A 2D-FFT ALGORITHM ON MESH CONNECTED MULTIPROCESSOR SYSTEMS

Hiroaki Kunieda and Kazuhito Itoh

Department of Electrical and Electronic Engineering
Tokyo Institute of Technology
Tokyo 152, Japan

Abstract: A direct computation algorithm of two dimensional fast Fourier transform (2D-FFT) is considered here for implementation in mesh connected multiprocessor array of both a 2D-toroidal and a rectangular type. Results are derived for a hardware algorithm including data allocation and interprocessor communications.

A performance comparison is carried out between the proposed direct 2D-FFT computation and the conventional one to show that a new algorithm gives higher speedup under a reasonable assumption on the speeds of operations.

1. Introduction

2-D FFT is used in most image processing devices, in applications such as pattern recognition, image reconstruction and correcting of image distortions. One of methods for computing 2D-FFT is to carry out twice one-dimensional FFT with respect to rows and columns of two dimensional data (the indirect 2D-FFT method). The well-known Cooley-Tukey algorithm [1] for 1-D FFT can be applied. Recently several researches have been reported for 2D-FFT to be calculated on multiprocessor systems [2][3]. Shorter computational time can be achieved on parallel processing by using multiprocessors in each stage of 2D-FFT computations. Although this type of computer is not yet commercially available, much research in the area indicates their potential advantage in various type of applications.

There is an alternative way to compute 2D-FFT [4][5]. It is the direct method which computes 2D-FFT directly by using 2D-butterfly operations. By this direct method, the number of the multiplications of direct 2-D FFT algorithm can be reduced to 3/4 of the multiplications of indirect 2-D FFT algorithm.

In this paper, we investigate a new 2D-FFT hardware algorithms based on the direct 2D-FFT method. We specifically consider here two kinds of multiprocessor systems. Both systems consist of a mesh connected array of identical Processor Elements (PE's) with minimum storage requirements. The result shows that the shorter processing time can be achieved than the time based on the indirect 2D-FFT on the same multiprocessor systems.

2. 2D-FFT Algorithm

We will consider here an $NxN=N^2$ point two-dimensional discrete Fourier transform implementation where $N=2^n$ or $n=\log_2 N$. Let's $A_0(i:k)$ and $An(u:v)$ be the original and Fourier transformed two dimensional data respectively where $0\leq i,k,u,v\leq N-1$. The two dimensional discrete Fourier Transformation is defined by the equation for all u and v:

$$X(u:v)=\sum_k(\sum_i A_0(i:k)W_N{}^{iu})*W_N{}^{kv} \quad \text{for } 0\leq u,v\leq N-1$$
$$W_N=\exp(-j2\ \ /N) \qquad (1)$$

In this equation, each sum is independent of the other. Therefore they can be computed on after the other using one dimensional FFT techniques.

Direct 2D FFT algorithm was reported by G.E.Rivard [4][5]. The algorithm is the expansion of the algorithm in 1D-FFT case. It consists of stage operations with N^2 two-dimensional data inputs and outputs and in each stage, $N^2/4$ two-dimensional butterfly operations will be carried out.

We express all indices in the form as

$$i=i_{n-1}2^{n-1} + \text{-----} + i_1 2 + i_0$$

and i_k are equal to 0 or 1 and are the contents of the respective bit positions in the binary representation of i. All arrays will now be written as functions of the bits of their indices. The k-th stage 2D-butterfly operations works as follows where $A_{k-1}(i:j)$ and $A_k(i:j)$ are the input and output data of the k-th stage respectively.

