MILITARY TECHNICAL COLLEGE

CAIRO - EGYPT

IMAGE PREDICTIVE VECTOR CODING USING NEURAL NETWORK WITH NEW TRAINING

A. S. EL-Fishawy*, R. M. Atta**, M. Hadhoud*** and K. H. Awadalla****

ABSTRACT

Image data compression is used to reduce the transmission rates or the amount of information to be sent or stored without greatly affecting the quality of the reconstructed images. Many techniques, such as vector quantization, have been used in order to satisfy these requirements. Using the neural network techniques, rather than the traditional technique in this subject decreases the loss of image information and hence enhances its quality specially at low bit rates.

Some of the neural network models proposed for image data compression still have some defects specially, when the actual images are not included in the training phase of the network.

In this paper a new training set is proposed to be used in the training set of the Kohonen self-organizing map when it used in image data compression applications to increase its efficiency. This proposed training set is statistically dependent with all images independent of their types. A predictive vector quantization using both Kohonen self-organizing map and the adaptive differential pulse code modulation with the linear neural network predictor is also introduced.

The simulation results this paper show that the performance of the proposed techniques is much better than that used by others.

* Assistant Prof. ** Assistant Lecturer *** Associate Prof. **** Prof.

Faculty of Electronic Engineering, Comm. Eng. Dept., Menouf, Egypt

1. INTRODUCTION

Digital coding of information is motivated by the need to compress information to improve the used rates of exiting transmission and storage devices without degrading the information quality after the coding process.

Many techniques are used to reduce transmission rates by reducing the redundancy or the correlation within the image pixels and then this new smaller uncorrelated set of data is transmitted. One of the effective techniques in image data compression is vector quantization (VQ) [1] . Vector quantization has been broadly considered for some coding in many digital communication systems due to the drastic bit rate reduction that it can achieve, while preserving image quality [2] . VQ is actually quantizing not just the parameters to be transmitted independently from each other, but a vector of them. VQ algorithms are then used to split the N-dimensional vector space to be quantized into L clusters and to choose the best representative codeword for each cluster. The codeword of this cluster is then transmitted. The receiver will take this codeword as a reproduction value of the original input vector X. Obviously, there will be an error between the original input vector and the reconstructed one, but, if the number of levels of the quantizer is controllable, and if it has been designed considering some distortion constraints, this error will be so small that it fulfills our needs.

In order to achieve much less error, the codewords of the vector quantizer must be chosen perfectly. Vector Quantization algorithm may be based either on a known probabilistic distribution model of the data, or on a large training sequence of representative data. In most cases, no probabilistic model can be assumed because of the difficulties that it implies [3] and therefore, the second approach must be used.

The more codewords of VQ the better chance to choose the best representative one for each cluster and therefore the less error will be. This is in the cost of the compression ratio, since the compression ratio is inversely proportional to the number of codewords. The generation of the codewords using the traditional methods is a slow and complex process.

The neural network based systems seem to perform better than traditional vector quantizations in image data compression applications. A model for image data compression using neural networks in the sense of vector quantization was proposed in [4] ,[5] . They have used the self-organizing map proposed by T. Kohonen [6] as a vector quantizer in image data compression. They have trained the network on a set of patterns chosen from the operational service images. The main disadvantage of these networks is that, the performance of the system degrades for images that are not statistically consistent with those used during training [7] .

For this reason, a new methodology for selecting the training data set is proposed. the proposed training set is chosen from a limited number of patterns, mainly, consistent with any image such that, the Kohonen self-organizing map can perform well for all images independent of their statistical properties. The training set consists of 2^N equiprobable disjoint subimages which is the same number as that of the available gray levels, where N is the number of bits used during transmission. The size of each subimage is chosen to be MxM, where M can take the value 2, 4, or 8. All MxM pixels of each subimage have the same gray levels, therefore there are 2^N blocks each carry only one gray level from the 2^N gray levels set.

Using the proposed training set in the training of the network, the network responses much better than other methods for the images which are not used during the training process in terms of the compression ratio.

A hybrid system, which is composed of both the vector quantization and the adaptive DPCM systems, is also used for much higher compression ratios.

By applying the adaptive DPCM with the proposed predictor in [8] on the codewords for the vector quantizer when the Kohonen self-organizing map is used with the proposed training set, the compression ratio of the overall network is the product of the compression ratios of both of them.

2. VECTOR QUANTIZATION USING KOHONEN SELF-ORGANIZING MAP

The Kohonen self-organizing map is a feedforward neural network with two layers as shown in Fig.(1). The first is the input layer which is fed by a set of subimages. Therefore, this layer consists of a number of neurons equal to the number of pixels in each subimage. The second layer is the best matching-score layer which activate the winner of all patterns. The inputs to this layer are received from the outputs of the input layer directly. In the network proposed in [5], the image is divided into blocks of a fixed size nxn. Then, these subimages are used during training of the network. When the network is fed repeatedly by the input patterns, a spatial ordering in the weight vectors takes place corresponding to the input pattern. The spatial extent of this ordering process is usually limited to a finite number of surrounding nodes to the winner node [9].

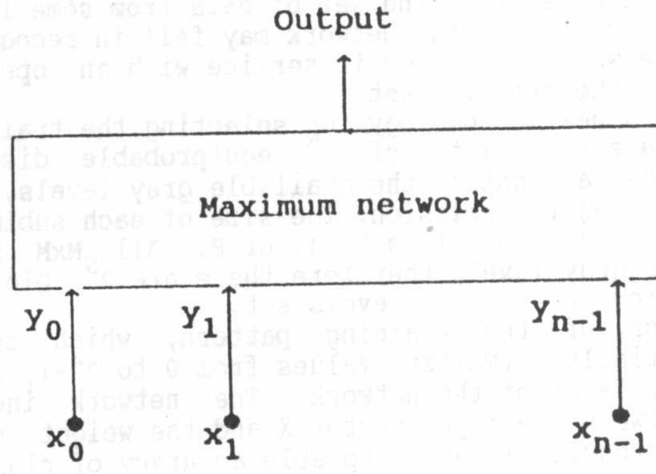


Fig.(1). The self-organizing map

At each learning step, all the neighbors to the winner node are updated such that

$$w_{k+1}(i) = \begin{cases} w_k(i) + \alpha_k \, (X_k(i) - w_k(i)) & i \in N_k(c) \\ w_k(i) & i \notin N_k(c) \end{cases} \qquad (1)$$

where $w_{k=1}(i)$ and $w_k(i)$ are the next and the current weight values respectively, $x_k(i)$ is the current input value, and α_k is the adaptation factor.

The neighbors are set initially to cover the whole network then, it shrinks with the time. The α_k is also chosen to be a monotonically decreasing function of time.

A fine tuning process is then required to improve the classification accuracy of the network. When the learning is completed, the weight vectors form the centroids of the partitioning of the input vector spade [9]. When encoding an image, the codewords of the weight vector corresponding to the best matching vectors are stored. The more nodes in the second layer the more codewords of the network and the better classification performance is obtained, but the less compression ratio will be. The main disadvantage of this network is that it is greatly oriented to the images used during training, such that the classification performance of the system degrades for images that are not statistically consistent with those used during training.

3. THE SELF-ORGANIZING MAP WITH A SPECIAL TRAINING SET

A special training set applied on the Kohonen self-organizing map is proposed to overcome the problem of orientating the weight vector of the network to the images which were used during training while misclassifying those which have not been used in the training process.

3.1 Training the Map With a Special Constructed Set

The training phase is critical in any neural network design because the training is largely influenced by the input vectors, the order of their selection, the rate of decrease of the adaptation parameter α_k and neighborhood size, and the topological formation of the output nodes [5]. As indicated above, the selected training set of data from some images makes the network orient to them such that the network may fail in recognition with an acceptable performance when it is used in service with an operational images which do not belong to the training set.

For this reason, a new methodology for selecting the training data set is proposed. The training set consists of 2^N equiprobable disjoint subimages which is the same number as that of the available gray levels, where N is the number of bits used during transmission. The size of each subimage is chosen to be MxM, where M can take the values 2, 4, or 8. All MxM pixels of each subimage have the same gray level, therefore there are 2^N blocks each carry only one gray level from the 2^N gray levels set.

