addresses have the same access time as the first item. If this problem can be solved, the dynamic memory will act like a drum with a latency of $\log_2 N$, and a transfer rate of one word per unit time.

References

[1] G. M. Adel'son-Vel'skii and E. M. Landis, "An algorithm for the organization of information," *Dokl. Akad. Nauk USSR Math.*, vol. 146, pp. 263–366, 1962.
[2] K. E. Batcher, "Sorting networks and their application," in *1968 Spring Joint Comput. Conf., AFIPS Conf. Proc.*, vol. 32. Washington, D.C.: Thompson, 1968, pp. 307–314.
[3] N. G. de Bruijn, "A combinatorial problem," *Proc. Kon. Ned. Akad. Wetensch*, vol. 49, pp. 758–764, 1946.
[4] C. C. Foster, in *Proc. Ass. Comput. Mach. Nat. Conf.*, 1965, pp. 192–205.
[5] C. W. Gear, *Computer Organization and Programming*. New York: McGraw-Hill, 1969.
[6] M. Pease, "An adaptation of the fast Fourier transform for parallel processing," *J. Ass. Comput. Mach.*, vol. 15, pp. 252–264, Apr. 1968.
[7] H. S. Stone, "Parallel processing with the perfect shuffle," *IEEE Trans. Comput.*, vol. C-20, pp. 153–161, Feb. 1971.

A Parallel Implementation of the Fast Fourier Transform Algorithm

G. D. BERGLAND, MEMBER, IEEE

Abstract—For many real-time signal processing problems, correlations, convolutions, and Fourier analysis must be performed in special-purpose digital hardware. At relatively high levels of performance, it becomes necessary for this hardware to perform some of its computations in parallel.

A parallel FFT algorithm is described that segments the fast Fourier transform algorithm into groups of identical parallel operations that can be performed concurrently and independently.

A hardware implementation of the algorithm is described in the context of the parallel element processing ensemble (PEPE) previously described by Githens [7], [8].

Index Terms—Cooley–Tukey algorithm, discrete Fourier transform, fast Fourier transform, Good's prime factor algorithm, real-time spectrum analysis, rightly parallel computing, special-purpose computer.

Introduction

THE introduction of the fast Fourier transform (FFT) algorithm has had a substantial impact on the field of digital signal processing. This algorithm provides an efficient method of performing the convolutions needed to implement digital filters, the correlations needed to implement matched filters, and the Fourier analysis needed for producing spectrograms. These correlation, convolution, and spectral analysis operations form the basis for radar signal processing, sonar signal processing, speech analysis, seismic signal processing, pattern recognition, image enhancement, and countless other operations.

Since the FFT algorithm is basic to implementing all of these operations, the extent to which they can be done in real time has in many cases been limited by the rate at which the fast Fourier transform algorithm can be executed. High performance requirements led to the design of special-purpose FFT processors that included several arithmetic units operating in parallel. Several methods of implementing the FFT algorithm that permit parallel operations were described in [1]. An additional parallel implementation was described by Bergland and Wilson in [2].

The parallel FFT algorithm described herein combines components of Bergland and Wilson's algorithm with a variation of the Good prime-factor algorithm described in [4] and [10]. The resulting algorithm computes an N-term discrete Fourier transform in a parallel manner when N can be expressed as the product of two integers R and S which are relatively prime. For ease of presentation, the special case where R is an odd integer and S is a power of two is considered here.

The parallel FFT algorithm that is described provides a convenient method of segmenting the computations of an N-term FFT for implementation in an odd number of small FFT processors working in parallel. This parallelism can be introduced either to increase the maximum length of the series that can be transformed or to decrease the execution time. The algorithm is most efficient when R is considerably smaller than S. In many applications the examples of R equal to 3, 5, and 7 are of special interest.

A machine implementation of the parallel FFT algorithm in the context of the Githen's parallel element processing ensemble (PEPE) [7], [8] is also described. This computer consists of an ensemble of identical processing elements all driven by a common control program.