$$A_k(i:j)$$
$$=A_k(i_{n-1},\text{--},i_{n-k},\text{--},i_0:j_{n-1},\text{--},j_{n-k},\text{--},j_0)$$
$$=A_{k-1}(i_{n-1},\text{-},0,\text{-},i_0:j_{n-1},\text{-},0,\text{-},j_0)$$
$$+A_{k-1}(i_{n-1},\text{-},0,\text{-},i_0:j_{n-1},\text{-},1,\text{-},j_0)*B_k*(-1)^{i_{n-k}}$$
$$+A_{k-1}(i_{n-1},\text{-},1,\text{-},i_0:j_{n-1},\text{-},0,\text{-},j_0)*C_k*(-1)^{j_{n-k}}$$
$$+A_{k-1}(i_{n-1},\text{-},1,\text{-},i_0:j_{n-1},\text{-},1,\text{-},j_0)*B_k*C_k$$
$$*(-1)^{i_{n-k}}*(-1)^{j_{n-k}}$$
$$\text{where } B_k=W_N{}^{i_{n-k+1}2^{n-2}+\text{ -- } i_{n-1}2^{n-k}},$$
$$C_k=W_N{}^{j_{n-k+1}2^{n-2}+\text{ -- } j_{n-1}2^{n-k}},$$
$$\text{for } 0\leq i,j\leq N-1, \text{ ie. } i_k,\ j_k=0 \text{ or } 1 \qquad (2).$$

The butterfly operations in the 1st stage are carried out among four data $A_0(i:j)$ whose (n-1)th bits of i and j indices are different. In the second stage, they are done among data $A_1(i:j)$ with different (n-2)th bits of i and j indices. In the last stage, the operations are performed among data An(i:j) with different 0th bits of i and j indices.

Each butterfly operation in each stage consists of additions or substractions of the four terms in equations (2). Since all four terms in four equations (2) for $A_k(i:j)$ of only different i_{n-k} and j_{n-k} are the same in value, it will be efficient to perform these four equations as the sets of operations. If we can calculate the three products once for each set of four equations, their execution time for each stage would be proportional to $3*(N^2/4)$ complex multiplication time and $8*(N^2/4)$ addition time.

The outputs An(i,j) of the last n-th stage give the desired Fourier sums. However, the indices of an An(i:j) must have its binary bits put in reverse order to yield its index in the array Xn(i:j) given by equations (1) as

$$Xn(i_{n-1},\text{---},i_0 : j_{n-1},\text{---},j_0)$$
$$=An(i_0,\text{---},i_{n-1} : j_0,\text{---},j_{n-1}) \qquad (3)$$

3. Mesh connected multiprocessor array

We consider here a mesh connected multiprocessor array, because we think mesh connected array may be suitable for 2D Signal Processing. Fig.1 and 2 show examples of 16 PE's of two different type of arrays which we have chosen among various structures of multi-processor systems. Fig.1 is called as a 2D toroidal array in which each PE has a equal position. While, Fig.2 is called as a simple rectangular array. A rectangular array has an advantage that it has only data transfer paths between adjacent PE's which will be suitable to be implemented in future one chip VLSI.

The mesh connected arrays with P PE's and $P=2^m$ are arranged in a $\sqrt{P} \times \sqrt{P}$ square matrix. Each PE is connected to four nearest neighbors in a two dimensional grid. If each PE's number is represented as PE(i,j) according to its geometrical position in a two dimensional grid where i,j=0,--,rP-1, PE (i,j) is connected to four PE's PE(i,j-1),(i,j+1),(i-1,j),(i+1,j) mod rP in a 2D toroidal array and four PE's PE(i,j-1),(i,j+1),(i-1,j),(i+1,j) in a rectangular-array.

We assume for both types of arrays that each PE has four bidirectional I/O ports which can transfer data to and from the neighbor PE's. Furthermore, we assume that each PE can input one data from one of the four neighbor PE's and at the same time can output one data to the four neighbor PE's.

If the number of two dimensional data is N^2, the same amount of data storage capacity will be required. we assume each PE has its local memory with capacity of N^2/P. We additionally assume that there is a central controller that supervises the operation for interprocessor communications.

