implementations - An overview," IEEE Trans. Audio Electroacoustics, vol.AU-17(2), pp.104-108.

[5] Bergland, G.D. and Wilson, D.E., 1969. "A fast Fourier transform algorithm for a global, highly parallel processor," IEEE Trans. Audio Electroacoustics, vol.AU-17(2), pp.125-127.

[6] Bergland, G.D., 1972. "A parallel implementation of the fast Fourier transform algorithm," IEEE Trans. Comput., vol.C-21, pp.366-370.

[7] Bhuyan, L.N. and Agrawal, D.P., 1983. "Performance analysis of FFT algorithms on multiprocessor systems," IEEE Trans. Soft. Eng., vol.SE-9, pp.512-519.

[8] Bhuyan, L.N. and Agrawal, D.P., 1983. "Design and performance of generalized interconnection networks," IEEE Trans. Comput., vol.C-32(12), pp.1081-1090.

[9] Bhuyan, L.N. and Agrawal, D.P., 1984. "Generalized hypercube and hyperbus structures for a computer network," IEEE Trans. Comput., vol.C-33(4), pp.323-333.

[10] Brenner, N.M., 1967. "Three FORTRAN programs that perform the Cooley-Tukey Fourier transform," M.I.T. Lincoln Lab., Tech. Note 1967-2.

[11] Brenner, N.M., 1969. "Fast Fourier transform of externally stored data," IEEE Trans. Audio Electroacoustics, vol.AU-17(2), pp.128-132.

[12] Chow, P., Vranesic, Z.G., and Yen, J.L., 1981. "A pipelined distributed arithmetic PFFT processor," IEEE Trans. Comput., vol.C-32(12), pp.1128-1136.

[13] Chu, S. and Burrus, C.S., 1982. "A prime factor FFT algorithm using disributed arithmetic," IEEE Trans. Acoust., Speech, Signal Processing, vol.ASSP-30(2), pp.217-226.

[14] Cooley, J.W. and Tukey, J.W., 1965. "An algorithm for the machine calculation of complex Fourier series," Math. of Comput., vol.19, pp.297-301.

[15] Cooley, J.W., Lewis, P.A.W., and Welch, P.D., 1967. "Historical notes on the fast Fourier transform," IEEE Trans. Audio Electroacoustics, vol.AU-15(2), pp.76-79.

[16] Corinthios, M.J., 1971. "A fast Fourier transform for high-speed signal processing," IEEE Trans. Comput., vol.C-20(8), pp.843-846.

[17] Corinthios, M.J., Smith, K.C., and Yen, J.L., 1975. "A parallel radix-4 fast Fourier transform computer," IEEE Trans. Comput., vol.C-24(1), pp.80-92.

[18] Despain, A.M., 1974. "Fourier transform computers using CORDIC iterations," IEEE Trans. Comput., vol.C-23(10), pp.993-1001.

[19] Fornberg, B., 1981. "A vector implementation of the fast Fourier transform algorithm," Math. Comput. vol.36(153), pp.189-191.

[20] Gentleman, W.M. and Sande, G., 1966. "Fast Fourier transforms - for fun and profit," AFIPS Proc. Fall Joint Computer Conf., vol.29, pp.563-578.

[21] Githen, J.A., 1970. "A fully parallel computer for radar data processing," Presented at the Nat. Aerosp. Electron. Conf.

[22] Gold, B., Lebow, I.L., McHugh, P.G., and Rader, C.M., 1971. "The FDP, a fast programmable signal processor," IEEE Trans. Comput., vol.C-28(2), pp.33-38.

[23] Gold, B. and Bially, T., 1973. "Parallelism in fast Fourier transform hardware," IEEE Trans. Audio Electroacoustics, vol.AU-21(1), pp.5-16.

[24] Good, I.J., 1960. "The interaction algorithm and practical Fourier analysis," J. Roy. Statist. Soc. Ser. B, vol.22, pp.372-375.

[25] Good, I.J., 1971. "The relationship between two fast Fourier transforms," IEEE Trans. Comput., vol.C-20, pp.310-317.

[26] Gottlieb, P. and Lorenzo, L.J., 1974. "Parallel data streams and serial arithmetic for fast Fourier transform processors," IEEE Trans. Acoust., Speech, Signal Processing, vol.ASSP-22(2), pp.111-117.

[27] Groginsky, H.L. and Works, G.A., 1970. "A pipeline fast Fourier transform," IEEE Trans. Comput., vol.C-19(11), pp.1015-1019.

[28] Korn, D.G. and Lambiotte, J.J.Jr., 1979. "Computing the fast Fourier transform on a vector computer," Math. Comput. vol.33(147), pp.977-992.

[29] Kruskal, C.P. and Snir, M., 1983. "The performance of multistage interconnection networks for multiprocessors," IEEE Trans. Comput., vol.C-32(12), pp.1091-1098.

[30] Mactaggart, I.R. and Jack, M.A., 1983. "Radix-2 FFT butterfly processor using distributed arithmetic," Elect. Lett., vol.19(2), pp.43-44.

[31] Mactaggart, I.R. and Jack, M.A., 1984. "A single chip radix-2 FFT butterfly architecture using parallel data distributed arithmetic," IEEE J. Solid State Circuits, vol.SC-19(3), pp.368-373.

[32] Pease, M.C., 1968. "An adaptation of the fast Fourier transform for parallel processing," vol.15(2), pp.252-264.

[33] Pease, M.C., 1977. "The indirect binary n-cube microprocessor array," IEEE Trans. Comput., vol.C-26, pp.458-473.

[34] Rader, C.M., 1968. "Discrete Fourier transforms when the number of data samples is prime," Proc. IEEE, vol.56, pp.1107-1108.

[35] Ramirez, R.W., 1985. The FFT Fundamentals and Concepts, Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

[36] Schendel, U., 1984. Introduction to Numerical Methods for Parallel Computers, Ellis Horward Limited, London.

[37] Singleton, R.C., 1967. "A method for computing the fast Fourier transform with auxiliary memory and limited high speed storage," IEEE Trans. Audio Electroacoustics, vol.AU-15(2), pp.91-98.

[38] Singleton, R.C., 1969. "An algorithm for computing the midex radix fast Fourier transform," IEEE Trans. Audio Electroacoustics, vol.AU-17(2), pp.93-103.

[39] Stone, H.S., 1971. "Parallel processing with perfect shuffle," IEEE Trans. Comput., vol.C-20, pp.153-161.

[40] Sudhakar, R., Agarwal, R.C., and Roy, S.C., 1981. "Fast computation of Fourier transform at arbitrary frequencies," IEEE Trans. Circuits Sys., vol.CAS-28(10), pp.972-980.

[41] Swarztrauber, P.N., 1986. "Symmetric FFTs," Math. Comput., vol.47(175), pp.323-346.

[42] Tuğtekin, Ş. and Kiper, A., "A concurrent simulation of the decimation-in-time- radix-2 FFT algorithm," Accepted for presentation at the AMSE International Conference on Modelling and Simulation, Istanbul, June 29 - July 1, 1988.

[43] Wesley, M.A., 1969. "Associative parallel processing for the fast Fourier transform," IEEE Trans. Audio Electroacoustics, vol.AU-17(2), pp.162-165.

[44] White, S.A., 1981. "A simple FFT butterfly arithmetic unit," IEEE Trans. Circuits Syst., vol.CAS-28(4), pp.352-355.

[45] Winograd, S., 1978. "On computing the discrete Fourier transform," Math. Comput., vol.32(141), pp.175-199.

[46] Programming in VAX FORTRAN, 1984, VAX/VMS Version 4.0.

[47] VAX FORTRAN User's Guide, 1984, VAX/VMS Version 4.0.

[48] Guide to Programming on VAX/VMS (FORTRAN Edition), 1984, VAX/VMS Version 4.0.

[49] Introduction to VAX/VMS System Routines, 1984, VAX/VMS Version 4.0.

[50] VAX/VMS System Services Reference Manual, 1985, VAX/VMS Version 4.2.