The input vector of the training pattern, which consists of MxM components with essentially normalized values from 0 to 2^N-1, is presented at the input of the first layer of the network. The network incorporating the Euclidean distance between the input vector X and the weight vector W as a measure of similarity results in an acceptable accuracy of classification as:

$$\| X - W(c) \| = \min_i (\| X - W(i) \|) \qquad (2)$$

where w(c) is the weight of the winner and w(i) is all the rest weights of the map.

The input vector is used to organize the corresponding feature map. The weights are then adapted according to equation (1). Another input vector is then presented at the input of the network and the process is then repeated. The selection of the vector X from the training patterns takes place randomly until all 2^N patterns are presented to the network then, the same set is used again and the process continues. The training continues until all weight vectors of the feature map converge to their final values and the training patterns are correctly separated. The set of weight vectors is then approximates the probability density function of the input vectors as a result of using the adaptation formula in equation (1) [6].

3.2 Fine Tuning of the Network

Initialization of weights with values chosen from an actual training set can also be done [6]. There can be a problem if some weights are initialized with values which are not representative of the training data. In this case, some nodes never win a competition and therefore are useless. In the fine tuning process, the learning rate α must start with a small value, say 0.02, and decrease with time until it reaches zero by the end of the fine tuning process through a number of steps say 100,000 steps [4]. It can be seen from experimental results that, actual fine-tuning learning takes place in less than 1000 samples [9]. Further training will result in smoother textures.

3.3 Actual Image Classification

Once the network is trained, the weight vector will be the desired codewords of vector quantization and the position of each of them in the map has to be encoded. During the encoding process, an input vector representing the pattern is entered to the network and the weight vector which corresponds to this input pattern is then stored. Each output corresponding to each input will be known to the transmitter as well as the receiver and will be stored as fixed search-table. Upon this encoding process, the network is then ready for operational service. Each of the actual images is divided into subimages of the same size as those of the training patterns. These subimages are fed sequentially into the network.

The code which represents the number of the best output node which consequently represents the input vector X, is then transmitted instead of transmitting the vector X. At the receiving end, the decoder has a table of weight vectors W, each defines its respective equiprobable subset. The decoder then emits the vector W as a substitute for the original vector X. The error made by the system is the distance between the vector X and the subset representation vector W.

4. THE ADVANTAGES OF THE PROPOSED TRAINING SET.

The proposed training set for Kohonen self-organizing map as a vector quantizer bandwidth compression system has a number of good characteristics. First, the subsets are equiprobable and thus adjusted to the statistics of the X vectors. This fact makes the transmitted codewords equiprobable, which is a property of any good code. Second, they can be applied on virtually any type of data that is normalized between zero and one. Third, vector quantizer makes the quality of the image tend to degrade gracefully as the number of bits, N, is reduced. This offers the possibility of building systems with high compression ratios in situations where transition fidelity can be sacrificed. The simulated results will show the performance improvement of the map using the proposed training set.

5. PREDICTIVE VECTOR QUANTIZATION

The transmitted codewords can be further compressed. Since these codewords are correlated with each other, then using any of the waveform compression techniques, for example the adaptive differential pulse code modulation (DPCM), will satisfy these requirements. Instead of sending these codewords directly, they can be entered as inputs to the adaptive DPCM system using the linear neural network predictor as in [8] with only one bit per pixel achieving a very high compression ratio with an acceptable SNR. Then, the output of the adaptive DPCM system is sent through the communication channel.

6. COMPUTER SIMULATION AND RESULTS.

The Kohonen self-organizing map, like any other neural network operates in parallel. But since there is no available hardware chip, the neural network structure can be simulated on a sequential computer and the proposed training algorithm can be applied on it. The adaptation factor α_k in equation (1) is set to be 0.5 at the beginning of the training process [9], whereas it starts with the value 0.002, in the beginning of the fine tuning process. It has been chosen to decrease linearly with time till it reaches zero by the end of the fine tuning process according to the formula:

$$\alpha_k = 0.5 \left[1 - \frac{k}{T} \right] \qquad (3)$$

where T is the total number of training steps. The values of the weight vector are initialized to a small different random values. The network will not behave well if the initial weight vector values are not different [6] . Each subimage of the proposed training set is applied at the input of the map. The network selects the winner node of the output, Then, the adaptation of the network according to equation (1) takes place. Another subimage is then fed into the input of the network and the process continues till the separation of the weights takes place.

The adaptation process of the neighbors of the winner node has been chosen to be applied at the beginning on the whole network and then shrink linearly with time. It is even possible to end the process with adapting the best winner node only. The images of size 512x512 with 256 gray levels are used. But, since we have a restricted number of gray levels which depend on the display adapter (in our case, it was a VGA card with 64 gray levels), the images are quantized into only 64 gray levels. The blocks of sizes 2x2, 4x4, and 8x8 representing the actual subimages were used each in a separate run. Upon stopping the adaptation process, each subimage is presented at the input of the network and the network will choose the winner of all nodes and send its codeword to the receiver which searches for the corresponding subimage to this codeword in its table.

Fig.(2) to Fig.(4) show the response of the Kohonen self-organizing map with the proposed training set when it is applied on a wide range of images. these images were chosen perfectly to be high-contrast images carrying much details and sharp edges. The performance of the proposed predictive vector quantization system is also shown. The corresponding compression ratios and signal to noise ratios are indicated on these figures.

Fig.(2-a) shows the original Lenna picture. This picture caries much details and a wide range of gray levels. Fig.(2-b) shows the reconstructed image using the Kohonen self-organizing map with the proposed training set when the size of the subimage is 2x2. This reconstructed image achieves a compression ratio of 4, and the SNR is 22.2 dB.

Note that, the performance of the network, when it uses the training set extracted from actual images, is better than that of the network when it uses the proposed training set if the reconstructed image was one of that used during training. The SNR of the reconstructed image of Lenna picture when the original picture is used during training is 25.6 dB [5] . Whereas, the network performance degrades greatly if this image was not one of those used in the training process, unless the picture has the same characteristics as the pictures used in the training process. When this image is statistically independent of those used during training, a loss of up to 10 dB in the SNR for these images is reported [10] . This will not occur if the network uses the proposed training set, since the network will treat all images almost the same. This is clear in the next reconstructed images. Fig.(2.c) shows the reconstructed image using adaptive DPCM, when the predictor proposed in [8] with one bit per pixel applied on the image in Fig.(2.b), achieving compression ratio of 32 and the SNR = 20.28 dB. The error between the final reconstructed image in Fig.(2.c) and the original image in Fig.(2.a), is shown in Fig.(2.d). Note that, in spite of the high compression ratio achieved in this reconstructed image, both the SNR and the visual appearance of it are considerably acceptable. This appears clearly in the error between the original and the reconstructed image. The error concentrates mainly at edges, but it can be eliminated to a great extent if a suitable low pass filter is used.

Fig.(3) shows the response of the Kohonen self-organizing map when it is applied on larger block sizes. Fig.(3.a) shows the original Lynda picture. This picture contains the face features as well as wide areas of the same gray levels. Fig.(3.b) shows the reconstructed image when the block size is 2x2 with a compression ratio of 4 and the SNR is 25.54 dB. Fig.(3.c) shows the response of the map when it is applied on 4x4 block-size subimages. The compression ratio increased to 16 but the SNR decreased to 19.93 dB. This decrease in SNR is acceptable for the corresponding increase in the compression ratio. The blocking effect appears in this Figure can easily be extracted using a suitable low pass filter. An example of this process is shown in Fig.(3.d), where the averaging low pass filter with a window of size 3x3 is applied on the image shown in Fig.(3.c). The adaptive DPCM using the linear neural network is applied on the images in Fig.(3.b) and Fig.(3.c). Fig.(3.e) and Fig.(3.f) show the reconstructed images with only one bit per pixel with SNR 23.56 dB and 17.53 dB and compression ratios of 32 and 128 respectively.

More compression ratio can be obtained if the block size is increased to 8x8. The compression ratio in this case is 64 but the SNR, produced for the reconstructed Lynda picture using 8x8 block-size subimage shown in Fig.(3.g), is 15.59 dB. Note that, in spite of the acceptable SNR for this compression ratio, the features of this reconstructed image began to disappear and the blocking effect appears clearly. Block sizes of 8x8, or larger, are used only in case of a wide area of the same gray level.