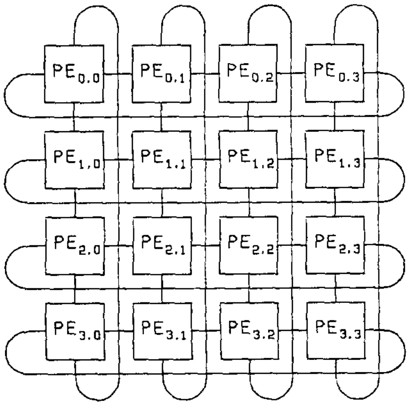


Fig.1 2D toridal Array (P=16)

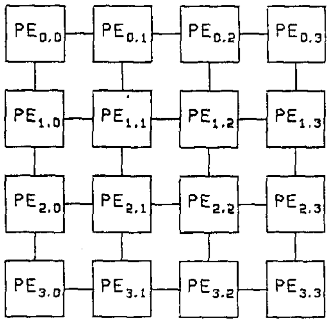


Fig.2 Rectangular Array (P=16)

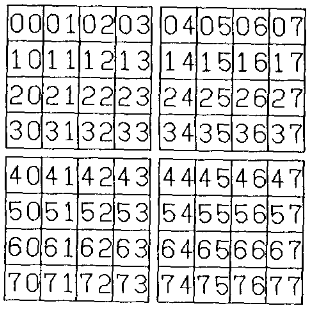
4. 2D-FFT Hardware Algorithm

The input data can allocated in many ways. We consider here the data allocation in which the input N x N matrix data $A_0(i:j)$, $0 \leq i,j \leq N-1$ will be partitioned into square $\sqrt{P} \times \sqrt{P}$ submatricies, A_0kl, $0 \leq k,l \leq \sqrt{P}-1$ and each submatrix A_0kl will be allocated to the PE(k,l). Local memory of PE(k,l) contains N^2/P input data. This type of the data allocation is suitable also for the other image processing applications. An example of this data storage is shown in Fig.3 in the case of N=8 and P=4.

Fig.4 shows the other type of the data allocation for N=8 and P=4. The input N x N matrix data $A_0(i:j)$ will be partitioned into (N/P) x N submatriceies and each one will be allocated to each PE. This type of data allocation has been used for the indirect 2D-FFT algorithm.

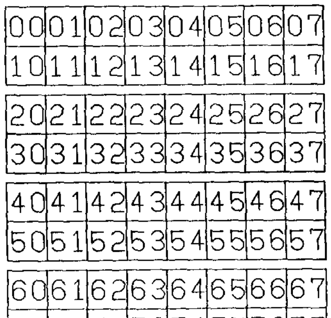
From this initial data allocation, the first several stages of FFT computations require the data transfers. According to our investigation, it is much more efficient to first exchange all the data required to proceed the maximum number of stages of FFT computations without interprocessor communications and then to start FFT computations. If FFT computations will not be able to continue without data transfers, the same procedure will be repeated, that is, to change data allocation and continue FFT computations. The following 2D-FFT hardware algorithm are derived for computing 2D direct FFT. This algorithm is valid for $n \geq 2^m$ or $N \geq P$.

[2D-FFT hardware algorithm]
(1) Bit reverse data transfers
(2) Ist to (n-m)th stage of FFT computations
(3) Bit reverse data transfer
(4) (n-m+1)th to nth stage of FFT computations



00	01	02	03	04	05	06	07
10	11	12	13	14	15	16	17
20	21	22	23	24	25	26	27
30	31	32	33	34	35	36	37
40	41	42	43	44	45	46	47
50	51	52	53	54	55	56	57
60	61	62	63	64	65	66	67
70	71	72	73	74	75	76	77

Fig.3 Square type data allocation (N=8,P=4)



00	01	02	03	04	05	06	07
10	11	12	13	14	15	16	17
20	21	22	23	24	25	26	27
30	31	32	33	34	35	36	37
40	41	42	43	44	45	46	47
50	51	52	53	54	55	56	57
60	61	62	63	64	65	66	67
70	71	72	73	74	75	76	77

Fig.4 Row-wise data allocation (N=8,P=4)

(1) Bit reverse data transfers

Bit-reverse-data-transfers are defined as the transfer operations to transfer all the data from PE's to PE's in such a way that data

$$X(i_{n-1},----,i_0 : j_{n-1},----,j_0)$$

from $PE(i_{n-1},----,i_{n-m} : j_{n-1},----,j_{n-m})$ to

$PE(i_0,i_1,---,i_{m-1} : j_0,j_1,---,j_{m-1})$

for $0 \leq i,j \leq N-1$.