[51] VAX/VMS Run-Time Library Routines Reference Manual, 1985, VAX/VMS Version 4.2.

[52] Guide to Networking on VAX/VMS, 1985, VAX/VMS Version 4.2.

[53] VAXcluster Systems Handbook, 1986.

APPENDIX

APPENDIX-A

PROGRAM LISTINGS

This appendix contains the source and complementary codes of the simulations. Both the file and program names are given separately in the following.

The simulation software for Bergland's parallel FFT algorithm consists of the following files that contain the specified program or the complementary code.

File name	Program name
CP.FOR	CONTROL_PROCESSOR
GET.FOR	GET_POINTS
PUT.FOR	PUT_RESULTS
PE.FOR	PROCESSING_ELEMENT
FFT4_2.FOR	FFT4_2

and, the complementary codes

IN_PLACE.FOR
IN_PLACE.OPT

The simulation software for decimation-in-time radix-2 FFT algorithm consists of the following files that contain the specified program or the complementary code.

File name	Program name
CP.FOR	CONTROL_PROCESSOR
GET.FOR	GET_POINTS
SCRAMBLE.FOR	BIT_REVERSED
PUT.FOR	PUT_RESULTS
PE.FOR	PROCESSING_ELEMENT

and, the complementary code

PE.COM

A.1. PROGRAMS FOR THE SIMULATION OF BERGLAND'S PARALLEL FFT ALGORITHM

```
C+
C FILE: FFT$BERGLAND:CP.FOR
C
C AUTHOR: Sadan Tugtekin,    CREATION DATE: November, 1987
C-
        PROGRAM   CONTROL_PROCESSOR
        !
        !Declare variables
        !
        COMPLEX*8 A(0:28671)            !Complex discrete data points
        INTEGER*4 FLAG(0:31),           !Flags of CLUSTER_2
        2         F/63/,                !Index variable for FLAG
        2         PE,                   !Processing element number
        2         R,                    !Odd factor of N
        2         RPLUS1,               !R + 1
        2         RMINUS1,              !R - 1
        2         S,                    !Other factor of N, a power of 2
        2         N,                    !Number of complex data points
        2         STATUS,               !Conditions returned
        2         MASK,                 !To run processes concurrently
        2         POWER,                !S = 2**POWER
        2         I                     !Incremental variable for loops
        CHARACTER OUT*7,                !Output file name for subprocess
        2         CPE                   !Character data type of PE
        !
        !Declare external system services and
        !run time library routines
        !
        INTEGER*4 SYS$ASCEFC,           !Ass. Comm. Event Flag Cluster
        2         SYS$SETEF,            !Set Event Flag
        2         SYS$CLREF,            !Clear Event Flag
        2         SYS$WAITFR,           !Wait for Single Event Flag
        2         LIB$SPAWN,            !Spawn Subprocess
        2         LIB$SIGNAL,           !Signal Exception Condition
        2         LIB$INIT_TIMER,       !Initialize Times and Counts
        2         LIB$SHOW_TIMER        !Show Accumulated Times and Counts
        !
        !Declare installed common memory area
        !
        COMMON    /IN_PLACE/    A,R,S,
```

```
     2          /PE_NUM/       PE,
     2          /SYNCHRONIZE/ FLAG
     !
     !Input R and S
     !
     TYPE 1100
1100 FORMAT(X'Enter R(=1,3,5,7,9) and S(<=4096): ',$)
     ACCEPT *, R,S
     !
     !Check validity of R and S
     !
     POWER = INT(ALOG(FLOAT(S)) / ALOG(2.0))
     IF (R .LT. 1 .OR. R .GT. 9) THEN
       STOP 'R must be 1,3,5,7 or 9'
     ELSE IF (R .EQ. 2 .OR. R .EQ. 4 .OR.
     2         R .EQ. 6 .OR. R .EQ. 8) THEN
       STOP 'R must be odd'
     ELSE IF (S .LT. 2 .OR. S .GT. 4096) THEN
       STOP 'S must be in  2 <= S <= 4096'
     ELSE IF (2**POWER .NE. S) THEN
       STOP 'S must be a power of 2'
     ELSE IF (R .EQ. 9 .AND. S .EQ. 4096) THEN
       STOP 'Common block exceeds the global page limit'
     END IF
     !
     !Initialize times and counts
     !
     CALL LIB$INIT_TIMER
     !
     !Calculate initial values
     !
     N = R * S
     RPLUS1 = R + 1
     RMINUS1 = R - 1
     !
     !Input FLAG by flags of CLUSTER_2
     !
     DO I=0,31
       F = F + 1
       FLAG(I) = F
     END DO
     F = 0
     !
     !Input A with complex discrete data points
     !
     CALL GET_POINTS
     !
```

```
!Associate common event flag CLUSTER_2
!
STATUS = SYS$ASCEFC(%VAL(FLAG(F)),'CLUSTER_2',,)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
!
!Set the lowest bit of the longword MASK (pass as a
!parameter to LIB$SPAWN routine) to run spawned
!subprocesses and CP, concurrently.
!
MASK = IBSET(MASK,0)
!
!Create and run PEs
!
DO PE=0,RMINUS1
  !
  !Make up an output file name for the subprocess
  !to be spawned
  !
  WRITE(CPE,'(I1)') PE
  OUT = 'PE'//CPE//'.OUT'
  !
  !Spawn the subprocess which will be run concurrently
  !with this CP and all other subprocesses
  !
  STATUS = LIB$SPAWN('RUN PE',,OUT,MASK)
  IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
  !
  !Wait, spawned subprocess must get its PE_number
  !from installed common memory area
  !
  STATUS = SYS$WAITFR(%VAL(FLAG(F)))
  IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
  !
  !Clear event flag for the next PE
  !
  STATUS = SYS$CLREF(%VAL(FLAG(F)))
  IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
END DO
!
!Wait, now PEs are working to finish their jobs
!
DO I=0,RMINUS1
  F = I + RPLUS1
  STATUS = SYS$WAITFR(%VAL(FLAG(F)))
  IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
END DO
!
```

```
!Show accumulated times and counts
!
CALL LIB$SHOW_TIMER
!
!Output transformation results
!
CALL PUT_RESULTS
END                         !Of control_processor
```

```
C+
C FILE: FFT$BERGLAND:GET.FOR
C
C AUTHOR: Sadan Tugtekin,    CREATION DATE: November, 1987
C-
        SUBROUTINE GET_POINTS
        !
        !Declare variables
        !
        COMPLEX*8 A(0:28671)            !Complex discrete data points
        REAL*4    X/0.0/,               !Value of x
        2         H                     !Step size of equally spaced data
        CHARACTER ANSWER                !Answer to exp(-x), sin x, cos x ?
        INTEGER*4 R,                    !Odd factor of N
        2         S,                    !Other factor of N, a power of 2
        2         N,                    !Number of complex data points
        2         I                     !Incremental variable for loops
        !
        !Declare run time library routine
        !
        INTEGER*4 STR$UPCASE
        !
        !Declare only /IN_PLACE/ part of the
        !installed common memory area
        !
        COMMON    /IN_PLACE/ A,R,S
        !
        !Find N
        !
        N = R * S
        !
        !N complex discrete data points can be obtained
        !from the familiar time domain waveforms
        !
        !        exp(-x),
        !        sin x,
        !   and
        !        cos x,
        !
        !with step size  0 < h <= 1.
        !
        !
        !Learn the user request
        !
10      TYPE 110
110     FORMAT(X,'E:exp(-x), S:sin x, C:cos x ? ',$)
        ACCEPT 120, ANSWER
```

```
120     FORMAT(A)
        !
        !Convert ANSWER to uppercase character
        !
        CALL STR$UPCASE(ANSWER,ANSWER)
        !
        !Check the validity of ANSWER
        !
        IF (ANSWER .NE. 'E' .AND.
       2   ANSWER .NE. 'S' .AND.
       2   ANSWER .NE .'C') GOTO 10
        !
20      TYPE 210
210     FORMAT(X,'Enter step size (0 < h <= 1): ',$)
        ACCEPT 220, H
220     FORMAT(F10.6)
        IF (H .LE. 0 .OR. H .GT. 1) GOTO 20
        IF (ANSWER .EQ. 'E') THEN
          !
          !exp(-x)
          !
          DO I=0,N-1
            A(I) = EXP(-X)
