Trade-Offs In Mapping FFT Computations onto Fixed Size Mesh Processor Array

Ju Wook Jang

Department of EE-Systems
University of Southern California
Los Angeles, CA 90089-0781

Wojtek Przytula

Hughes Research Labs
3011 Malibu Canyon Road
Malibu, CA90265.

Abstract

The FFT computation involves global data transfer. Inefficiencies can result when implementing large FFT computations on mesh arrays with small local storage. Several factors are to be considered when mapping FFT computation onto mesh arrays. These include amount of available local storage, I/O bandwidth, concurrent execution of I/O, arithmetic logic operations within the PEs, and interprocessor communication operations. Indeed, the mapping of the computation is farther complicated by hardware features such as multi function pipeline, multi port memories in the PEs. In this paper several mappings of FFT computation are evaluated with respect to I/O time, computation time and communication time on a $p \times p$ Systolic/Cellular Array Processor developed at Hughes Research Labs. Various mappings are obtained by modifying the FFT signal flow graph.

1 Intruduction

Since the introduction of the well-known Cooley and Tuckey FFT algorithm[13] for computing the DFT, many researchers have developed a variety of FFT algorithms[15, 14, 16]. Consequently, VLSI circuits for DFT have been extensively studied[4, 5, 6, 7]. Thompson has proved that any VLSI circuit that computes the DFT on N points in T time units and using A area must have $AT^2 = \Omega(N^2 \log^2 N)$[11]. Several VLSI implementations show asymptotic optimality in the AT^2 measure. Vuillemin has implemented N point FFT in $O(\log N)$ time on a FFT network whose area is $O(N^2)$[8]. Preparata and Vuillemin implemented the FFT in $O(\log^2 N)$ time on a cube connected cycle network with $O(\frac{N^2}{\log^2 N})$ area[9]. The implementation on mesh connected network takes $O(N^{\frac{1}{2}})$ time and $O(N \log^2 N)$ area[10].

Among the proposed implementations, the mesh implementation is of practical importance, since the mesh is simple to build and many signal processing algorithms have been developed on such arrays. Since asymptotically optimal implementations of FFT on the mesh are known, efforts should be directed to reduce the constant related to the AT^2 performance. Also, in a number of FFT applications N is not large, but computation is repeated over large sets of data. In this case the constant factor hidden in the asymptotic performance will have significant impact on the performance. Tran Thong[3] has suggested a method to reduce the constant related to the area complexity by half through refactorization of the Fourier coefficients in FFT.

In this paper, we propose a modification of the implementation presented in [1]. The modification is based on use of alternative form of FFT flow diagram. As a result the interprocessor communication time can be reduced by half. The scheme is applicable to mesh connected arrays of processing elements, each of which has a small local storage. As pointed out by Thompson[11], in measuring time performance of N point FFT on $\sqrt{N} \times \sqrt{N}$ mesh, the interprocessor communication time, which is $O(\sqrt{N})$, plays dominant role. The interprocessor communication time in the FFT implementation by Wojtek et al.[1] is $4\sqrt{N}$ while that of the systolic DFT implementation[12] is $2\sqrt{N}$. Our scheme results in reducing the processing time close to that of [12] while keeping the local storage equal to [1].

This saving in time performance is obtained by fully exploiting the mesh connection whose idle time is almost 50% in [1] and 0% in [12]. In [1], the East/West(E/W) mesh connection is idle during the

TH0363-2/91/0000/0170$01.00 © 1991 IEEE

stages requiring North/South(N/S) communication. We have virtually reduced the idle time of mesh connection to zero by modifying the FFT formula so that the resulting signal flow graph allows the E/W communication to be overlapped with N/S communication.

Section 2 describes the architecture. Section 3 defines the FFT problem. Section 4 begins with a simple implementation of mapping FFT onto the mesh array. Corresponding signal flow graph of the implementation is given and the resulting utilization of mesh connection is shown. In section 4.2, modification of FFT formula and its resulting signal flow graph is shown. Detailed mapping is described with analysis of time performance and utilization of mesh connection.