As the result of these operations, the data $A_0(i_{n-1},----,i_0 : j_{n-1},----,j_0)$ of the same $i_{n-1},i_{n-2},---,i_m$ will be stored in the same PE.

(2) 1st - (n-m)th stage of FFT computations

Each PE computes the 1st to (n-m)th butterfly stages without interprocessor communications. The output data

$$A_{n-m}(i_{n-1},----,i_0 : j_{n-1},----,j_0)$$

of (n-m)th stage are stored in

$$PE(i_0,i_1,---,i_{m-1} : j_0,j_1,---,j_{m-1}).$$

To continue this process, interprocessor communication will be needed.

(3) Bit reverse data transfers

The result of the bit reverse data transfer will be that the data

$$A_{n-m}(i_{n-1},----,i_0 : j_{n-1}----,j_0)$$

are stored in

$$PE(i_{n-1},i_{n-2},--,i_{n-m} : j_{n-1},j_{n-2}--,j_{n-m}).$$

The data $A_{n-m}(i_{n-1},----,i_0 : j_{n-1},----,j_0)$ of the same $i_{n-1},i_{n-2},---,i_{n-m}$, which are calculated together in further stages, are stored in the same PE.

(4) (n-m+1)th to nth stage of FFT computations

FFT computation can be continued without interprocessor communications until the final n stages are complete.

5. Data Transfer

The data transfer from PE(i,j) to PE(k,l) consist of several transfers between two neighbors, the number of which are proportional to the distance between the source and destination PE's. Since a 2D-toroidal array has additional connections which have these distances to be shorter, the less number of data transfers for the bit reverse data transfers in a 2D-toroidal array will be required than in a rectangular array.

Data X(i:j) of different $i_{n-m-1},--,i_m$, and $j_{n-m-1},--,j_m$ are stored in $PE(i_{n-1},---,i_{n-m} : j_{n-1},---,j_{n-m})$ and are transferred to $PE(i_0,---,i_{m-1} : j_0,---,j_{m-1})$. Therefore, $2^{n-2m} \times 2^{n-2m} = (N/P)^2$ data have the same source and the same destination PE. In addition to that, each PE distributes equal number of data ($=N^2/P^2$) to all PE's including itself. In this sense, the bit reverse data transfers are equivalent to the transfers of transpose of matrices stored row-wise on mesh-connected multiprocessor array.

5.1 2D toroidal array

Optimum algorithm for this data transfer problem on 2D toroidal array has already been derived [2]. If τ represents a unit transfer time between any two neighbors, the bit reverse data transfers on 2D toroidal array require the transfer time

$$T_{lowest1} = \frac{N^2}{2\sqrt{P}} * \tau \qquad (4).$$

which is independent on algorithms.
The derived algorithm is optimum because its transfer time achieves th lowest bound of the transfer time $T_{lowest1}$.

5.2 Rectangular array

In case of one data transfer from PE(i:j) to PE(k:l) along a shortest path in a rectangular array, the required number of data transfers between two neighbors are determined by the distance between PE's, that is, $|i-k|+|j-l|$. Therefore, the total number of data transfers between two neighbor PE's is obtained by

$$S_a = \sum_{i=0}^{\sqrt{P}-1} \sum_{j=0}^{\sqrt{P}-1} \sum_{k=0}^{\sqrt{P}-1} \sum_{l=0}^{\sqrt{P}-1} (N^2/P^2)*(|i-k|+|j-l|)$$

$$= \frac{2N^2}{3\sqrt{P}} * (P-1) \qquad (5).$$

The lowest bound of the transfer time $T_{lowest2}$ is given as

$$T_{lowest2} = \frac{2N^2}{3P\sqrt{P}} * (P-1)*\tau \qquad (6).$$

[Theorem 1]

There exists no bit reverse data transfer algorithm on a rectangular array which takes the lowest bound of the transfer time $T_{lowest2}$.

In this paper, we propose an transfer algorithm on a rectangular array which is optimum among ones which perform row directional transfers and column directional transfers one after the other.