            X = X + H
          END DO
        ELSE IF (ANSWER .EQ. 'S') THEN
          !
          !sin x
          !
          DO I=0,N-1
            A(I) = SIN(X)
            X = X + H
          END DO
        ELSE
          !
          !cos x
          !
          DO I=0,N-1
            A(I) = COS(X)
            X = X + H
          END DO
        END IF
        !
        RETURN                  !To control_processor
        END                     !Of get_points
```

```
C+
C FILE: FFT$BERGLAND:PUT.FOR
C
C AUTHOR: Sadan Tugtekin,    CREATION DATE: November, 1987
C-
        SUBROUTINE PUT_RESULTS
        !
        !Declare variables
        !
        COMPLEX*8 A(0:28671)            !Complex discrete data points
        CHARACTER ANSWER                !Answer to Screen, Disk, End?
        INTEGER*4 R,                    !Odd factor of N
        2         S,                    !Other factor of N, a power of 2
        2         N,                    !Number of complex data points
        2         I                     !Incremental variable for loops
        !
        !Declare run time library routine
        !
        INTEGER*4 STR$UPCASE
        !
        !Declare only /IN_PLACE/ part of the
        !installed common memory area
        !
        COMMON    /IN_PLACE/ A,R,S
        !
        !Find N
        !
        N = R * S
        !
        !Learn the user request to where the results
        !will be output
        !
10      TYPE 100
100     FORMAT(X,'S:Screen, D:Disk, E:End? ',$)
        ACCEPT 120, ANSWER
120     FORMAT(A)
        !
        !Convert ANSWER to uppercase character
        !
        CALL STR$UPCASE(ANSWER,ANSWER)
        !
        !Check validity of ANSWER
        !
        IF (ANSWER .NE. 'S' .AND.
        2  ANSWER .NE. 'D' .AND.
        2   ANSWER .NE .'E') GOTO 10
        !
```

```
      !Output the results to Screen or Disk, or End
      !the program
      !
      IF (ANSWER .EQ. 'S') THEN
        !
        !Output the results to Screen
        !
        TYPE *, ' *** FFT RESULTS ***'
        DO I=0,N-1
          TYPE 110, I,REAL(A(I)),AIMAG(A(I))
110       FORMAT(X,'A(',I4.4,') = (',F14.8,',',F14.8,')')
        END DO
      ELSE IF (ANSWER .EQ. 'D') THEN
        !
        !Output the results to Disk
        !
        TYPE *, 'Wait, I''m writing to FFT.DAT'
        OPEN (UNIT=1,
     2       FILE='FFT.DAT',
     2       STATUS='NEW',
     2       RECL=80)
        WRITE(1,*) ' *** FFT RESULTS ***'
        DO I=0,N-1
          WRITE(1,110) I,REAL(A(I)),AIMAG(A(I))
        END DO
      END IF
      !
      !If ANSWER is 'E' then End the program
      !
      RETURN                    !To control_processor
      END                       !Of put_results
```

```
C+
C FILE: FFT$BERGLAND:PE.FOR
C
C AUTHOR: Sadan Tugtekin,    CREATION DATE: November, 1987
C-
      PROGRAM   PROCESSING_ELEMENT
      !
      !Declare variables
      !
      COMPLEX*8 A(0:28671),          !Complex discrete data points
      2         B(0:4095),           !Data array for standard FFT
      2         W(1:8)               !CEXP() values used to input B
      REAL*4    TEMP,                !Temporary variable
      2         TPIE                 !Two pie
      INTEGER*4 FLAG(0:31),          !Flags of CLUSTER_2
      2         F/0/,                !Index variable for FLAG
      2         PE,                  !Processing element number
      2         R,                   !Odd factor of N
      2         RPLUS1,              !R + 1
      2         RMINUS1,             !R - 1
      2         S,                   !Other factor of N, a power of 2
      2         SMINUS1,             !S - 1
      2         N,                   !Number of complex data points
      2         Q,                   !Used for
      2         Q1,                  !generating
      2         Q2,                  !Q values
      2         P,                   !Used
      2         P1,                  !for
      2         P2,                  !generating
      2         SR/1/,               !P
      2         RS/1/,               !values
      2         S_SR,                !S * SR
      2         R_RS,                !R * RS
      2         PRM_R,               !To pass R and S
      2         PRM_S,               !as parameters
      2         STATUS,              !Conditions returned
      2         I                    !Incremental variable for loops
      LOGICAL*1 EXIT                 !To exit from DO WHILE()s
      !
      !Declare external system services and
      !run time library routines
      !
      INTEGER*4 SYS$ASCEFC,          !Ass. Comm. Event Flag Cluster
      2         SYS$SETEF,           !Set Event Flag
      2         SYS$WAITFR,          !Wait for Single Event Flag
      2         LIB$SIGNAL,          !Signal Exception Condition
      2         LIB$INIT_TIMER,      !Initialize Times and Counts
```

```
2         LIB$SHOW_TIMER        !Show Accumulated Times and Counts
!
!Declare installed common memory area
!
COMMON    /IN_PLACE/    A,R,S,
2         /PE_NUM/      PE,
2         /SYNCHRONIZE/ FLAG
!
!Initialize times and counts
!
CALL LIB$INIT_TIMER
!
!Associate common event flag CLUSTER_2
!
STATUS = SYS$ASCEFC(%VAL(FLAG(F)),'CLUSTER_2',,)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
!
!Get PE_number from installed common
!memory area and signal CP
!
P1 = PE
STATUS = SYS$SETEF(%VAL(FLAG(F)))
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
!
!Calculate initial values
!
N = R * S
PRM_R = R
PRM_S = S
RPLUS1 = R + 1
RMINUS1 = R - 1
SMINUS1 = S - 1
TPIE = 8.0 * ATAN(1.0)
!
!Determine elements of W
!(Since the zeroeth element of W is 1 for each PE
! and all the elements of W is 1 for PE0, they are
! not considered.)
!
IF (P1 .NE. 0) THEN
  TEMP = TPIE * P1 / R
  DO Q1=1,RMINUS1
    W(Q1) = CEXP(CMPLX(0.0,TEMP * Q1))
  END DO
END IF
!
!Generate input values of standard FFT algorithm
```

```
!(First equation of Bergland's Algorithm.
! It represents a set of R point transforms.)
!
IF (P1 .NE. 0) THEN
  DO Q2=0,SMINUS1
    B(Q2) = B(Q2) + A(MOD(PRM_R * Q2,N))
    DO Q1=1,RMINUS1
      Q = R * Q2 + S * Q1
      Q = MOD(Q,N)
      B(Q2) = B(Q2) + W(Q1) * A(Q)
    END DO
  END DO
ELSE
  DO Q2=0,SMINUS1
    DO Q1=0,RMINUS1
      Q = R * Q2 + S * Q1
      Q = MOD(Q,N)
      B(Q2) = B(Q2) + A(Q)
    END DO
  END DO
END IF
F = P1 + 1
!
!Invoke Radix 4_2 FFT subroutine
!(Second equation of Bergland's Algorithm.
! It represents a set of S point transforms.)
!
CALL FFT4_2(B,PRM_S)
!
!Signal other PEs which are
!waiting for this PE
!
IF (R .NE. 1) THEN
  STATUS = SYS$SETEF(%VAL(FLAG(F)))
  IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
END IF
!
!Wait for other PEs to put transformation
!results to A
!
IF (R .NE. 1) THEN
  DO I=0,RMINUS1
    F = I + 1
    IF (I .NE. P1) THEN
      STATUS = SYS$WAITFR(%VAL(FLAG(F)))
      IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
    END IF
```

```
  END DO
END IF
!
!Calculate SR
!
IF (R .NE. 1) THEN
  EXIT = .FALSE.
  DO WHILE (.NOT. EXIT)
    S_SR = S * SR
    IF (MOD(S_SR,PRM_R) .EQ. 1) THEN
      EXIT = .TRUE.
    ELSE
      SR = SR + 1
    END IF
  END DO
ELSE
  S_SR = 1
END IF
!
!Calculate RS
!
EXIT = .FALSE.
DO WHILE (.NOT. EXIT)
  R_RS = R * RS
  IF (MOD(R_RS,PRM_S) .EQ. 1) THEN
    EXIT = .TRUE.
  ELSE
    RS = RS + 1
  END IF
END DO
!
!Put the transformed values to A
!
Q = 0
TEMP = S_SR * P1
DO P2=0,SMINUS1
  P = TEMP + R_RS * P2
  P = MOD(P,N)
  A(P) = B(Q)
  Q = Q + 1
END DO
!
!Show accumulated times and counts
!
CALL LIB$SHOW_TIMER
!
!Signal CP that is waiting for ending