Good's Prime-Factor Algorithm

The fast Fourier transform algorithm gives an efficient method of evaluating the expression

Manuscript received June 8, 1971; revised August 20, 1971.
The author is with Bell Telephone Laboratories, Inc., Whippany, N. J. 07981.

$$X(j) = \sum_{k=0}^{N-1} A(k) W_N^{jk} \tag{1}$$

where $W_N = \exp(2\pi i/N)$, $i = \sqrt{-1}$, $j = 0, 1, 2, \cdots, N-1$, and N can be expressed as the product of an arbitrary number of positive integers.

When only two factors of N are involved, i.e., $N = R \cdot S$, the resulting expressions can be written in the form

$$A_1(j_0, k_0) = \sum_{k_1=0}^{R-1} A(k_1, k_0) W_R^{j_0 k_1} \tag{2}$$

$$\hat{A}_1(j_0, k_0) = A_1(j_0, k_0) W_N^{j_0 k_0} \tag{3}$$

$$X(j_1, j_0) = \sum_{k_0=0}^{S-1} \hat{A}_1(j_0, k_0) W_S^{j_1 k_0} \tag{4}$$

where $j_0, k_1 = 0, 1, \cdots, R-1$; $k_0, j_1 = 0, 1, \cdots, S-1$; $j = j_1 R + j_0$, and $k = k_1 S + k_0$. Note that (2) represents a set of R point transforms, (4) represents a set of S point transforms, and (3) represents the rereferencing (or twiddling [6]) of the intermediate results.

In the Good prime-factor algorithm (see Cooley *et al.* [4]), j and k are defined such that

$$\begin{aligned} k &= Rk_0 + Sk_1, && \bmod N,\ 0 \le k < N \\ k_0 &= k \cdot R_S (\bmod S), && \bmod S,\ 0 \le k_0 < S \\ k_1 &= k \cdot S_R (\bmod R), && \bmod R,\ 0 \le k_1 < R \end{aligned} \tag{5}$$

and

$$\begin{aligned} j &= S \cdot S_R j_0 + R \cdot R_S j_1, && \bmod N,\ 0 \le j < N \\ j_0 &= j (\bmod R), && \bmod R,\ 0 \le j_0 < R \\ j_1 &= j (\bmod S), && \bmod S,\ 0 \le j_1 < S \end{aligned} \tag{6}$$

where S_R and R_S are solutions of

$$\begin{aligned} S \cdot S_R &= 1 (\bmod R), && S_R < R \\ R \cdot R_S &= 1 (\bmod S), && R_S < S, \end{aligned} \tag{7}$$

respectively.

Thus (1) becomes

$$X(j_1, j_0) = \sum_{k_0=0}^{S-1} \sum_{k_1=0}^{R-1} A(k_1, k_0) W_N^{jk} \tag{8}$$

where

$$\begin{aligned} W_N^{jk} &= W_N^{(S \cdot S_R j_0 + R \cdot R_S j_1)(Rk_0 + Sk_1)} \\ &= (W_N^{RS})^{(S_R j_0 k_0 + R_S j_1 k_1)} (W_S^{R \cdot R_S})^{j_1 k_0} (W_R^{S \cdot S_R})^{j_0 k_1}. \end{aligned} \tag{9}$$

Since $W_N^{RS} = 1$, $W_R^{S \cdot S_R} = W_R$ and $W_S^{R \cdot R_S} = W_S$, we have

$$W_N^{jk} = W_R^{j_0 k_1} W_S^{j_1 k_0}. \tag{10}$$

Thus we can write

$$X(j_1, j_0) = \sum_{k_0=0}^{S-1} \underbrace{\sum_{k_1=0}^{R-1} A(k_1, k_0) W_R^{j_0 k_1}}_{A_1(j_0, k_0)} W_S^{j_1 k_0} \tag{11}$$

or equivalently

$$A_1(j_0, k_0) = \sum_{k_1=0}^{R-1} A(k_1, k_0) W_R^{j_0 k_1} \tag{12}$$

$$X(j_1, j_0) = \sum_{k_0=0}^{S-1} A_1(j_0, k_0) W_S^{j_1 k_0}. \tag{13}$$

Equation (12) represents a set of R point transforms and (13) represents a set of S point transforms. Note that they are similar in form to (2) and (4); however, the rereferencing factors of (3) have been eliminated. By redefining the j_0, j_1, k_0, and k_1 indices, the rereferencing has been done by forming cyclic permutations of the subsequences being transformed and by allowing subsequences of the results to be permuted from their normal order. Since permutation in one domain corresponds to rereferencing in the other domain, the permutations can be chosen to eliminate the rereferencing operations that were required in (3). (See the explanation in Gold and Radar [9, ch. 6] letting $a = R - S_R$ and $b = R_S$.)