If we denote by X(is,js)(id,jd) the $(N/P)^2$ data which will be transferred from the source processor PE(is,js) to the destination processor PE(id,jd). X(is,js)(id,jd) will be transferred by row directional transfers from PE(is,js) to PE(is,jd) and then move to PE(id,jd) by column directional transfers. Therefore, in row directional transfers, each PE must transfer $(N/P)^2*\sqrt{P}$ data to each other PE in the same row. After that, each PE distributes the same amount of data to each other PE in the same column. The same algorithms can be applied to these two directional transfers. Therefore, we only show here an algorithm for row directional transfers. In convenience, we define the clock CLK as the transfer time in which $(N/P)^2*\sqrt{P}$ data will be transferred from a PE to its neighbor PE, that is, $1\ CLK = (N/P)^2*\sqrt{P}*\tau$.

We will derive the lower bound $T_{lowest3}$ of the transfer time for row directional transfers which is independent on algorithms. PE(i,j) must send X(is,js)(id,jd) for $0 \leq id \leq \sqrt{P}-1$ and $0 \leq js \leq j < jd \leq \sqrt{P}-1$ to right PE(i,j+1) and also send X(is,js)(id,jd) for $0 \leq id \leq \sqrt{P}-1$ and $\sqrt{P}-1 \geq js \geq j > jd \geq 0$ to left PE(i,j-1). PE(i,j) must transfer $(j+1)*(\sqrt{P}-1-j)*\{(N/P)^2*\sqrt{P}\}$ data to PE(i,j+1) and must transfer $(\sqrt{P}-j)*j$ data to PE(i,j-1). The total number of data transfers through PE(i,j) to either PE(i,j-1) or PE(i,j+1) are

$$\{(j+1)*(\sqrt{P}-1-j)+(\sqrt{P}-j)*j\}*\{(N/P)^2*\sqrt{P}\} \quad (8)$$

The maximum value is obtained by PE(i,j) for $j=\sqrt{P}/2-1$ and $\sqrt{P}/2+1$ which is $(P/2-1)*\{(N/P)^2*\sqrt{P}\}$ or $(P/2-1)$ clock periods. This gives the theoretical lower bound $T_{lowest3}$ of row and column directional bit reverse data transfers in a rectangular array as

$$T_{lowest3} = \frac{N^2*(P-2)}{\sqrt{P}*P}*\tau \qquad (7)$$

[Algorithm for row directional transfers]

(1) Set CLK=0.

(2) When CLK is even, PE(i,j) and PE(i,j+1) for $0\leq i\leq\sqrt{P}-1$, $j=2k$ and $0\leq k\leq\sqrt{P}/2-1$ exchange $(N/P)^2*\sqrt{P}$ data. Set HCLK=CLK/2 and increment CLK by 1 and goto (3).

When CLK is a odd number, PE(i,j) and PE(i,j+1) for $0\leq i\leq\sqrt{P}-1$, $j=2k+1$ and $0\leq k\leq\sqrt{P}/2-2$ exchange $(N/P)^2*\sqrt{P}$ data. Set HCLK = (CLK-1)/2 and increment CLK by 1 and goto (3).

The data to be transferred to the right direction from PE(i,j) to PE(i,j+1) are

$$\begin{aligned} &X(isr,jsr)(idr,jdr)\\ &\quad isr=i, \quad jsr=j-v\\ &\quad idr=0,1,--,\sqrt{P}-1, \quad jdr=(\sqrt{P}-1)-u \end{aligned} \qquad (8)$$

for $jsr<=j<jdr<\sqrt{P}$ where u and v are positive integers of $0\leq v\leq j$ which satisfy $HCLK=(j+1)*u+v$.
If indices don't satisfy $jsr\leq j<jdr<\sqrt{P}$, such data transfers will not be performed.
The data to be transferred to the left direction from PE(i,j+1) to PE(i,j) are

$$\begin{aligned} &X(isl,jsl)(idl,jdl)\\ &\quad isl=i, \quad jsl=j+1+s\\ &\quad idl=0,1,--,\sqrt{P}-1, \quad jdl=r \end{aligned} \qquad (9)$$

where r and s are positive integers which satisfy $HCLK=(\sqrt{P}-j-1)*r+s$ and $0\leq s\leq\sqrt{P}-j-2$.
If indices don't satisfy $jdl\leq j+1<jsl<\sqrt{P}$, such data transfer will not be performed.