```

```
!
F = P1 + RPLUS1
STATUS = SYS$SETEF(%VAL(FLAG(F)))
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
END                          !Of processing_element
```

```
C+
C FILE: FFT$BERGLAND:FFT4_2.FOR
C
C AUTHOR: Sadan Tugtekin,    CREATION DATE: November, 1987
C-
      SUBROUTINE FFT4_2(A,N)
      COMPLEX*8 A(0:*),              !Complex discrete data points
     2          A1,
     2          A2,                  !Temporary variables
     2          A3
      REAL*4    TEMP,                !Variables
     2          PIE,
     2          COSI,                !to
     2          SINE,
     2          COS1,                !calculate
     2          COS2,
     2          COS3,                !twiddle
     2          SIN1,
     2          SIN2,                !factors
     2          SIN3,
     2          REL,                 !Arguments
     2          IMG                  !for CMPLX()
      INTEGER*4 LENGTH,
     2          SKIP,
     2          START,               !Index
     2          P0,
     2          P1,                  !variables
     2          P2,
     2          P3,
     2          PERMUTE(0:20),       !To permute results
     2          PMT/0/,              !Index variable for PERMUTE
     2          N,                   !Number of complex data points
     2          NMINUS1,             !N - 1
     2          N4OR2,               !N / 4  or  N / 2
     2          LMINUS1              !LENGTH - 1
      INTEGER*2 FACTOR(11),          !Factors of N
     2          M/0/,                !Number of factors
     2          SYMMETRIC,           !Number of symmetric factors
     2          J                    !Index variable
      LOGICAL*1 EXIT                 !To exit from DO WHILE()s
      !
      !Calculate initial values
      !
      IF (N-2 .LT. 0) RETURN
      PIE = 4.0 * ATAN(1.0)
      NMINUS1 = N - 1
      LENGTH = N
```

```
PERMUTE(PMT) = N
! -
!Determine factors of N symmetrically
!
PO = N
DO WHILE (PO-(PO/16)*16 .EQ. 0)
  M = M + 1
  FACTOR(M) = 4
  PO = PO / 16
END DO
IF (PO .LE. 4) THEN
  SYMMETRIC = M
  FACTOR(M+1) = PO
  IF (PO .NE. 1) M = M + 1
ELSE
  M = M + 2
  SYMMETRIC = M - 1
  FACTOR(SYMMETRIC) = 2
  FACTOR(M) = 2
END IF
IF (SYMMETRIC. NE. 0) THEN
  DO J = SYMMETRIC,1,-1
    M = M + 1
    FACTOR(M) = FACTOR(J)
  END DO
  J = 0
END IF
DO WHILE (LENGTH .GT. 1)
  !
  !Set up required values of transform split
  !
  TEMP = PIE / FLOAT(LENGTH)
  COSI = 2.0 * SIN(TEMP)**2
  SINE = SIN(TEMP+TEMP)
  J = J + 1
  IF (FACTOR(J) .EQ. 4) THEN
    !
    !Calculate Radix_4 transform
    !
    SKIP = LENGTH
    LENGTH = LENGTH / 4
    LMINUS1 = LENGTH - 1
    PMT = PMT + 1
    PERMUTE(PMT) = LENGTH
    COS1 = 1.0
    SIN1 = 0.0
    DO START=0,LMINUS1
```

```
    DO P0=START,NMINUS1,SKIP
      P1 = P0 + LENGTH
      P2 = P1 + LENGTH
      P3 = P2 + LENGTH
      REL = REAL(A(P1)) - REAL(A(P3))
      IMG = AIMAG(A(P1)) - AIMAG(A(P3))
      A1 = A(P0) - A(P2) + CMPLX(-IMG,REL)
      A3 = A(P0) - A(P2) + CMPLX(IMG,-REL)
      A2 = A(P0) - A(P1) + A(P2) - A(P3)
      A(P0) = A(P0) + A(P1) + A(P2) + A(P3)
      IF (START .GT. 0) THEN
        !
        !Multiply by twiddle factors
        !
        A(P1) = A1 * CMPLX(COS1,SIN1)
        A(P2) = A2 * CMPLX(COS2,SIN2)
        A(P3) = A3 * CMPLX(COS3,SIN3)
      ELSE
        !
        !Skip these multiplications for points
        !which have no twiddle factor or
        !twiddle factor of value 1
        !
        A(P1) = A1
        A(P2) = A2
        A(P3) = A3
      END IF
    END DO
    IF (START-LMINUS1 .LT. 0) THEN
      !
      !Calculate a new set of twiddle factors
      !
      TEMP = SIN1 + (SINE*COS1 - COSI*SIN1)
      COS1 = COS1 - (COSI*COS1 + SINE*SIN1)
      SIN1 = TEMP
      COS2 = COS1 * COS1 - SIN1 * SIN1
      SIN2 = 2.0  * COS1 * SIN1
      COS3 = COS1 * COS2 - SIN1 * SIN2
      SIN3 = COS1 * SIN2 + COS2 * SIN1
    END IF
  END DO
ELSE
  !
  !Calculate Radix_2 transform
  !
  SKIP = LENGTH
  LENGTH = LENGTH / 2
```

```
PMT = PMT + 1
PERMUTE(PMT) = LENGTH
P2 = LENGTH + 1
!
!Calculate transform values of points
!which have no twiddle factor or
!twiddle factor of value 1
!
DO P0=0,NMINUS1,SKIP
  P1 = P0 + LENGTH
  A1 = A(P0) - A(P1)
  A(P0) = A(P0) + A(P1)
  A(P1) = A1
END DO
IF (LENGTH .GT. 1) THEN
  !
  !Calculate twiddle factors
  !
  COS1 = 1.0 - COSI
  SIN1 = SINE
  P0 = P0 - NMINUS1
  EXIT = .FALSE.
  DO WHILE (.NOT. EXIT)
    !
    !Multiply by twiddle factors
    !
    DO P0=P0,NMINUS1,SKIP
      P1 = P0 + LENGTH
      A1 = A(P0) - A(P1)
      A(P0) = A(P0) + A(P1)
      A(P1) = A1 * CMPLX(COS1,SIN1)
    END DO
    P1 = P0 - NMINUS1
    COS1 = -COS1
    P0 = P2 - P1
    IF (P0 .LE. P1) THEN
      P0 = P0 + 1
      IF (P0 .LT. P1) THEN
        !
        !Calculate a new set of twiddle factors
        !
        TEMP = SIN1 + (SINE*COS1 - COSI*SIN1)
        COS1 = COS1 - (COSI*COS1 + SINE*SIN1)
        SIN1 = TEMP
      ELSE
        EXIT = .TRUE.
      END IF
```

```
          END IF
        END DO
      END IF
    END IF
  END DO
  !
  !Unscramble the results
  !
  START = 1
  N4OR2 = PERMUTE(1)
  SKIP = N4OR2
  PMT = 0
  DO WHILE (START .LT. NMINUS1)
    A1 = A(START)
    A(START) = A(SKIP)
    A(SKIP) = A1
    START = START + 1
    SKIP = SKIP + N4OR2
    DO WHILE (SKIP .GE. PERMUTE(PMT)-1 .AND. START .LT. NMINUS1)
      SKIP = SKIP - PERMUTE(PMT)
      PMT = PMT + 1
      SKIP = SKIP + PERMUTE(PMT+1)
      DO WHILE (SKIP .LE. PERMUTE(PMT)-1)
        PMT = 0
        DO WHILE (SKIP .LE. NMINUS1)
          IF (START .LT. SKIP) THEN
            GOTO 999
          ELSE
            START = START + 1
            SKIP = SKIP + N4OR2
          END IF
        END DO
      END DO
    END DO
999 END DO
  RETURN                    !To processing_element
  END                       !Of fft4_2
```

```
C+
C FILE: FFT$BERGLAND:IN_PLACE.FOR
C
C ABSTRACT:
C       To communicate between processes using a common block,
C       you must install the common block as a shared shareable image
C       and link each program that references the common block against
C       that shareable image.
C
C       IN_PLACE.FOR is a FORTRAN program containing non_executable
C       code for the common block to be installed.
C
C       To compile and link the program, type the following commands:
C
C       $ FORTRAN IN_PLACE
C       $ LINK/SHAREABLE IN_PLACE
C
C       Give yourself CMKRNL privilege (required for use of Install
C       Utility) and install the shareable image by following commands:
C
C       $ INSTALL
C       INSTALL> CREATE FFT$BERGLAND:IN_PLACE/WRITEABLE/SHARED
C       INSTALL> EXIT
C
C
C AUTHOR: Sadan Tugtekin,    CREATION DATE: November, 1987
C-
        !
        !Declare variables used in common block
        !
        COMPLEX*8 A(0:28671)            !Complex discrete data points
        INTEGER*4 FLAG(0:31),           !Flags of CLUSTER_2
        2         PE,                   !Processing element number
        3         R,                    !Odd factor of N
        4         S                     !Other factor of N, a power of 2
        !
        !Declare common block
        !
        COMMON    /IN_PLACE/    A,R,S,
        2         /PE_NUM/      PE,
        3         /SYNCHRONIZE/ FLAG
        END
```

```
!+
! FILE: FET$BERGLAND:IN_PLACE.OPT
!
! ABSTRACT:
!       Linking against a shareable image requires
!       an options file.
!
!       IN_PLACE.OPT is that options file.
!
! AUTHOR: Sadan Tugtekin,    CREATION DATE: November, 1987
!-
       IN_PLACE / SHAREABLE
!
! End of options file IN_PLACE.OPT
```