From all the above reconstructed images it has been seen that the visual appearance of the reconstructed images of the self-organization map using the adaptive DPCM system with the technique proposed in [8] as well as the SNR were not greatly affected, while the compression ratios increased by a factor of 8.

Fig.(5) shows the relation between the compression ratio and the SNR when both vector quantizer of Kohonen self-organizing map with the proposed training set is used alone and when the adaptive DPCM with the linear neural network predictor is applied on its output codewords. The relation indicates that, using this predictive vector quantization technique performs much better than using vector quantization only since the adaptive DPCM can extract the correlation between codewords of vector quantization.

7. CONCLUSION

In this paper a proposed training set for Kohonen self-organizing map was introduced. Then, this map with the proposed training set was used in image data compression as a vector quantizer. The reconstructed images when Kohonen self-organizing map with the proposed training set was used, show that, the performance of the network from the signal to noise ratio as well as the visual appearance points of view is much better than that of the network when the training set was extracted from the actual images if the used image was not one of those used during training.

The predictive vector quantizer, which is a hybrid system of both the vector quantization and the adaptive DPCM systems, was achieved. In this system, the adaptive DPCM system with the linear neural network predictor is applied on the codewords of the vector quantizer of Kohonen self-organizing map when the proposed training set is used.

Using this hybrid system, the overall performance was greatly improved over that if either the vector quantization or adaptive DPCM is used individually. The signal to noise ratio improvement in the proposed predictive vector quantization is in the order of 4-6 dB over each of vector quantization and the adaptive DPCM systems. This is a good improvement in the performance of image data compression systems.

REFERENCES

[1] K. Thyagarajan, and H.Sanchez, "Encoding of Videoconferencing Signals Using VDPCM," Signal Processing Image Communication1990, pp. 81-84.

[2] A J. Jain, "Image Data Compression: A review," Proc. IEEE, Vol. 60, No.3, March 1981, pp. 349-389.

[3] A. Izquierdo, j. Sueiro, and J. Mendez, "Self-Organizing Features Map and Their Application to Digital Coding of Information," International Workshop IWANN'91 Canada-Spain Sep. 17-19, 1991.

[4] N. Nasrabadi, Y. Feg, "Vector Quantization of Images Base Upon the Kohonen Self-Organizing Feature Maps," Proc. IEEE International Conf. on Neural networks, ICNN-88 (San Diego, CA, 1988), pp. I-101 - I-108.

[5] K. Thyagarajan, and A. Eghblamoghadam, "Design of a Vector Quantizer Using a Neural Network," Electronics and Communications 1990.

[6] T. Kohonen, "Self-Organizing Map,", Proc. IEEE, Vol. 78, No. 9, Sep. 1990, pp. 1464-1480.

[7] R. Hecht-Nielsen, "Neurocomputing," Hnc, Inc. and University of California, San Diego, 1989.

[8] R M. Atta, "Image Data Compression Using Neural Networks," M.Sc. Thesis, Menoufia University, Egypt, 1992.

[9] D. Erickson and K. Tyagarajan, "A Neural Network Approach to Image Compression," IEEE International Symposium on Circuits and Systems, May 10-13, 1992.

[10] F. Arduini, S. Fioravanti, and D. Giusto, "Adaptive Image Coding Using Multilayer Neural Networks," IEEE International Conf. on ASSP. May 14-17, 1991, pp. II-381 - II384.

Fig.(2.a). The original Lenna picture

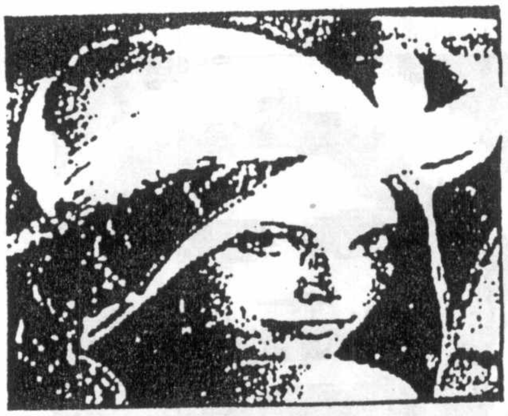
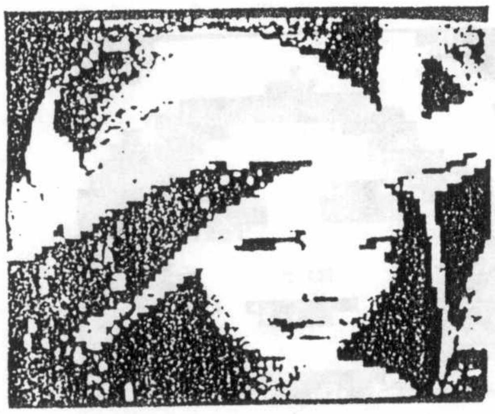


Fig.(2.b). The reconstructed image using 2x2 block-size when the Kohonen self-organizing map with the proposed algorithm is used. The SNR = 22.2 dB.



ig.(2.C). The reconstructed image when applying he adaptive DPCM, using the proposed linear neural etwork predictor with only one bit per pixel, on the image in Fig.(2 .b). The SNR = 20.28 dB.

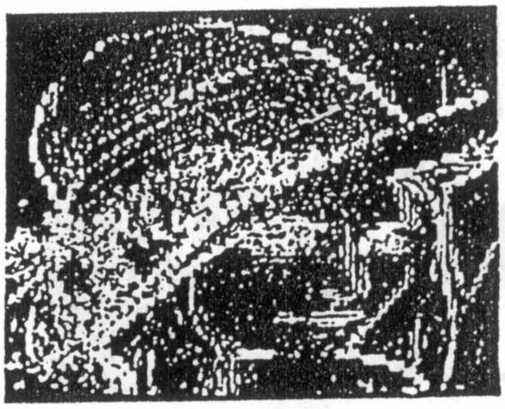


Fig.(2.d). The error between the reconstrucd image in Fig.(2 .c) and the original image Fig.(2 .a).

Fig.(3.a). The original Lynda picture.

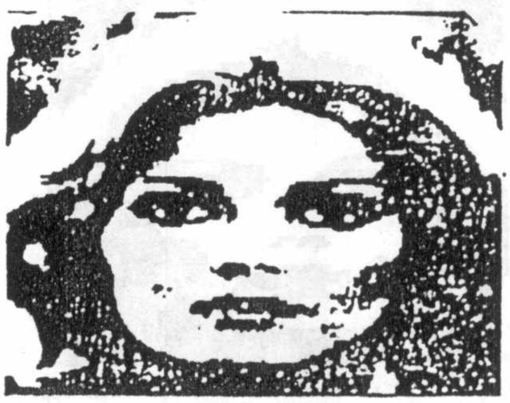


Fig.(3.b). The reconstructed image using 2x2 block-size when the Kohonen self-organizing map with the proposed algorithm is used. The SNR = 25.56 dB.

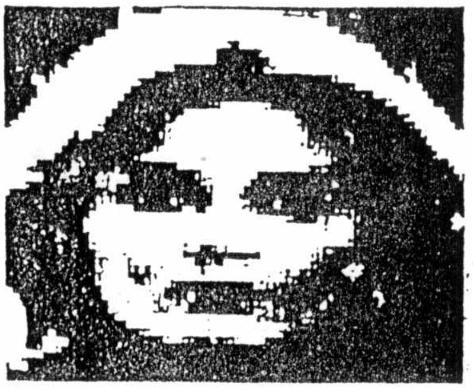


Fig.(3.C). The reconstructed image using 4X4 block-size when the Kohonen self-organizing map with the proposed algorithm is used. The SNR = 19.93 dB.

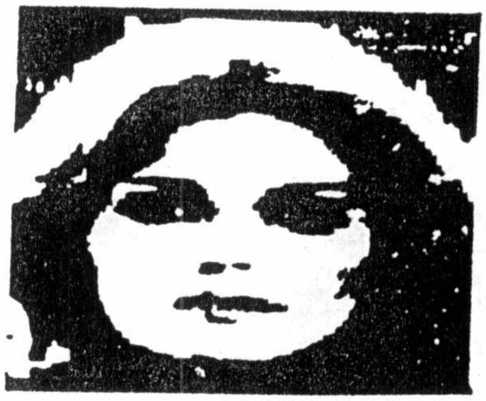


Fig.(3.d). The filtered image using averaging LPF for the reconstructed image in Fig.(3 .c).