(3) If CLK=P/2, stop, otherwise go to (2).

PE(i,j) for even j will perform row right-directional transfers during $(\sqrt{P}-j-1)(j+1)$ clock periods only when CLK's are even numbers and row left-directional transfer during $(\sqrt{P}-j)j$ clock periods only when CLK's are odd numbers. While PE(i,j) for odd j will perform row left-directional transfers during $(\sqrt{P}-j)j$ clock periods only when CLK's are even numbers of and row right-directional transfers during $(\sqrt{P}-j-1)(j+1)$ clock periods only when CLK's are odd numbers. For either odd or even j, PE(i,j) needs $(\sqrt{P}-j-1)(j+1)+(\sqrt{P}-j)j$ clock periods. The row directional data transfers are illustrated as in Fig.5 in the case of $\sqrt{P}=4$.

CLK	PE(i,0)	PE(i,1)	PE(i:2)	PE(i:3)
0	(0 3)=>	<=(1 0)	(2 3)=>	<=(3 0)
1		(1 3)=>	<=(2 0)	
2	(0 2)=>	<=(2 0)	(1 3)=>	<=(3 1)
3		(0 3)=>	<=(3 0)	
4	(0 1)=>	<=(3 0)	(0 3)=>	<=(3 2)
5		(1 2)=>	<=(2 1)	
6				
7		(0 2)=>	<=(3 1)	

Note: (js jd) represents data X(i,js)(id,jd).
Arrows shows the direction of trasfer.

Fig.5 Row directional transfers ($\sqrt{P}=4$)

[Theorem 2] The above algorithm of row right directional transfers of the bit reverse data transfer is optimum among the row-column type transfer algorithms.

6. Comparison

The performance comparison is carried out between the conventional indirect 2D-FFT hardware algorithm and the proposed direct 2D-FFT hardware algorithm on both a 2D-toroidal array and a rectangular array.

The time taken for proposed direct 2D-FFT hardware algorithms are as follows.

(Direct Method 2D-toroidal array)

$$T_{d1}=\frac{3N^2}{4P}(\log_2N-2)Tm+\frac{2N^2}{P}(\log_2N)Ta+\frac{N^2}{\sqrt{P}}\tau \qquad (10)$$

(Direct Method Rectangular array)

$$T_{d2}=\frac{3N^2}{4P}(\log_2N-2,Tm+\frac{2N^2}{P}(\log_2N)Ta+\frac{2N^2}{\sqrt{P}P}(P-2)\tau \qquad (11)$$

where Tm and Ta are the time taken for a complex multiplication and a complex addition on each PE respectively, and τ are the time taken for transferring a datum from a PE to a PE.

To compare the computation time with one by indirect 2D-FFT, we assume to use the same proposed algorithm for data transfers in case of 2D-FFT on a rectangular array.

(Indirect method 2D-toroidal array)

$$T_{i1}=\frac{N^2}{P}(\log_2N-2)Tm+\frac{2N^2}{P}(\log_2N)Ta+\frac{N^2}{2\sqrt{P}}\tau \qquad (12).$$

(Indirect method Rectangular array)

$$T_{i2}=\frac{N^2}{P}(\log_2N-2)Tm+\frac{2N^2}{P}(\log_2N)Ta+\frac{N^2}{\sqrt{P}P}(P-2)\tau \qquad (13).$$

(1) Speedup by direct 2D-FFT

Without the loss by the communication time, P PE's systems can run P times as fast as a single PE. Therefore, speedup efficiency a, defined as

$$a = \frac{\text{(time taken in a single PE)}}{\text{(time taken in P PE's) x P}}$$

gives an efficiency for a multiprocessor system with respect to speed. Fig.6 shows the speedup efficiencies against the addition to transfer operation time ratios for N=256 data, 2D-toroidal array of P=16 PE's and several multiplication to addition operation time ratios. For the practical addition to transfer operation time ratios greater than 5, the multiplication and addition operations are shown to be dominant in 2D-FFT hardware algorithm.