A.2. PROGRAMS FOR THE SIMULATION OF DECIMATION-IN-TIME RADIX-2 FFT ALGORITHM

```
C+
C FILE: FFT$BHUYAN:CP.FOR
C
C AUTHOR: Sadan Tugtekin,    CREATION DATE: November, 1987
C-
        PROGRAM   CONTROL_PROCESSOR
        !
        !Declare variables
        !
        COMPLEX*8 A(0:65535),           !Complex discrete data points
        2         CTEMP                 !Complex temporary variable
        INTEGER*4 STATUS,               !Conditions returned
        2         MASK,                 !To run processes concurrently
        2         CHANNEL,              !I/O channel for a mailbox
        2         LUNA(0:8),            !Array for logical unit numbers
        2         LP/0/,                !Index variable for LUNA
        2         N,                    !Number of complex data points
        2         NOVER2,               !N / 2
        2         NOVERP,               !N / P
        2         NOVER2P,              !N / (2 * P)
        2         P,                    !Total number of PE
        2         POVER2,               !P / 2
        2         PE,                   !Processing element number
        2         REMOTE_PE,            !PE_number on remote node
        2         M,                    !N = 2**M
        2         K,                    !P = 2**K
        2         TBOI,                 !Theoretical beginning of input
        2         TEOI,                 !Theoretical end of input
        2         TBOO1,                !Theoretical beginning of output one
        2         TEOO1,                !Theoretical end of output one
        2         TEOO2,                !Theoretical end of output two
        2         FINISH,               !Index
        2         TRANSFER,             !variables
        2         OUTPUT                !for data transfer
        CHARACTER OUT*7,                !Output file name for subprocess
        2         CPE,                  !Character data type of PE
        2         MAILBOX*5,            !Mailbox name
        2         TASK*80               !Procedure name at remote node
        !
        !Declare external system services and
```

```
      !run time library routines
      !
      INTEGER*4 SYS$CREMBX,           !Create Mailbox, Assign an I/O Channel
      2         LIB$GET_LUN,          !Get Logical Unit Number
      2         LIB$SPAWN,            !Spawn Subprocess
      2         LIB$SIGNAL,           !Signal Exception Condition
      2         LIB$INIT_TIMER,       !Initialize Times and Counts
      2         LIB$SHOW_TIMER        !Show Accumulated Times and Counts
      !
      !Input P,N
      !
      TYPE 1100
1100  FORMAT(X'Enter P(=2,4,8) and N(<=65536): ',$)
      ACCEPT *, P,N
      !
      !Check validity of P and N
      !
      M = INT(ALOG(FLOAT(N)) / ALOG(2.0))
      K = INT(ALOG(FLOAT(P)) / ALOG(2.0))
      IF (P .LT. 2 .OR. P .GT. 8) THEN
        STOP 'P must be 2,4,or 8'
      ELSE IF (P .GE. N) THEN
        STOP 'N must be at least twice P'
      ELSE IF (2**K .NE. P) THEN
        STOP 'P must be a power of 2'
      ELSE IF (N .LT. 2 .OR. N .GT. 65536) THEN
        STOP 'N must be in  2 <= S <= 65536'
      ELSE IF (2**M .NE. N) THEN
        STOP 'N must be a power of 2'
      END IF
      !
      !Calculate initial values
      !
      NOVER2  = N / 2
      NOVERP  = N / P
      NOVER2P = N / (2 * P)
      POVER2  = P / 2
      TASK =  'DOGU'SADAN TUGTEKIN'::'TASK=PE''
      !
      !Input A with complex discrete data points
      !
      CALL GET_POINTS(A,N)
      !
      !Scramble A by bit_reversing
      !
      CALL BIT_REVERSED(A,N,M)
      !
```

```
!Initialize times and counts
!
CALL LIB$INIT_TIMER
!
!Set the lowest bit of the longword MASK (pass as a
!parameter to LIB$SPAWN routine) to run spawned
!subprocesses and CP, concurrently.
!
MASK = IBSET(MASK,0)
!
!Create and run local and remote PEs
!
DO PE=1,POVER2
  !
  !Make up an output file name for the local
  !subprocess to be spawned
  !
  WRITE(CPE,'(I1)') PE
  OUT = 'PE'//CPE//'.OUT'
  !
  !Spawn the local subprocess
  !
  STATUS = LIB$SPAWN('RUN PE',,OUT,MASK)
  IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
  !
  !To create the remote processes, get a logical
  !unit number and request a logical link connection
  !between CP and remote node
  !
  REMOTE_PE = PE + POVER2
  STATUS = LIB$GET_LUN(LUNA(REMOTE_PE))
  IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
  OPEN  (UNIT=LUNA(REMOTE_PE),
2        FILE = TASK,
2        FORM = 'UNFORMATTED',
2        STATUS = 'OLD')
END DO
!
!Get a logical unit number and create a common
!mailbox between CP and all local subprocesses
!
STATUS = LIB$GET_LUN(LUNA(LP))
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
STATUS = SYS$CREMBX(,CHANNEL,,,,,'CONTROL_PROCESSOR')
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
OPEN (UNIT = LUNA(LP),
2     FILE = 'CONTROL_PROCESSOR',
```

```
2     STATUS = 'UNKNOWN',
2     CARRIAGECONTROL = 'LIST',
2     FORM = 'UNFORMATTED')
!
!Transfer the identification data of local
!subprocesses and remote processes
!
DO PE=1,POVER2
  !
  !Transfer the identification data of specified
  !local subprocess to the common mailbox created
  !between CP and all local subprocesses
  !
  WRITE(UNIT=LUNA(LP)) N,P,PE
  !
  !Transfer the identification data of specified
  !remote process over the logical link established
  !between CP and that remote process
  !
  REMOTE_PE = PE + POVER2
  WRITE(UNIT=LUNA(REMOTE_PE)) N,P,REMOTE_PE
END DO
!
!Transfer A to local and remote PEs
!
DO PE=1,POVER2
  !
  !Create a mailbox between CP and
  !the specified local PE to transfer A
  !
  WRITE(CPE,'(I1)') PE
  MAILBOX = 'MBX0'//CPE
  STATUS = SYS$CREMBX(,CHANNEL,,,,,MAILBOX)
  IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
  STATUS = LIB$GET_LUN(LUNA(PE))
  IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
  OPEN (UNIT = LUNA(PE),
2       FILE = MAILBOX,
2       STATUS = 'UNKNOWN',
2       CARRIAGECONTROL = 'LIST',
2       FORM = 'UNFORMATTED')
  !
  !Calculate theoretical beginning and
  !theoretical end of input for that local PE
  !
  TBOI = NOVERP * (PE - 1)
  TEOI = NOVERP *  PE - 1
```

```
  !
  !Transfer the data to that local PE
  !
  DO TRANSFER=TBOI,TEOI
    WRITE(UNIT=LUNA(PE)) A(TRANSFER)
  END DO
  !
  !Calculate theoretical beginning and theoretical
  !end of input for the specified remote PE
  !
  REMOTE_PE = PE + POVER2
  TBOI = NOVERP * (REMOTE_PE - 1)