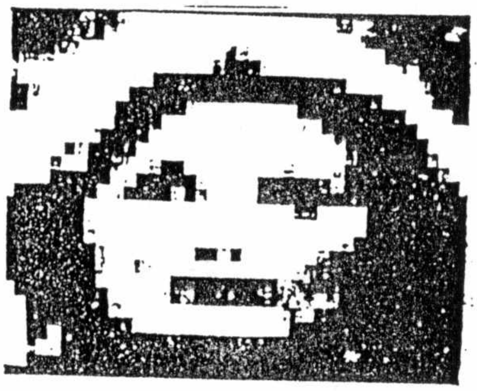


Fig.(3 e). The reconstructed image when applying the adaptive DPCM, using the proposed linear neural network predictor with only one bit per pixel, on the image in Fig.(3 .c). The SNR = 17.53 dB.

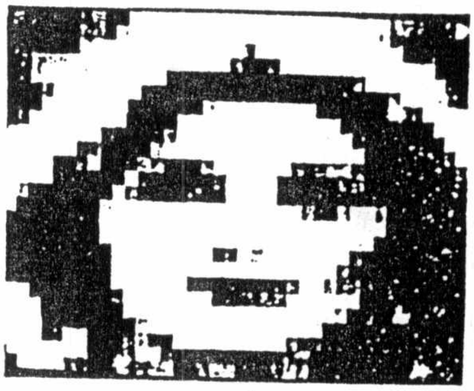


Fig.(3.f). The reconstructed image using 8x8 block-size when the Kohonen self-organizing map with the proposed algorithm is used. The SNR = 15.56 dB.

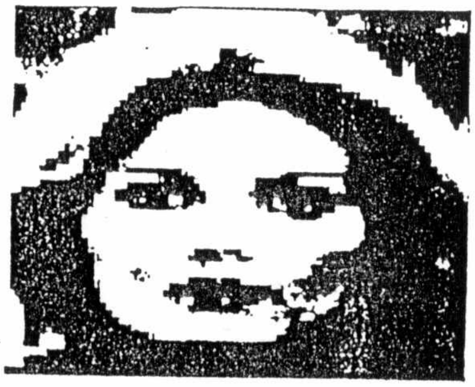


Fig.(3.g). The reconstructed image when applying the adaptive DPCM, using the proposed linear neural network predictor with only one bit per pixel, on the image in Fig.(3 .b). The SNR = 23.56 dB.

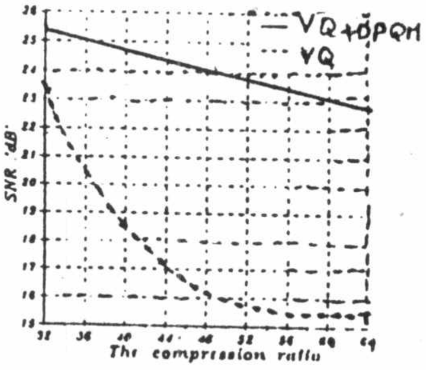


Fig.(4). The relation between the compression ratio and the signal to noise ratio for VQ. and predictive VQ.

ASAT

MILITARY TECHNICAL COLLEGE

CAIRO - EGYPT

PARALLEL IMPLEMENTATION OF RADIX-2 FAST FOURIER TRANSFORM ON TRANSPUTER ARRAY

Dr. Fatma A. Omara* Eng. Mohamed A. Shohla*

ABSTRACT

The Fast Fourier Transform (FFT) is frequently used in various fields of science and engineering such as tomography, speech recognition, image processing, digital signal processing, coding theory, etc. There are many versions of the FFT algorithms which are cononic. The most powerfull versions are based either on the Cooley-Tukey algorithm or on the Sande-Tukey algorithm. Also, the Radix-2 is most common in the applications of the FFT algorithms. Therefore, the parallel implementation of the FFT is considered very important for minimizing the execution time. This implementation should be on a parallel architecture. This paper describs a parallel implementation of the Radix-2 FFT using a transputer array as the underlying parallel architecture and the developed code is written in OCCAM language where the partitioned algorithm is used in this implementation. The Radix-2 FFT is implemented using three methods. These are; the single track method, the double track method and the overlapping method. Results for a variety of different number of data samples and number of processes are included in this paper. Comparisons are made among the three methods with the sequential one from the speed up points of view.

1- INTRODUCTION

High performance computers are increasingly in demand in many areas of computer applications such as engineering design and automation, energy resources, medical fileds, artificial intelligence, remote sensing, military and basic research areas [1],[2]. For higher throughput and ability to handle very large amount of data, the parallel processing should be used. However, there are three fundamental techniques to express parallelism. These are; pipelined algorithm, partitioned algorihm and relaxed algorithm [2]. The choice of any of these techniques depends on the features and characteristics of the applications, as well as on the type of the underlying parallel architecture. More details about parallel architectures and parallel processing can be found in [1], [3], [4] and [5].
On the other hand, the transputer based systems are considered to be MIMD (Multiple Instruction Multiple Data stream) machine. The transputer [6] is a single-chip microprocessor (we can say, it is a microcomputer). It has a processor, a memory and four links to connect one transputer to others, all in a single VLSI chip. Many versions of transputers have been produced, such as T212 16-bit processor, T414 32-bit processor and finally T800 32-bit

* Dept. of Computer Science & Eng., Faculty of Electronic Eng., Menoufia University, Menouf

processor with 64-bit floating-point processor as shown in Fig. (1). The T800 transputer is the avaiable type in our faculty and the parallel implementation of the FFT has been devloped using it. A transputer array can be easily constructed by identical transputers in fixed or reconfigurable topologies [6]. On the other hand, the OCCAM language reflects the way in which transputers are constructed to be multiprocessor systems [6]. Programming in OCCAM [7] enables an application to be described as a collection of processes, where such processes can be executed concurrently, and communicate with each other through soft or hard channels. More detials about transputer and it's OCCAM language are found in [8], [9], [10] and [11]. On the other hand, the Fourier transform is used in many fields of science and engineering. The Discrete Fourier Transform (DFT) is easy to implement on digital computers and has wide applications in the areas of speech transmission, coding theory, image processing and digital signal processing [12]. The Fast Fourier Transform (FFT) is a computer algorithm which has been devloped to compute the DFT using minimum number of operations, then it reduces the computing time. The Radix-2 FFT goes back to Runge and Koeing in 1924 and Stumpff in 1937 but nobody has noticed that because of the way they were described. In 1965 Cooley and Tukey published the first definitive paper in this subject, which was titled as "An algorithm for machine calculation of complex fourier series". The work in this paper presents an implementation of Radix-2 FFT on $N=2^n$ data points using $P=2^m$ processes executed in parallel on T800 transputer. The Radix-2 FFT is implemented using three methods. These methods are; the single track method, the double track method and the overlapping method.

2- THE FAST FOURIER TRANSFORM

The Discrete Fourier Transform (DFT) can be expressed as

$$X(n) = \sum_{k=0}^{N-1} x(k)\, W^n \qquad ; n=0,1,\ldots..,N-1 \qquad (1)$$

Where N= Number of complex data points, and

W= exp (-j2π/N).

The direct computation of this equation requires N^2 complex multiplications and N(N-1) complex additions. When N is a power of 2, there is more efficient method known as the Radix-2 Fast Fourier Transform (FFT) that can be used instead. The Radix-2 FFT requires only $(N/2)LOG_2(N/2)$ complex multiplications and $(N/2)LOG_2N$ complex additions. The computation of Radix-2 FFT is broken into butterflies [13]. When $N=2^n$, the computational of Radix-2 FFT consists of n stages or computation levels with each stage consists of N/2 butterflies. Each butterfly consists of two data points. To compute the butterfly; the weighting factor W^n is generated first and then multiplied by the two data points to get the butterfly. Thus, each butterfly consists of one multiplication, one addition and one subtraction. There are many versions of the FFT algorithms which are canonic. The most powerful algorithms are the Cooley-Tukey algorithm and the Sande-Tukey algorithm [12]. Each algorithm can be implemented in two modes; in the first mode the initial data is in natural order (so the output data is in reversed-bit order) and in the second mode the initial data is in bit-reversed order (so the output data is in natural order).