The Parallel FFT Algorithm

In developing the parallel FFT (PFFT) algorithm, note that the R, S point transforms of (13) can be evaluated independently and in parallel via any standard FFT algorithm once the A_1 terms are available. Note also that the S, R point transforms that compute these A_1 terms can be computed on the fly by using (12) directly. The samples of the original A series are used in their normal order.

In Fig. 1, a machine organization is shown in which R separate computers (or small FFT processors) operate under the control of a common host computer. The host computer provides an interface with the outside world, and provides buffering for the input and output data. It can also provide data and instructions to each of the R small computers. The N term series initially stored in the A array of the host computer, will be replaced by its finite discrete Fourier transform (DFT).

Each of the R small computers is capable of performing the S point transforms of (13) on the data stored in its P array under the control of the host computer. Each of the sums of (12), are accumulated as the N term series is presented sequentially to the entire ensemble of computers. One memory location in each of the computing elements is reserved for accumulating a term of each of the S transforms indicated by (12). Since R Fourier coefficients will be computed for each data set, each of the computers will "specialize" in computing a particular Fourier coefficient. For example, computer number 0 will compute the dc term for all S of the data sets, computer number 1 will compute all of the first harmonic Fourier coefficients, computer number 2 all of the second harmonic coefficients, and so on.

The procedure for accumulating these coefficients is for each computer to simply accept every sample as it is presented to all of the computers in parallel, to multiply it by the appropriate power of W (previously stored in its W array), and to add the resulting product to the correct location in its P array. Since all of the small computing elements always access the same location, the algorithm proceeds in

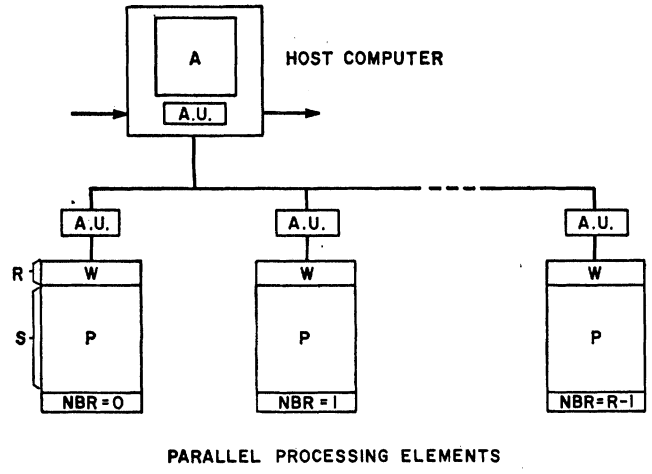

Fig. 1. Parallel element processing ensemble (PEPE).

parallel, performing identical operations in each of the elements.

For given values of R and S, the order in which the W array and the P array are accessed can be seen by noting the correspondence between the A values indexed by k and the A values as indexed by k_0 and k_1.

For an example of $N=R\cdot S=3\cdot 4=12$, this correspondence can be seen in Fig. 2. For this example, the W arrays would be loaded as shown in Fig. 3.

Note that the zeroth element has those weights needed for it to compute the dc term of any three point series, the first element has those weights needed to compute the first harmonic, and so on. With this arrangement, for any given $A(k)$, $k=0, 1, \cdots, N-1$, the table in Fig. 2 tells both the address of the W term to be used and the location in the P array into which the product is to be added. The $A(1)$ term, for example, is multiplied by the $W(1)$ term and then added to $P(3)$. $A(2)$ is multiplied by $W(2)$ and added to $P(2)$, etc. The reason for this correspondence is apparent from (12) since k_1 is the index summed over, and k_0 denotes which transform is being performed.