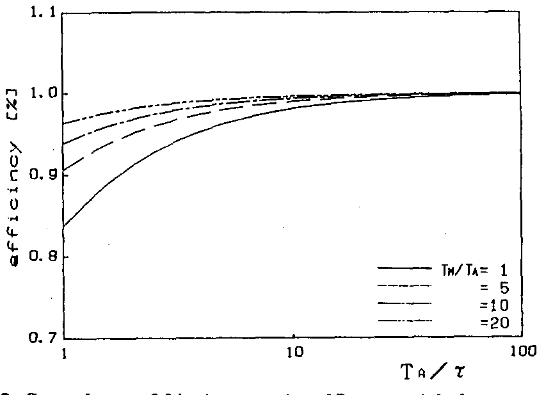


Fig.6 Speedup efficiency in 2D toroidal array for direct 2D-FFT (N=256, P=16)

(2) Comparison to the conventional 2D-FFT

Fig.7 shows the computation time ratios between the proposed direct method and the conventional indirect method on 2D-toroidal array for N=256 and P=256. In the practical case of several Ta/Tm and Tm/τ, the computation time ratios less than 0.9 indicate the higher speed of the proposed one than the conventional one.

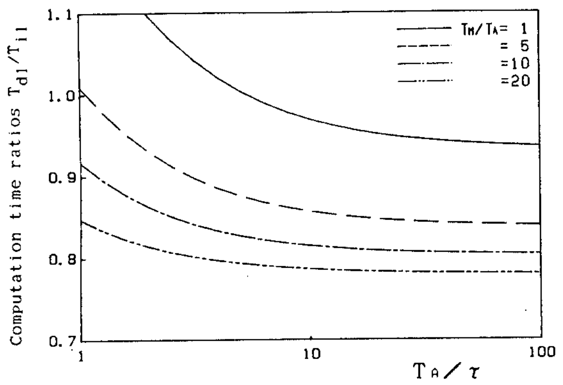


Fig.7

Computation time ratios between direct and indirect 2D-FFT on 2D-toroidal array methods (N=256, P=256)

(3) Comparison between 2D-toroidal and rectangular array

Bit reverse data transfer in a rectangular array needs twice the transfer time as long as in 2D-toroidal array. Since the required time for multiplications and additions are the same in both types, the computation time ratios between two systems as shown in Fig.8 are proportional to interprocessor communication overhead which is determined by the addition to transfer operation time ratio and the number of PE's.

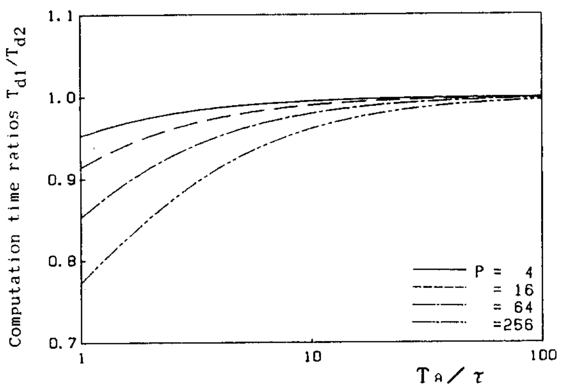


Fig.8

Computation time ratios between 2D toroidal and rectangular array for direct 2D-FFT (N=256, Tm/Ta=5)

7. Concluding Remarks

A algorithm on multiprocessor systems is considered for fast discrete Fourier transformation of two dimensional array. The results are summarized as follows.

(1) Hardware algorithms to implement direct 2D-FFT method on both 2D-toroidal and rectangular array were derived. Although the interprocessor communication time increase, the number of multiplications can be reduced to 3/4 of ones in conventional indirect 2D-FFT case.

(2) The bit reverse data transfer algorithm on 2D-toroidal array is optimum with respect to the transfer time.