3- HARDWARE FOR PARALLEL IMPLEMENTATION OF THE FFT

The hypercube topology is considered the best known topology for the parallel implementation of the FFT [14]. On the other hand, the transputer provides four communication links to construct the network. These links provide communication with up to four transputers which make it easy to develope a parallel architecture with any number of transputers. In this work, the transputer array constructed as a hypercube network is used to implement the FFT algorithm in parallel. This hypercube network for the FFT implementation is proposed by Khan and it's known as Khan's FFT engine [14]. The Khan's FFT engine for various dimensions is shown in Fig. (2). Because only one T800 transputer is available for the authors, we have implemented the parallel Radix-2 FFT with the parallelism on the processes level not on processors level. Because of the structure of the OCCAM language, the implementation on transputer array can be done by using the same code of the implementation on a signle transputer but with some changes only in the mapping declarations.

4- SERIAL IMPLEMENTATION OF THE RADIX-2 FFT

The Radix-2 FFT using the Cooley-Tukey and the Sande-Tukey alogrithms has been implemented in serial. Each algorithm can be implemented in two modes; one mode with the initial data is in natural order (so the output data is in reversed-bit order) and the second mode with the initial data is in bit-reversed order (so the output data is in natural order). The implementation of the two algorithms are the same, the only difference between the two algorithms is in the computation of the butterflies. In the case of Cooley-Tukey butterfly, the weights are multiplied before the application of 2-point DFT, while the weights are multiplied after the application of 2-point DFT in the case of Sande-Tukey butterfly. The realtion between the two butterflies is described in equation (2).

$$X0 = x_0 + x_1 W^k \qquad X0 = (x_0 + x_1) W^k$$
$$X1 = x_0 - x_1 W^k \qquad X1 = (x_0 - x_1) W^k \qquad (2)$$

(a)Cooley-Tukey butterfly (b) Sande-Tukey butterfly

Comparison between these two algorithms has been made, it clears that the computation time of the Cooley-Tukey algorithm is faster than that of the Sande-Tukey algorithm because of the recursive generation of the weights. The comparision between these algorithms is shown in Fig. (3).

5- PARALLEL IMPLEMENTATION OF RADIX-2 FFT

The parallel implementation of Radix-2 FFT depends upon the distribution of data among the processes or the sequence of computations. There are three methods have been used to implement the Radix-2 FFT in parallel. These methods are; the single track method, the double track method and the overlapping method.

5.1- The Single Track Method

In this method, the vector of data x of length $N=2^n$ (n is an integer) is distributed among $P=2^m$ (m is an integer) processes. By using consecutive storage technique, the first N/P data points are split into the first process and the second N/P data points are split into the sceond process and so on. The mapping of data vector of length N=16 (when the input data is in natural order or in bit-reversed) among P=4 processes is described bellow in table 1.

Table 1 Mapping of 16 data points among P=4 processes in the single track method.

PROCESSES			
P0	P1	P2	P3
0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	15

a) Mapping of natural order data

PROCESSES			
P0	P1	P2	P3
0	2	1	3
8	10	9	11
4	6	5	7
12	14	13	15

B) Mapping of bit-reversed data

When the FFT of $N=2^n$ data points is computed on $P=2^m$ processes (where n> m), the algorithm consists of n stages with each stage consists of N/2 butterflies. For the n stages, there are m stages with each butterfly in these stages is distributed among two processes where communication between them is needed to compute this butterfly. Therefore, these m stages are called distributed stages because they need communication between the processes. The rest (n-m) stages are called local stages because the butterfly in each stage is held by only one process. Hence, each process can compute part of the rest (n-m) stages without communication with any other processes. The single track method has many disadvantages. The first one, an extra buffer of the same length of data vector x is needed for the communication between processes in the distributed stages. The second disadvantage is the imbalance of workload between the processes which computes the butterflies. The last disadvantage is that each process in the network is in either communication mode or computation mode. Because the capability of the transputer to communicate and compute simultaneously, it can possible to make use this capability in improving the implementation. This will be explained later. The comparision between the Cooley-Tukey and Sande-Tukey algorithms for P=4 processes is shown in Fig. (4).

5.2- The Double Track Method

According to this method, the data vector x of length N is divided into two equal halves to form subvectors x1 and x2. The first subvector x1 is distributed among the $P=2^m$ processes as in the single track method. Similarly, the second subvector x2 is distributed among the P processes. The mapping of natural order and bit-reversed data for N=16 is described in table 2. As in the single track method, there are n stages with each stage contains N/2 butterflies. These n stages consists of m distributed stages and (n-m) local stages. In the distributed stages when the data is in natural order, each process computes, in parallel, it's porition of distributed butterflies and then exchanges it's lower or upper half data with the appropriate process. Since, the communication occurs after (or before) the computation depending upon the order of the data being computed. Therefore, there is no need for extra buffering as in the case of single track method. The load balance between the processes has been improved because each process has to compute it's own porition of distributed butterflies. The computation time of the double track method is less than that of the single track method because there is no need for extra buffer for communication and the better load balance between the processes. The comparision between the Cooley-Tukey and Sande-Tukey algorithms is shown in Fig. (5).

Table 2 Mapping of 16 data points among P=4 processes in the double track method.

PROCESSES			
P0	P1	P2	P3
0	2	4	6
1	3	5	7
8	10	12	14
9	11	13	15

a) Mapping of natural order data

PROCESSES			
P0	P1	P2	P3
0	2	1	3
8	10	9	11
4	6	5	7
12	14	13	15

B) Mapping of bit-reversed data

5.3- The Overlapping Method

The overlapping method depends on the overlap of the computation and the communication in each process. According to this method, the data are distributed among P processes as in the case of the double track method. Then, the data in each process preprocessed such that it is divided into 2^H (where H is an integer) data hops. As an example, the data preprocessed such that it is divided into two data hops (H=1) for N=32 on P=4 processes is shown in table 3. During the distributed stages, all processes computes it's porition of butterflies for the 0th data hop. Then, when each process computes the butterflies for 1st data hop, it also exchanges the data of 0th data hop in the same time and so on. At any time, each process computes the FFT of the (i+1) data hop and exchanges the data of i hop simultaneously. At last, each process exchanges the data of the (2^H-1) data hop. At this point, we must note that the transputer is the only processor which has gotten the possibility for implementing computations and communications simultaneously. Accordingly the overlapping method can be implemented on transputer. The comparision between the Cooley-Tukey and Sande-Tukey algorithms is shown in Fig. (6).

Table 3 Preprocessed data for N=32, H=1 and P=4 processes.

PROCESSES

P0	P1	P2	P3
0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	15
16	20	24	28
17	21	25	29
18	22	26	30
19	23	27	31

Initial data

P0		P1		P2		P3	
1-hop	2-hop	1-hop	2-hop	1-hop	2-hop	1-hop	2-hop
0	2	4	6	8	10	12	14
1	3	5	7	9	11	13	15
16	18	20	22	24	26	28	30
17	19	21	23	25	27	29	31

Data after preprocessed

6- THE RESULTS

From our experimintal results of the execution time, it has been noticed that the double track method is better than the single track method because it overcomes the load imbalance between the processes and does not need extra buffer for communication which has happened in the single track method. Also, the execution time of the overlapping method is shorter relative to the double track method because it overlapes the computation and the communication operations, while in the single track method and the double track method each process could be in only computation mode or in only communication mode. On the other hand, The best execution time in the overlapping method is obtained when the number of data hops is minimum. This is due to the communication latency. The comparision of the serial implementation of the Radix-2 FFT and the three parallel algorithms is dipected in Fig. (7) and Fig. (8).

7- CONCLUSIONS

By using the transputer and it's OCCAM language, we can easily construct a parallel architecture from a collection of transputers which operate concurrently and communicate through their links. This paper describes the parallel implementation of the Radix-2 FFT on the transputer array. The T800 transputer array is used simulated on one transputer (which is available) as the underlying parallel architecture. On the other hand, the Radix-2 FFT has been implemented using three methods. These methods are; the single track method, the double track method and the overlapping method. Comparisons are made among these three methods and the serial algorithm. It can be concluded from our comparision that the overlapping method is much better from the execution time point of view relative to the other methods. Also, it has been made clear that the overlapping method can be implemented only on transputers. Finally, we can state that using the transputers as the underlying parallel architecture is considered to be the best one for implementing the FFT algorithms in parallel.