When all of the input terms have been processed, the computing element memories have accumulated the intermediate results shown in Fig. 4.

From (13), it is clear that these sets of A_1 terms can next each be transformed independently and in parallel. These transforms over the k_0 index can be performed using any standard FFT algorithm that employs an in-place reordering technique.

After performing these R independent S point transforms, all that remains is to read out the Fourier coefficients of the N term series in order. The correspondence between the $X(j)$ notation and the $X(j_1, j_0)$ notation is indicated in Fig. 5. Note that this table locates the $X(j)$ terms with respect to both the number of the elements that they are in and their address in the P array. Reading the $X(j)$ terms out in order of ascending frequency can be done conveniently by forming a mod R counter that identifies the value of j_0 and a mod S counter that identifies the value of j_1.

The number of complex multiplication operations that must be performed in sequence using this algorithm is the

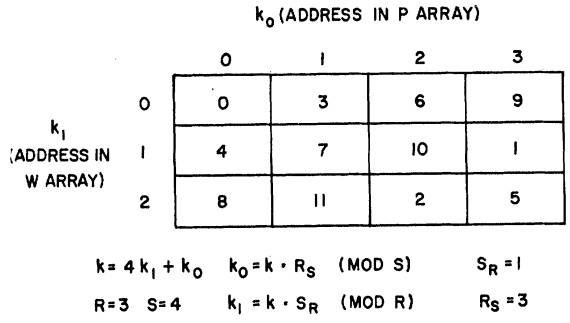

Fig. 2. The mapping of k into k_0 and k_1.

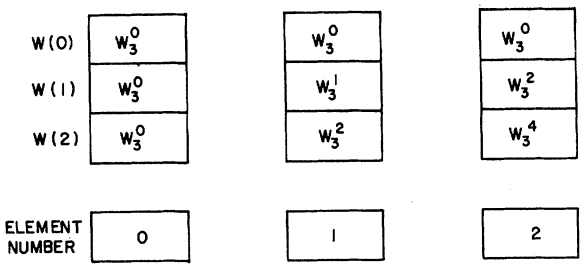

Fig. 3. The contents of the W arrays for the example of $N=3\cdot 4=12$.

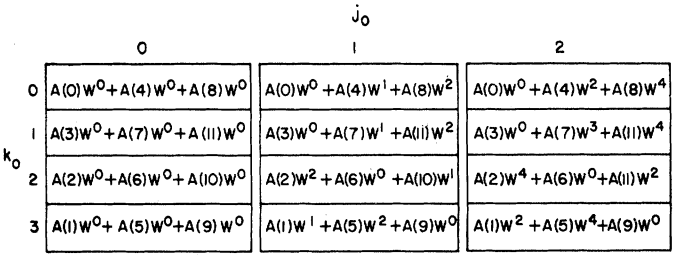

Fig. 4. Intermediate results formed in the P array.

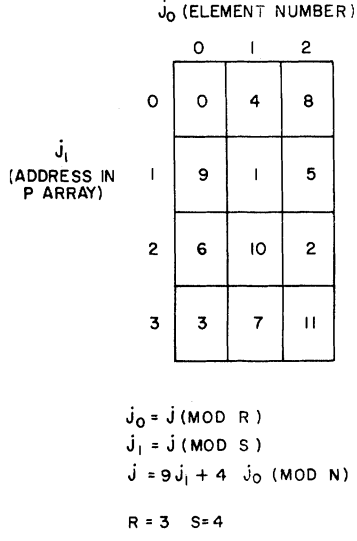

Fig. 5. Location of j terms as a function of j_0 and j_1.

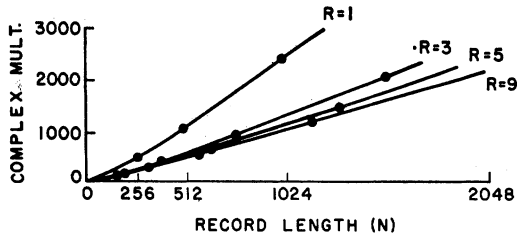

Fig. 6. Number of complex multiplications required by the PFFT algorithm as a function of record length.

sum of the operations in (12) and the $((\log_2 S)/3-1)S+1$ operations required in evaluating (13) (see [3]). Thus the PFFT algorithm requires that a total of $(R+(\log_2 S)/3-2)S+1$ complex multiplications be performed sequentially per execution. This equation is plotted for several values of R and N in Fig. 6.