  TEOI = NOVERP *  REMOTE_PE - 1
  !
  !Do the data transfer over the logical link
  !established between CP and that remote PE
  !
  DO TRANSFER=TBOI,TEOI
    WRITE(UNIT=LUNA(REMOTE_PE)) A(TRANSFER)
  END DO
END DO
!
!Organize the communication of local
!subprocesses and remote processes
!
DO PE=1,POVER2
  REMOTE_PE = PE + POVER2
  DO TRANSFER=1,NOVER2P
    !
    !Get data from specified remote PE
    !
    READ(UNIT=LUNA(REMOTE_PE)) CTEMP
    !
    !Put data to specified local PE
    !
    WRITE(UNIT=LUNA(PE)) CTEMP
    !
    !Get data from specified local PE
    !
    READ(UNIT=LUNA(PE)) CTEMP
    !
    !Put data to specified remote PE
    !
    WRITE(UNIT=LUNA(REMOTE_PE)) CTEMP
  END DO
END DO
!
```

```
!Get the transformation results
!from local and remote PEs
!
DO PE=1,POVER2
  !
  !Calculate theoretical output limits
  !of the specified local PE
  !
  TB001 = NOVER2P * (PE - 1)
  TE001 = NOVER2P *  PE - 1
  TE002 = TE001 + NOVER2
  !
  !Use the mailbox created between specified
  !local PE and CP, to get transformed values
  !
  DO TRANSFER=TB001,TE002,NOVER2
    FINISH = TRANSFER + NOVER2P - 1
    DO OUTPUT=TRANSFER,FINISH
      READ(UNIT=LUNA(PE)) A(OUTPUT)
    END DO
  END DO
  !
  !Calculate theoretical output limits
  !of the specified remote PE
  !
  REMOTE_PE = PE + POVER2
  TB001 = NOVER2P * (REMOTE_PE - 1)
  TE001 = NOVER2P *  REMOTE_PE - 1
  TE002 = TE001 + NOVER2
  !
  !Use the logical link established between specified
  !remote PE and CP, to get transformed values
  !
  DO TRANSFER=TB001,TE002,NOVER2
    FINISH = TRANSFER + NOVER2P - 1
    DO OUTPUT=TRANSFER,FINISH
      READ(UNIT=LUNA(REMOTE_PE)) A(OUTPUT)
    END DO
  END DO
END DO
!
!Show accumulated counts and times
!
CALL LIB$SHOW_TIMER
!
!Output transformation results
!
```

```
CALL PUT_RESULTS(A,N)
END                          !Of control_processor
```

```
C+
C FILE: FFT$BHUYAN:GET.FOR
C
C AUTHOR: Sadan Tugtekin,    CREATION DATE: November, 1987
C-
        SUBROUTINE GET_POINTS(A,N)
        !
        !Declare variables
        !
        COMPLEX*8 A(0:*)                !Complex discrete data points
        REAL*4    X/0.0/,               !Value of x
        2         H                     !Step size of equally spaced data
        CHARACTER ANSWER                !Answer to exp(-x), sin x, cos x ?
        INTEGER*4 N,                    !Number of complex data points
        2         I                     !Incremental variable for loops
        !
        !Declare run time library routine
        !
        INTEGER*4 STR$UPCASE
        !
        !N complex discrete data points can be obtained
        !from the familiar time domain waveforms
        !
        !        exp(-x),
        !        sin x,
        !   and
        !        cos x,
        !
        !with step size  0 < h <= 1.
        !
        !
        !Learn the user request
        !
10      TYPE 110
110     FORMAT(X,'E:exp(-x), S:sin x, C:cos x ? ',$)
        ACCEPT 120, ANSWER
120     FORMAT(A)
        !
        !Convert ANSWER to uppercase character
        !
        CALL STR$UPCASE(ANSWER,ANSWER)
        !
        !Check the validity of ANSWER
        !
        IF (ANSWER .NE. 'E' .AND.
        2   ANSWER .NE. 'S' .AND.
        2   ANSWER .NE .'C') GOTO 10
```

```
        !
20      TYPE 210
210     FORMAT(X,'Enter step size (0 < h <= 1): ',$)
        ACCEPT 220, H
220     FORMAT(F10.6)
        IF (H .LE. 0 .OR. H .GT. 1) GOTO 20
        IF (ANSWER .EQ. 'E') THEN
          !
          !exp(-x)
          !
          DO I=0,N-1
            A(I) = EXP(-X)
            X = X + H
          END DO
        ELSE IF (ANSWER .EQ. 'S') THEN
          !
          !sin x
          !
          DO I=0,N-1
            A(I) = SIN(X)
            X = X + H
          END DO
        ELSE
          !
          !cos x
          !
          DO I=0,N-1
            A(I) = COS(X)
            X = X + H
          END DO
        END IF
        !
        RETURN                  !To control_processor
        END                     !Of get_points
```

```
C+
C FILE: FFT$BHUYAN:SCRAMBLE.FOR
C
C AUTHOR: Sadan Tugtekin,    CREATION DATE: November, 1987
C-
        SUBROUTINE BIT_REVERSED(A,N,M)
        !
        !Declare variables
        !
        COMPLEX*8 A(0:*),               !Complex discrete data points
        2         A1                    !Complex temporary variable
        INTEGER*4 PERMUTE(0:20),        !To permute results
        2         PMT,                  !Index variable for PERMUTE
        2         SKIP,                 !Index
        2         START,                !variables
        2         NOVER2,               !N / 2
        2         N,                    !Number of complex data points
        2         NMINUS1,              !N - 1
        2         M                     !N = 2**M
        !
        !Determine the elements of PERMUTE
        !by powers of 2 from N to 1
        !
        PERMUTE(0) = N
        DO PMT = 1,M
          PERMUTE(PMT) = PERMUTE(PMT-1) / 2
        END DO
        PMT = 0
        !
        !Calculate initial values
        !
        NMINUS1 = N - 1
        START = 1
        NOVER2 = PERMUTE(1)
        SKIP = NOVER2
        !
        !Scramble the array A by bit_reversing
        !
        DO WHILE (START .LT. NMINUS1)
          A1 = A(START)
          A(START) = A(SKIP)
          A(SKIP) = A1
          START = START + 1
          SKIP = SKIP + NOVER2
          DO WHILE (SKIP .GE. PERMUTE(PMT)-1 .AND. START .LT. NMINUS1)
            SKIP = SKIP - PERMUTE(PMT)
            PMT = PMT + 1
```

```
          SKIP = SKIP + PERMUTE(PMT+1)
          DO WHILE (SKIP .LE. PERMUTE(PMT)-1)
            PMT = 0
            DO WHILE (SKIP .LE. NMINUS1)
              IF (START .LT. SKIP) THEN
                GOTO 999
              ELSE
                START = START + 1
                SKIP = SKIP + NOVER2
              END IF
            END DO
          END DO
        END DO
999     END DO
        RETURN                    !To control_processor
        END                       !Of bit_reversed
```

```
C+
C FILE: FFT$BHUYAN:PUT.FOR
C
C AUTHOR: Sadan Tugtekin,    CREATION DATE: November, 1987
C-
      SUBROUTINE PUT_RESULTS(A,N)
      !
      !Declare variables
      !
      COMPLEX*8 A(0:*)                !Complex discrete data points
      CHARACTER ANSWER                !Answer to Screen, Disk, End?
      INTEGER*4 N,                    !Number of complex data points