8- REFERENCES

[1] Hwang K. and Briggs F. A., "Computer Architecture and Parallel Processing", Mc Graw-Hill, 1985.
[2] Quinn M. J., "Designing Efficient Algorithms For Parallel Computers", Mc Graw-Hill, 1987.
[3] Hockney R. W. and Jesshope C. R., "Parallel Computer 2, Architecture, Programming and Algorithms", Adam Hilger, 1988.
[4] Anderson A. J. ,"Multiple Processing, A system Overview", Prentice Hall, 1989.
[5] Desrochers G. R., "Principles of Parallel and Multiprocessing", McGraw-Hill, 1988.
[6] Jesshope C. R., "Transputer and Switches as Objects in OCCAM", Proc. Of the Parallel Computing 8, North-Holland,pp. 19-30, 1988.
[7] Omara F. A., "Implementation of Prolog Using Massively Parallel Processes", ph. D, Menoufia University, 1989.
[8] Ghonaimy M. A. R. , Fatma A. Omara, "Transputer & OCCAM", Proc. Of the 16th International Conference for statistics, Computer Science, Social and Demographic Research, Cairo, Egypt, 1991.
[9] INMOS Limited, "Transputer Reference Manual", PrinticHall, 1988.
[10] INMOS Limited, "OCCAM 2 Reference Manual", Printic Hall, 1988.
[11] Jones G. and Goldsmith M., "Programming in OCCAM 2", Prentice Hall, 1988.
[12] Brigham E. O., "The Fast Fourier Transform", New Jersey, Prentice Hall, 1974.
[13] Wold E. H. And Despain A. M., "Pipeline and Parallel Pipeline FFT Processors for VLSI Implementations", IEEE Trans. on Computers, Vol. c-33, No. 5, May 1984.
[14] Khan A. and Stepphans N.,"Fast Fourier Transform on Transputers", J. Edwards Ed., IOS press, 1991.

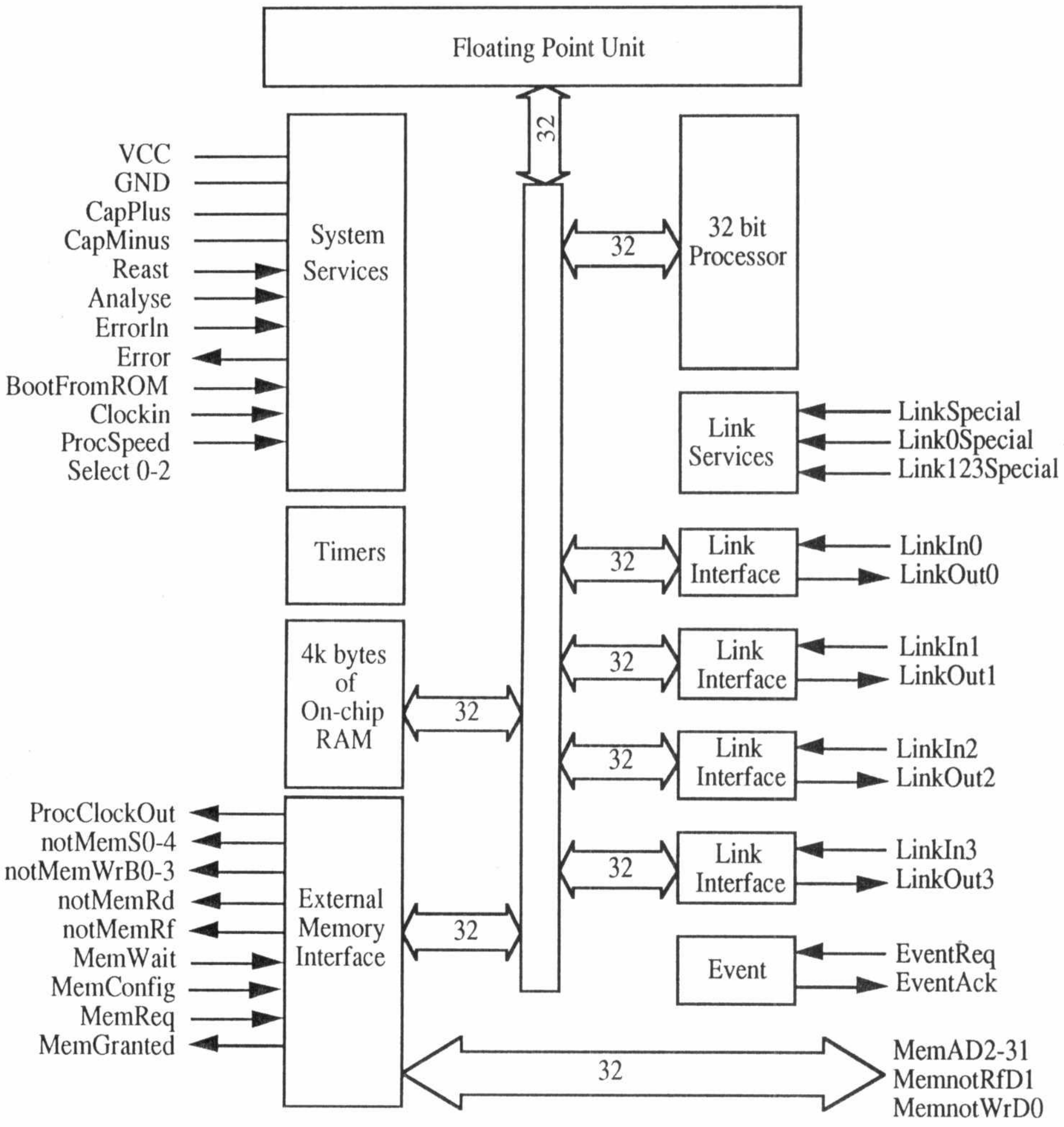
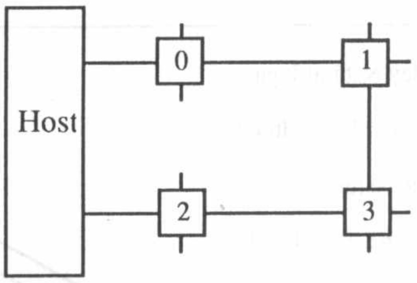
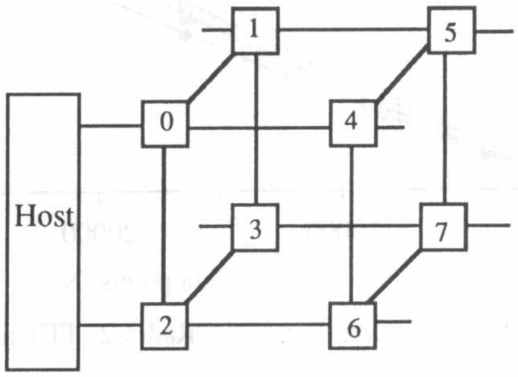


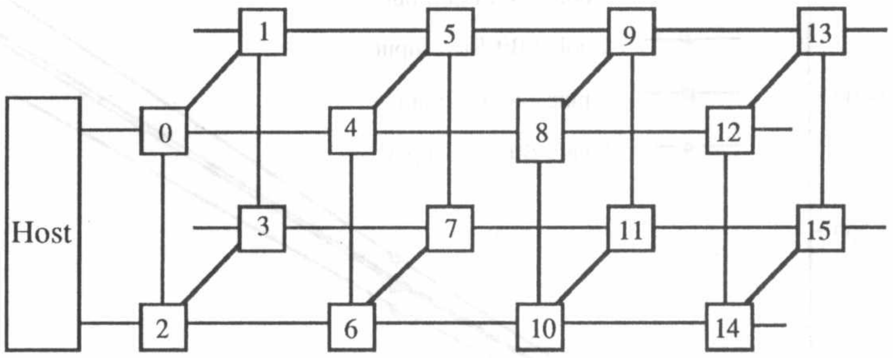
Fig. (1) IMS T800 block diagram



a) 4 transputers (2-dimensional) Khan's FFT engine connected with the host.



b) 8 transputers (3-dimensional) Khan's FFT engine connected with the host.



c) 16 transputers (4-dimensional) Khan's FFT engine connected with the host.