Hardware Implementations

Parallel Element Processing Ensemble (PEPE) Implementation

One attractive hardware implementation of the parallel FFT algorithm makes use of the parallel element processing ensemble (PEPE) described by Githens [7], [8]. PEPE consists of an ensemble of small computing elements (similar to those of Fig. 1) that operate under the control of a standard general-purpose computer that acts as its host. It is particularly well suited to problems where a series of similar operations must be performed rapidly on a large number of data sets.

Each processing element contains its own memory and arithmetic unit, but the same instruction stream is presented to all of the elements simultaneously. On the basis of data stored within each element memory, each processing element is capable of either executing a particular set of instructions or ignoring it. A medium-scale integration (MSI) PEPE design has been completed in which each processing element would occupy a volume of less than 0.03 ft^3 and be capable of executing approximately 1 000 000 instructions per second (1.0 MIPS). An integrated circuit (IC) feasibility model of PEPE, capable of executing approximately 250 000 instructions per second, has been operational since May 1971.

Using PEPE IC model execution times,[1] P–FFT run times are plotted for various values of R and S in Fig. 7. These execution times plotted as a function of N (where permitted values are shown as dots along the lines) are shown in Fig. 8. By using an MSI implementation of PEPE, the execution times could be reduced by approximately a factor of four.

A PEPE implementation of an FFT processor is especially well suited to multiple channel Fourier analysis. One host computer could control a large number of identical elements, all operating in parallel. Since the cost per element would normally drop as the number of elements produced increased, this machine organization becomes quite appealing when implemented with MSI technology.

[1] This represents the time actually spent by the IC model in executing the PFFT algorithm. It excludes several host computer overhead operations peculiar to the present IC model—host computer interconnection that will not be present in the MSI PEPE configuration.

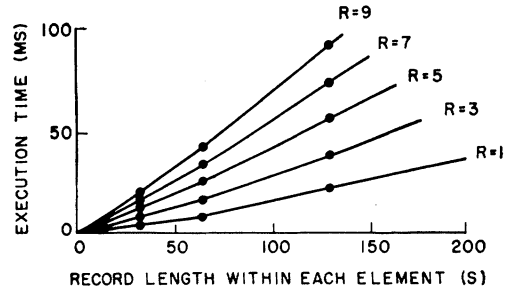

Fig. 7. PFFT estimated execution time for an MSI PEPE.

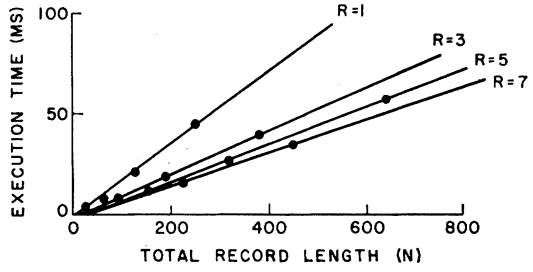

Fig. 8. PFFT estimated execution time for an MSI PEPE.

Other Machine Implementations

The PFFT algorithm is also useful for permitting the modular growth of existing FFT processors. Given a number of identical processors that can each perform 1024 point transforms, these processors could be interconnected to perform 3072 point transforms, 5120 point transforms, etc. Thus the algorithm provides a method of substantially increasing the record lengths that can be processed for only a moderate increase in execution time. As noted previously, the primary usefulness of this method will be when R is much smaller than S.

While there are several ways of introducing parallelism into the FFT algorithm, the PFFT algorithm seems well suited to meeting performance requirements that are only slightly out of the reach of a sequential processor, without going to the more expensive cascade processor architecture. Another possible advantage in many applications is that this algorithm permits the addition of parallel computations in the FFT algorithm, while still retaining the capability of doing block rescaling operations.