(3) The bit reverse data transfer algorithm on a rectangular array was investigated. An derived algorithm is shown to be optimum among algorithms which consist of row and column directional transfers. However, the further investigation on general optimum transfer algorithm will be needed.

References

[1] Cooley,J.W. and Tukey,J.W.: "An Algorithm for the Machine Calculation of Complex Fourier series", Math. Compt., Vol.19, pp.297-301, April, 1965.

[2] Bhuyan,L.N. and Agrawal,D.P.:"Performance Analysis of FFT Algorithms on Multiprocessor Systems", IEEE Trans. Softw. Eng., Vol.SE-9, No.4, pp.512-521, 1983.

[3] Nakano,H. and Tsuda,T.:"Optimizing Inter-Processor Data Transfers in Transposions of Matrices Stored Row-Wise on Mesh-Connected Parallel Computers", Information Processing Society of Japan, Vol.27, No.3, March, 1986.

[4] Rivard G.:"Direct Fast Fourier Transform of Bivariate Functions", IEEE Trans. ASSP, Vol.ASSP-25, pp.250-252, 1977.

[5] Blahut R.:"Fast Algorithms for Digital Signal Processing", Addison-Wesley, 1985.

Appendix

[Proof of Theorem 1]

First, we will count the number of data transfers as possible as many through PE(0,0), one of the corner PE's. PE(0,0) may transfer

(1) $(N/P)^2*(P-1)$ data for data transfers from PE(0:0) to the other (P-1) PE's.

(2) $(N/P)^2*(P-1)$ data for data transfers from PE(0,j) to the other PE(i,0) where $1\leq i,j\leq \sqrt{P}-1$.

(3) $(N/P)^2*(P-1)$ data for data transfers from PE(i,0) to the other PE(0,j) where $1\leq i,j\leq \sqrt{P}-1$.

PE(0,0) need not transfer all the data of (2) and (3), because there are alternative shortest path which don't pass through PE(0,0). Therefore, the total number S of the data (1)-(3),

$$S = \frac{N^2}{P^2}*(2\sqrt{P}-1)*(\sqrt{P}-1) \qquad (A1)$$

will be the maximum possible number of data transfers through PE(0,0).

The total number of data transfers Sa is given in eq.(5). It is easy to show for $P>4$,

$Sa/P > S$.

This means that even our over-estimation of the data transfers through PE(0,0) is less than Sa/P. However, the bit reverse data transfers need Sa transfer as a whole. Therefore there must be at least a PE through which the number of data greater than Sa/P will be transferred. The maximum number of data transfers S' of $S' > Sa/P$ will determine the transfer time T. The transfer time by any algorithm is proved to be longer than the lower bound as

$T=S'\tau > Sa/P*\tau = T_{lowest2}$.

(Q.E.D)

[Proof of Theorem 2]

The algorithm stops when $CLK=P/2$ which gives the transfer time $(P/2)*(N^2/\sqrt{P})\tau$. This value differs from the lowest bound Tlowest3 of the bit reverse transfer algorithm. During the clock period of P/2-1, there are no data transfer in this algorithm. If we skip this clock period, we can perform the row right-directional transfers during (P/2-1) clock periods. Therefore, the algorithm is proved to be the fastest row-column transfer type algorithm.

Next, we will show the derived row-directional transfer algorithm guarantees the arrival of data before the same data should be sent. Let's Tc(j) be a CLK value when A(is:js)(id:jd) for $js<jd$ will be in PE(is:j) during the transfers where $js<j<jd$. Tc(j) are derived from equations as

$Tc(j)=2(\sqrt{P}-1-jd-js)+2(\sqrt{P}-jd)j$ for even j

$Tc(j)=2(\sqrt{P}-1-jd-js)+2(\sqrt{P}-jd)j+1$ for odd j.

Since $\sqrt{P}-jd>0$, Tc(j) are monotone increasing function of j. Furthermore it is easily proved that $Tc(j)<Tc(j+1)<Tc(j+2)$ for any j. That is to say, data will arrive at PE(is,j) before PE(is,j) need the same data to transfer to its neighbor PE.

(Q.E.D.)