     2          I                    !Incremental variable for loops
      !
      !Declare run time library routine
      !
      INTEGER*4 STR$UPCASE
      !
      !Learn the user request to where the results
      !will be output
      !
10    TYPE 110
110   FORMAT(X,'S:Screen, D:Disk, E:End? ',$)
      ACCEPT 120, ANSWER
120   FORMAT(A)
      !
      !Convert ANSWER to uppercase character
      !
      CALL STR$UPCASE(ANSWER,ANSWER)
      !
      !Check the validity of ANSWER
      !
      IF (ANSWER .NE. 'S' .AND.
     2    ANSWER .NE. 'D' .AND.
     2    ANSWER .NE .'E') GOTO 10
      !
      !Output the results to Screen or Disk, or End
      !the program
      !
      IF (ANSWER .EQ. 'S') THEN
        !
        !Output the results to Screen
        !
        TYPE *, ' *** FFT RESULTS ***'
        DO I=0,N-1
          TYPE 130, I,REAL(A(I)),AIMAG(A(I))
130       FORMAT(X,'A(',I4.4,') = (',F14.8,',',F14.8,')')
```

```
  END DO
ELSE IF (ANSWER .EQ. 'D') THEN
  !
  !Output the results to Disk
  !
  TYPE *, 'Wait, I''m writing to FFT.DAT'
  OPEN (UNIT=1,
2       FILE='FFT.DAT',
2       STATUS='NEW',
2       RECL=80)
  WRITE(1,*) ' *** FFT RESULTS ***'
  DO I=0,N-1
    WRITE(1,130) I,REAL(A(I)),AIMAG(A(I))
  END DO
END IF
!
!If ANSWER is 'E' then End the program
!
RETURN                    !To control_processor
END                       !Of put_results
```

```
C+
C FILE: FFT$BHUYAN:PE.FOR
C
C AUTHOR: Sadan Tugtekin,    CREATION DATE: November, 1987
C-
      PROGRAM   PROCESSING_ELEMENT
      !
      !Declare variables
      !
      COMPLEX*8 A(0:32767),          !Complex discrete data points
      2         CTEMP,               !Complex temporary variable
      2         OUTGOING,            !Outgoing data
      2         W0,                  !First value of W
      2         W1                   !Second value of W
      REAL*4    TPIE,                !Two pie
      2         TEMP                 !Real temporary value
      INTEGER*4 STATUS,              !Conditions returned
      2         CHANNEL,             !I/O channel for mailbox
      2         LUNA(0:8),           !Array for logical unit numbers
      2         LP/0/,               !Index variable for LUNA
      2         N,                   !Number of complex data points
      2         NOVER2,              !N / 2
      2         NOVERP,              !N / P
      2         NOVER2P,             !N / (2 * P)
      2         P,                   !Total number of PE
      2         POVER2,              !P / 2
      2         PE,                  !Processing element number
      2         DIFFERENCE,          !Difference between two PE
      2         PARTNER,             !Communication partner of PE
      2         RANGE,               !Twice DIFFERENCE
      2         PEMOD,               !Value of MOD(PE,RANGE)
      2         M,                   !N = 2**M
      2         K,                   !P = 2**K
      2         MMINUSK,             !M - K
      2         BOI,                 !Beginning of input
      2         EOI,                 !End of input
      2         TBOI,                !Theoretical beginning of input
      2         TEOI,                !Theoretical end of input
      2         STAGE,               !Stage number
      2         WP0,                 !First power of W
      2         WP1,                 !Second power of W
      2         OLDWP0,              !Previous value of WP0
      2         OLDWP1,              !Previous value of WP1
      2         SKIP,                !Index
      2         LENGTH,              !variables
      2         WLENGTH,             !to
      2         START,               !calculate
```

```
2          FINISH,                !butterfly
2          FIRST,                 !F
2          SECOND,                !F
2          TRANSFER,              !T
2          OUTPUT                 !operations
CHARACTER  MAILBOX*5,             !Mailbox name
2          NODE*4,                !Network node name
2          CPE,                   !Character data type of PE
2          CPARTNER               !Character data type of PARTNER
LOGICAL*1  REMOTE/.FALSE./        !Check for remote node
!
!Declare external system services and
!run time library routines
!
INTEGER*4  SYS$CREMBX,            !Create Mailbox, Assign an I/O Channel
2          LIB$GET_LUN,           !Get Logical Unit Number
2          LIB$GETSYI,            !Get System-Wide Information
2          LIB$SIGNAL,            !Signal Exception Condition
2          LIB$INIT_TIMER,        !Initialize Times and Counts
2          LIB$SHOW_TIMER         !Show Accumulated Times and Counts
!
!Include $SYIDEF for use of
!routine LIB$GETSYI
!
INCLUDE '($SYIDEF)'
!
!Initialize times and counts
!
CALL LIB$INIT_TIMER
!
!Obtain network node name of the processor
!
STATUS = LIB$GETSYI(SYI$_NODENAME,,NODE,,,)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
IF (NODE .NE. 'BATI') REMOTE = .TRUE.
!
!Get a logical unit number to input
!the identification data of this PE
!
STATUS = LIB$GET_LUN(LUNA(LP))
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
IF (.NOT. REMOTE) THEN
  !
  !Create (or assign an I/O channel to) the common
  !mailbox for this local PE to communicate with CP
  !
  STATUS = SYS$CREMBX(,CHANNEL,,,,,'CONTROL_PROCESSOR')
```

```
  IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
  OPEN (UNIT = LUNA(LP),
2       FILE = 'CONTROL_PROCESSOR',
2       STATUS = 'UNKNOWN',
2       CARRIAGECONTROL = 'LIST',
2       FORM = 'UNFORMATTED')
ELSE
  !
  !Complete the logical link request for
  !this remote PE to communicate with CP
  !
  OPEN  (UNIT=LUNA(LP),
2         FILE = 'SYS$NET',
2         FORM = 'UNFORMATTED',
2         STATUS = 'OLD')
END IF
!
!Input the identification data
!of this PE from CP
!
READ(UNIT=LUNA(LP)) N,P,PE
!
!Calculate initial values
!
TPIE = 8.0 * ATAN(1.0)
M = INT(ALOG(FLOAT(N)) / ALOG(2.0))
K = INT(ALOG(FLOAT(P)) / ALOG(2.0))
MMINUSK = M - K
NOVER2  = N / 2
NOVERP  = N / P
NOVER2P = N / (2 * P)
POVER2  = P / 2
BOI = 0
EOI = NOVERP - 1
TBOI = NOVERP * (PE - 1)
TEOI = NOVERP *  PE - 1
WRITE(CPE,'(I1)') PE
!
!Input A from CP
!
IF (.NOT. REMOTE) THEN
  !
  !Create a mailbox between this
  !local PE and CP to input A
  !
  LP = PE
  MAILBOX = 'MBX0'//CPE
```

```
  STATUS = SYS$CREMBX(,CHANNEL,,,,,MAILBOX)
  IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
  STATUS = LIB$GET_LUN(LUNA(LP))
  IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
  OPEN (UNIT = LUNA(LP),
2       FILE = MAILBOX,
2       STATUS = 'UNKNOWN',
2       CARRIAGECONTROL = 'LIST',
2       FORM = 'UNFORMATTED')
ELSE
  !
  !Use the completed logical link established
  !between this remote PE and CP to input A
  !
  LP = 0
END IF
!
!Do the data transfer
!
DO TRANSFER=BOI,EOI
  READ(UNIT=LUNA(LP)) A(TRANSFER)
END DO
!
!Calculate M - K stages butterfly FFT operations
!which have no data transfer among PEs
!
DO STAGE=1,MMINUSK
  !
  !Setup required values of transform split
  !
  SKIP = 2**STAGE
  LENGTH = SKIP / 2
  WP0 = -1
  WP1 = WP0 + LENGTH
  FINISH = BOI + (LENGTH - 1)
  TEMP = TPIE / SKIP
  !
  !Start iterative FFT computations
  !
  DO START=BOI,FINISH
    !
    !Determine W0 and W1 values
    !
    WP0 = WP0 + 1
    WP1 = WP1 + 1
    W0 = CEXP(CMPLX(0.0,TEMP * WP0))
    W1 = CEXP(CMPLX(0.0,TEMP * WP1))
```

```
    !
    !Do the butterfly operations
    !
    DO FIRST=START,EOI,SKIP
      SECOND = FIRST + LENGTH
      CTEMP = A(FIRST) + W1 * A(SECOND)
      A(FIRST) = A(FIRST) + W0 * A(SECOND)
      A(SECOND) = CTEMP
    END DO
  END DO
END DO
!
!Calculate butterfly FFT operations after M - K stage
!
DO STAGE=MMINUSK+1,M
  !
  !Setup required values of transform split
  !and calculate DIFFERENCE, RANGE, and PEMOD
  !
  SKIP = 2**STAGE
  WLENGTH = SKIP / 2
  DIFFERENCE = P * 2**(STAGE - 1) / N
  RANGE = DIFFERENCE * 2
  IF (PE .GT. RANGE) THEN
    PEMOD = MOD(PE,RANGE)
    IF (PEMOD .EQ. 0) PEMOD = RANGE
  ELSE
    PEMOD = PE
  END IF
  !
  !Determine communication partner of PE and
  !do the necessary data transfer
  !
  IF (PEMOD .LE. DIFFERENCE) THEN
    !
    !Partner's PE_number is greater.
    !
    PARTNER = PE + DIFFERENCE
    !
    !Is the partner at remote node or in local node?
    !
    IF (PE .LE. POVER2 .AND. PARTNER .GT. POVER2) THEN
      !
      !Partner is at remote node and this PE is in
      !local node.
      !To communicate with partner;
      !  - first, use the mailbox created between
```

```
      !    this local PE and CP, to communicate with CP,
      !  - and then CP uses the logical link established
      !    between partner and CP.
      !
      LP = PE
    ELSE
      !
      !Both partner and this PE are in the same node.
      !(It doesn't matter, the node is local or remote.)
      !Create a mailbox to communicate with partner.
      !
      LP = PARTNER
      WRITE(CPARTNER,'(I1)') PARTNER
      MAILBOX = 'MBX'//CPE//CPARTNER
      STATUS = SYS$CREMBX(,CHANNEL,,,,,MAILBOX)
      IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
      STATUS = LIB$GET_LUN(LUNA(LP))
      IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
      OPEN (UNIT = LUNA(LP),
     2      FILE = MAILBOX,
     2      STATUS = 'UNKNOWN',
     2      CARRIAGECONTROL = 'LIST',
     2      FORM = 'UNFORMATTED')
    END IF
    !
    !Do the data transfer
    !
    START = BOI + NOVER2P
    DO TRANSFER=START,EOI
      OUTGOING = A(TRANSFER)
      READ(UNIT=LUNA(LP)) A(TRANSFER)
      WRITE(UNIT=LUNA(LP)) OUTGOING
    END DO
    !
    !Calculate initial values of WP0 and WP1
    !
    IF (STAGE .EQ. MMINUSK+1) OLDWP0 = TBOI
    OLDWP1 = OLDWP0 + WLENGTH
    WP0 = MOD(OLDWP0,SKIP)
    WP1 = WP0 + WLENGTH
    !
  ELSE
    !
    !Partner's PE_number is less.
    !
    PARTNER = PE - DIFFERENCE
    !
```

```
      !Is the partner at remote node or in local node?
      !
      IF (PE .GT. POVER2 .AND. PARTNER .LE. POVER2) THEN
        !
        !Partner is in local node and this PE is at
        !remote node.
        !To communicate with partner;
        !  - first, use the logical link established
        !    between this remote PE and CP,
        !  - and then CP uses the mailbox created between
        !    partner and CP, to communicate with partner
        !
        LP = 0
      ELSE
        !
        !Both partner and this PE are in the same node.
        !(It doesn't matter, the node is local or remote.)
        !Create a mailbox to communicate with partner.
        !
        LP = PARTNER
        WRITE(CPARTNER,'(I1)') PARTNER
        MAILBOX = 'MBX'//CPARTNER//CPE
        STATUS = SYS$CREMBX(,CHANNEL,,,,,MAILBOX)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
        STATUS = LIB$GET_LUN(LUNA(LP))
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
        OPEN (UNIT = LUNA(LP),
     2        FILE = MAILBOX,
     2        STATUS = 'UNKNOWN',
     2        CARRIAGECONTROL = 'LIST',
     2        FORM = 'UNFORMATTED')
      END IF
      !
      !Do the data transfer
      !
      FINISH = BOI + (NOVER2P - 1)
      DO TRANSFER=BOI,FINISH
        WRITE(UNIT=LUNA(LP)) A(TRANSFER)
        READ(UNIT=LUNA(LP)) A(TRANSFER)
      END DO
      !
      !Calculate initial values of WP0 and WP1
      !
      IF (STAGE .EQ. MMINUSK+1) OLDWP1 = TEOI - (LENGTH - 1)
      OLDWP0 = OLDWP1 - WLENGTH
      WP1 = MOD(OLDWP1,SKIP)
      WP0 = WP1 - WLENGTH
```

```
    !
  END IF
  !
  !Start iterative FFT computations
  !
  TEMP = TPIE / SKIP
  DO FIRST=BOI,EOI-LENGTH
    !
    !Determine W0 and W1 values
    !
    W0 = CEXP(CMPLX(0.0,TEMP * WP0))
    W1 = CEXP(CMPLX(0.0,TEMP * WP1))
    !
    !Do the butterfly operations
    !
    SECOND = FIRST + LENGTH
    CTEMP = A(FIRST) + W1 * A(SECOND)
    A(FIRST) = A(FIRST) + W0 * A(SECOND)
    A(SECOND) = CTEMP
    !
    !Calculate next powers of W
    !
    WP0 = WP0 + 1
    WP1 = WP1 + 1
  END DO
END DO
!
!Transfer the FFT results of this PE to CP
!
IF (.NOT. REMOTE) THEN
  !
  !Use the mailbox created between this
  !local PE and CP to transfer the results
  !
  LP = PE
ELSE
  !
  !Use the logical link established between
  !this remote PE and CP to transfer the results
  !
  LP = 0
END IF
!
!Do the data transfer
!
DO OUTPUT=BOI,EOI
  WRITE(UNIT=LUNA(LP)) A(OUTPUT)
```

```
END DO
!
!Show accumulated counts and times
!
CALL LIB$SHOW_TIMER
END                         !Of processing_element
```

```
$! FILE: EET$BERGLAND:PE.COM
$!
$! ABSTRACT:
$!      PE.COM is executed for each connect request
$!      to establish a logical link between the CP in
$!      local node and the PE's in remote node.
$!
$! AUTHOR: Sadan Tugtekin,    CREATION DATE: November, 1987
$!
$ SET PROCESS/PRIVILEGE=GRPNAM
$ DEFINE/TABLE=LNM$PROCESS_DIRECTORY LNM$TEMPORARY_MAILBOX LNM$GROUP
$ RUN PE
$ PURGE/KEEP=4 NETSERVER.LOG
$ EXIT                    ! End of PE.COM
```