For a given constant value of N, a new sequential or cascade processor designed around a standard FFT algorithm would perform fewer arithmetic operations, and therefore should be less expensive than a processor designed to use the PFFT algorithm. In many signal processing systems, however, there is often the requirement for analyzing data to several different resolutions (and therefore for using several different values of N). When the record lengths being analyzed are long, and the finest resolution is only needed on an occasional basis, the PFFT algorithm may still be an economical way of meeting this requirement.

As an example, consider a system that must usually pro-

cess 60 independent signals in real time with the resolution implied by 4096 point records. With few additions, this system could be reconfigured to process up to 20 of these signals with the resolution implied by 12 288 point records via the PFFT algorithm. Thus it provides a convenient way for trading off channels for increased resolution.

Conclusions

With the advent of large-scale integration (LSI) and medium-scale integration (MSI) circuit technology, it is now economical to implement large numbers of identical modules that can operate in parallel. The PFFT algorithm gives the special-purpose computer designer an additional degree of flexibility that in several cases allows a favorable tradeoff to be made between parallelism and performance.

Without an LSI or an MSI implementation, there are still a number of signal processing applications that require parallelism simply to meet the real-time constraints of the problem or to permit the analysis of extremely long records. When these cases are the exception rather than the rule, it may be more economical to combine existing or previously designed processors via the PFFT algorithm than to design and implement a completely new processor. In multichannel systems it can be used to permit a tradeoff between the number of channels and the resolution.

From an efficiency viewpoint, the main utility of the PFFT algorithm lies in applications where R is considerably smaller than S. Examples where R is equal to 3 or 5 should be of particular interest since they can bridge the gap between the sequential processor with no parallelism and the cascade processor with an inherent parallelism of $\log_2 N$. In other applications, the main advantage might be that it provides a convenient method of introducing a factor of three.

Note that the improvement in the performance of the algorithm is greatest for small values of R and large values of S. As R becomes larger and larger, the $N-S$ operations associated with evaluating (12) take a larger and larger percentage of the total execution time of the algorithm.

In a special-purpose machine, some gains could be made by introducing additional hardware so that (12) and (13) could be computed concurrently on adjacent data sets. This would amount to requiring that two arithmetic units and instruction streams be executed in each element concurrently. While this would be a substantial change from the PEPE architecture shown in Fig. 1, it might be worth considering for special applications.

Acknowledgment

The very capable programming assistance of Miss N. M. Dunn, Mrs. J. A. Malsbury, and Miss B. L. Moss is gratefully acknowledged.

References

[1] G. D. Bergland, "Fast Fourier transform hardware implementations—An overview," *IEEE Trans. Audio Electroacoust.*, vol. AU-17, pp. 104–108, June 1969.
[2] G. D. Bergland and D. E. Wilson, "An FFT algorithm for a global, highly parallel processor," *IEEE Trans. Audio Electroacoust.*, vol. AU-17, pp. 125–127, June 1969.
[3] G. D. Bergland, "A fast Fourier transform algorithm using base 8 iterations," *Math. Comput.*, vol. 22, pp. 275–279, Apr. 1968.
[4] J. W. Cooley, P. A. W. Lewis, and P. D. Welch, "The fast Fourier transform algorithm and its applications," IBM Res. Paper RC-1743, Feb. 1967.
[5] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Math. Comput.*, vol. 19, pp. 297–301, Apr. 1965.
[6] W. M. Gentlemen and G. Sande, "Fast Fourier transforms for fun and profit," in *1966 Fall Joint Comput. Conf., AFIPS Conf. Proc.*, vol. 29. Washington, D. C.: Spartan, 1966.
[7] J. A. Githens, "A fully parallel computer for radar data processing," presented at the Nat. Aerosp. Electron. Conf., May 1970.
[8] ——, "An associative, highly-parallel computer for radar data processing," presented at the Symp. Parallel Processor Syst., Technol. and Appl., June 1969.
[9] B. Gold and C. M. Rader, *Digital Processing of Signals.* New York: McGraw-Hill, 1969.
[10] I. J. Good, "The relationship between two fast Fourier transforms," *IEEE Trans. Comput.*, vol. C-20, pp. 310–317, Mar. 1971.