Fig. (2) Khan's engines FFT of various dimensions connected with the host.

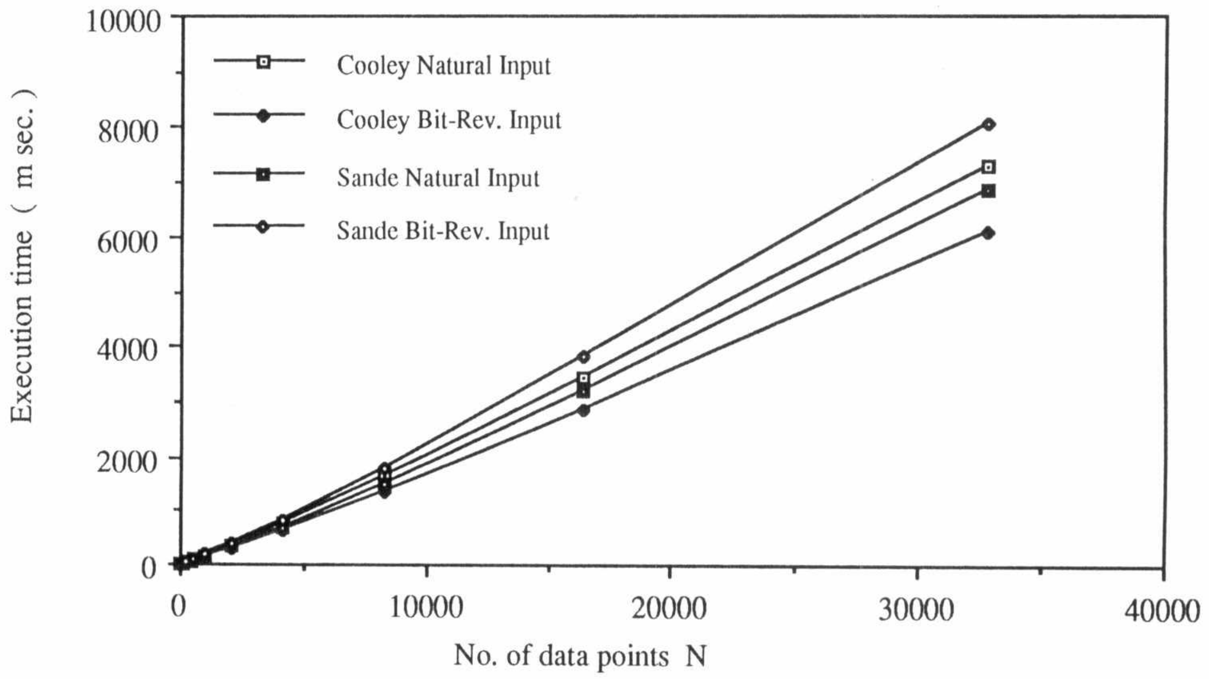


Fig. (3) The execution time of the Radix-2 FFT using four serial algorithms.

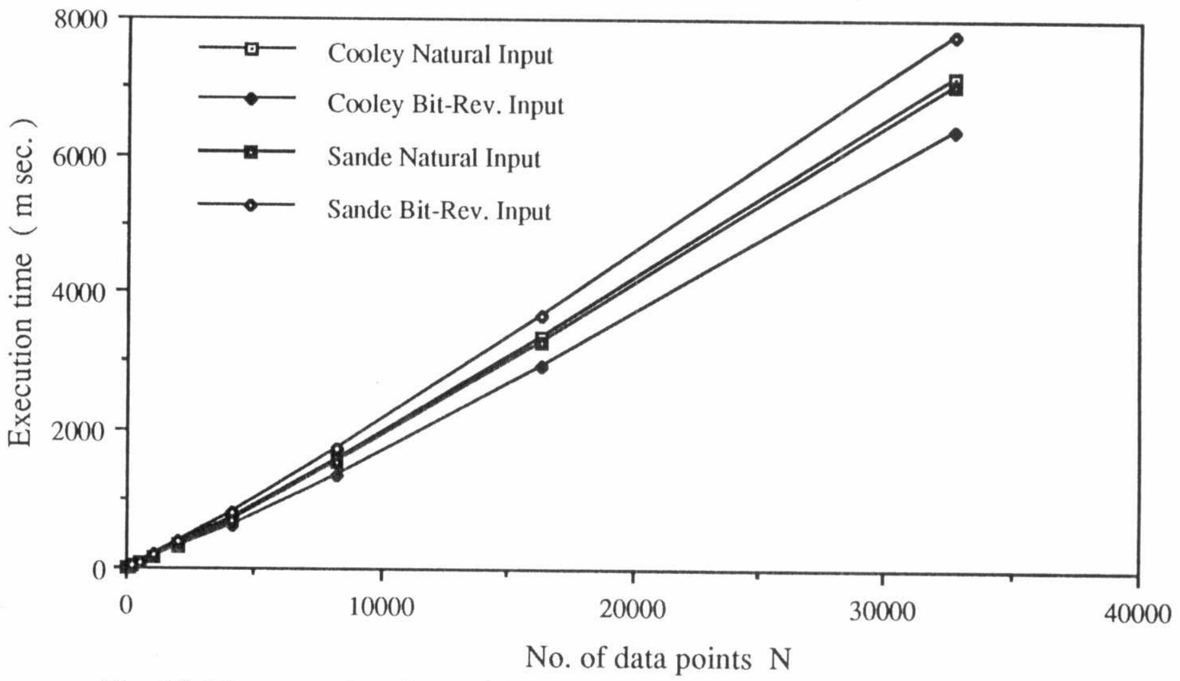


Fig. (4) The execution time of the Single Track method of the Radix-2 FFT using four algorithms and P = 4 processes.

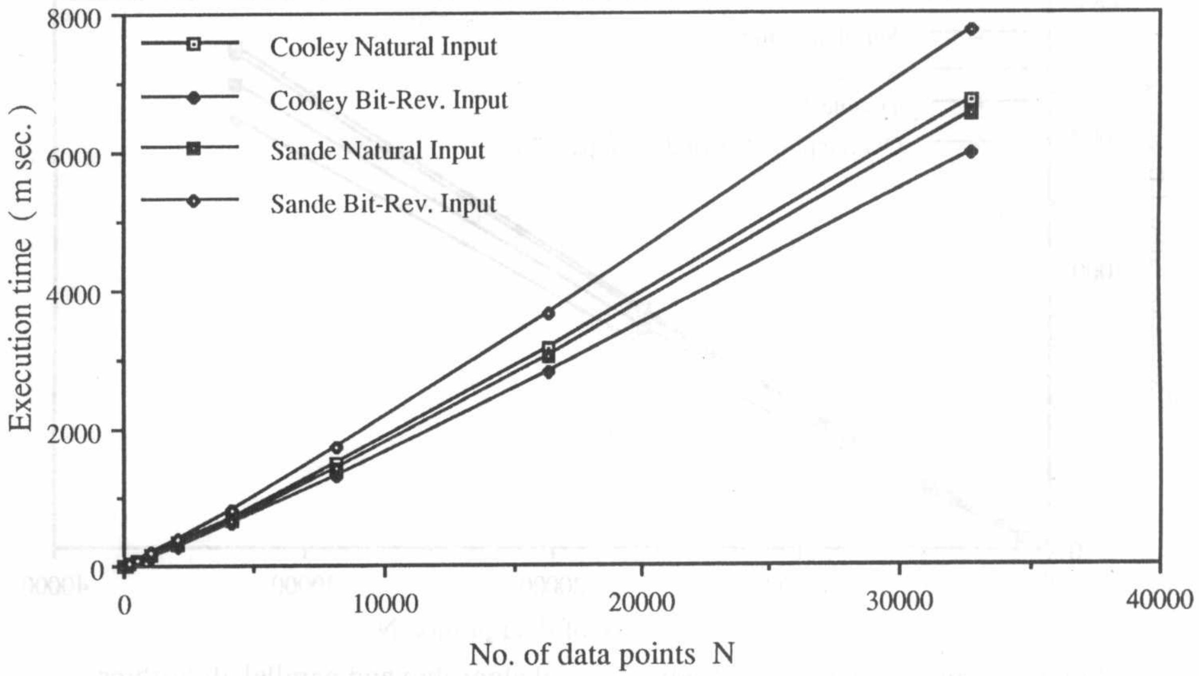


Fig. (5) The execution time of the Double Track method of the Radix-2 FFT using four algorithms and P = 4 processes.

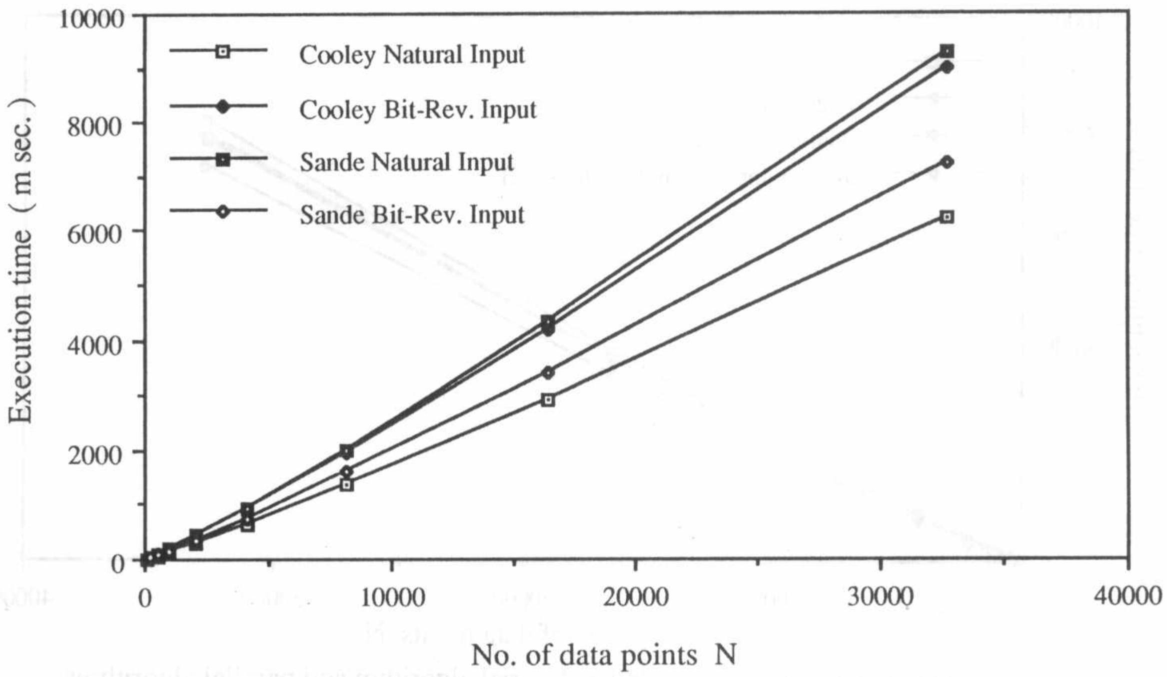


Fig. (6) The execution time of the Overlapping method of the Radix-2 FFT using four algorithms and P = 4 processes and number of data Hops = 2.

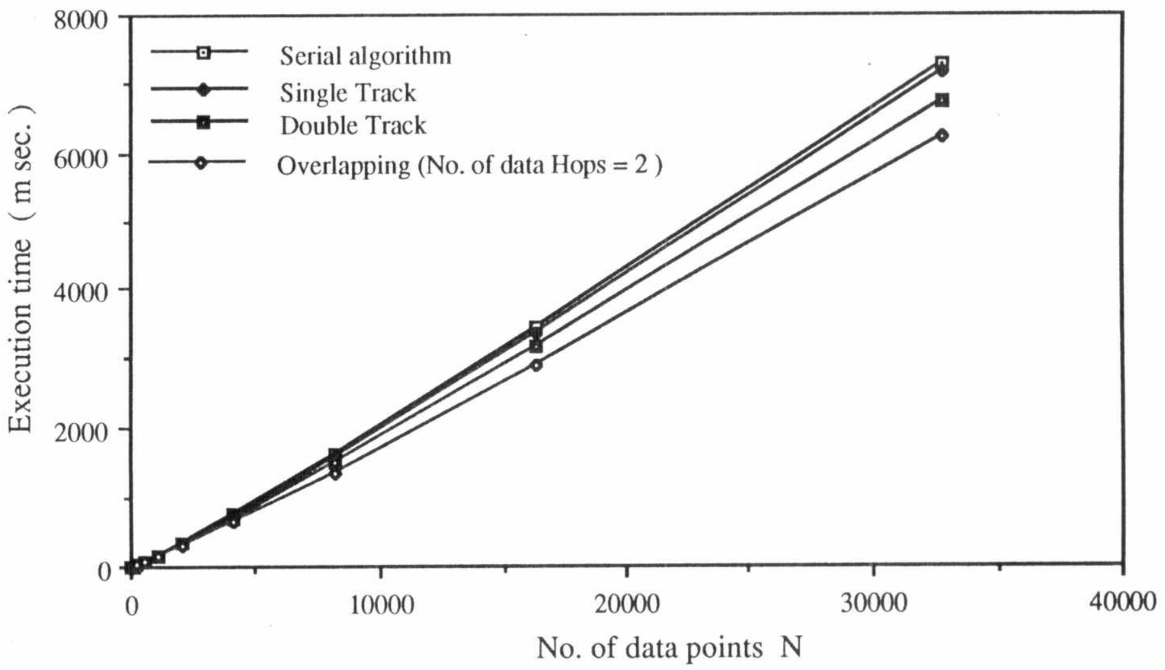


Fig. (7) Comparsion between Radix-2 serial algorithm and parallel algorithms (for p = 4 processes) for Cooley-Tukey (Natural Order Inputs) technique.

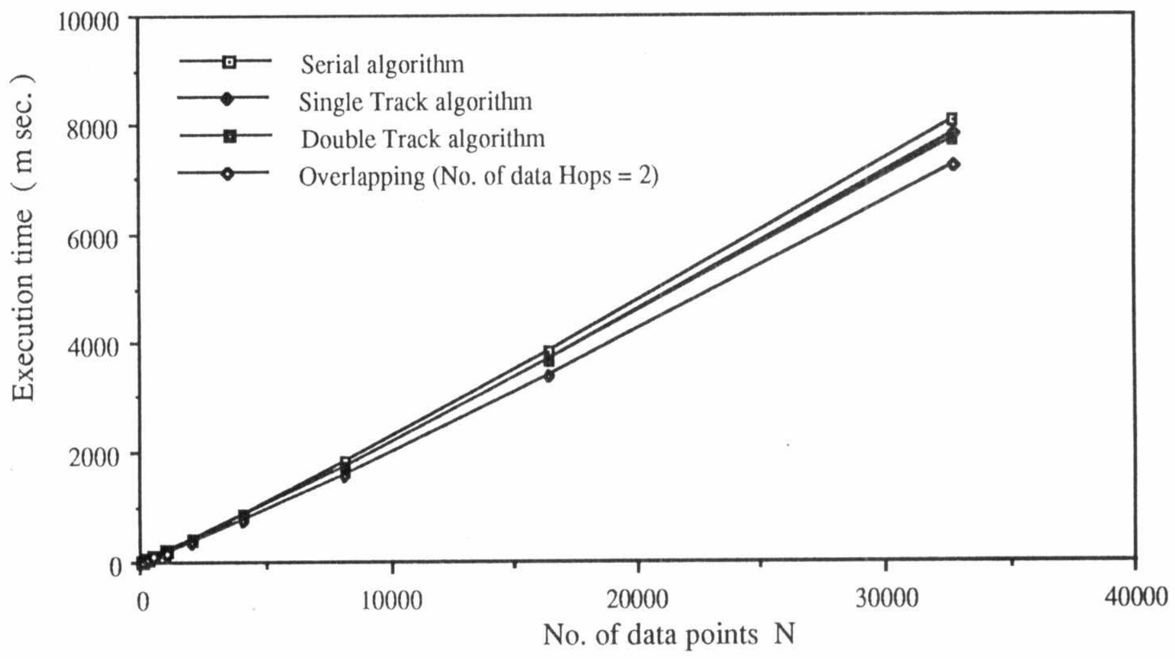


Fig. (8) Comparsion between Radix-2 serial algorithm and parallel algorithms (for p = 4 processes) for Sande-Tukey (Bit-Reversed Inputs) technique.