2 The Architecture

The parallel implementation of FFT discussed in this paper has been developed in conjunction with our work on implementation of signal and image processing algorithms on a mesh-connected SIMD coprocessor developed at Hughes. The basic architecture and its implementation are presented briefly in this section.

The diagram of the machine is shown in figure 1. The coprocessor consists of an array of processing elements, dualport array memory, the controller with program memory and the interface with host. The characteristic feature of the architecture are very high bandwidth connections between the processors of the array, and between the array and the array memory. The processing elements of the array are controlled in unison by a single SIMD controller.

The processing elements are connected to four nearest neighbors and the boundary processors are connected by wraparound links. The top and bottom rows of the processor array access data in the array memory by rows using a single address per memory port. The processors of the array may be selectively disabled. The array operates in two modes systolic and cellular. In the cellular mode, which is utilized for our FFT implementation, first the data are loaded into the processor array from the array memory, then the computation in the array takes place, and finally the results are unloaded back into the memory. In the systolic mode these three phases are pipelined down to the level of individual data rows.

The processing elements, of which basic diagram is shown in figure 2, consists of multiple functional elements, multiported local memory, I/O ports, and

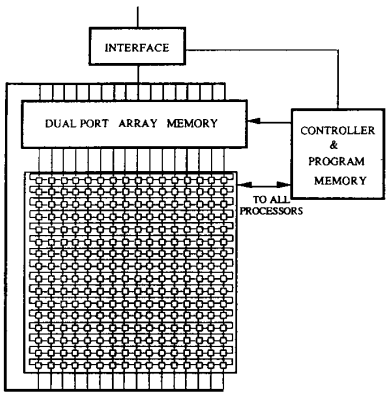

Figure 1: The Hughes Systolic/Cellular Array Processor

instruction decode and masking unit. The I/O port can be used concurrently with the functional units, i.e. communications and computations can be overlapped. Also several functional units in the processing element can operate concurrently, providing one more level of concurrence.

The common controller sequences the instructions stored in the program memory and broadcasts them to the processing elements. There is a feedback connection from the processors to the controller, which enables the coprocessor to perform data dependent computations.

The architecture has been implemented as a prototype coprocessor for control of robot arm [2]. It is a 16 by 16 array of custom, 32-bit, fixed point processing elements each consisting of 24 registers of local memory, two multipliers, two adders, divider and logic unit. The data memory is based on static dualported RAM 256 bit wide and 2K deep. The array is controlled by wide instruction word, with separate fields for functional units, I/O and system control. A new implementation of the architecture is under development at Hughes.

The implementation of FFT presented in the paper is applicable to a large class of meshconnected arrays. We will make some assumptions about the rate of computation and communication, which do not necessarily reflect those of our prototypes but are selected to provide simple illustration of our approach.

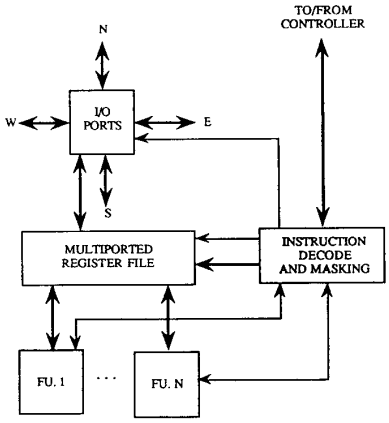

Figure 2: Block diagram of PE

3 The Problem

A number of applications of FFT in radar, image and signal processing can be represented as the following problem:

Input $S_I(n,k)$. $n = 0 : N-1$, N data blocks, $k = 0 : K-1$, K data points per each block. Each data point is represented by a 64 bit complex number. Thus, K point FFT is computed N times.

Processing

for $n = 0 : N-1$
  for $k = 0 : K-1$
  $S_O(n,m) = \frac{1}{K}\sum_{k=0}^{K-1} S_I(n,k)exp(-2j\pi\frac{km}{K})$
  end
end

Output $S_O(n,m)$ $n = 0 : N-1$ data blocks, $m = 0 : K-1$ output coefficients per block.

We employ the decimation-in-frequency FFT algorithms [13] in which the number of multiplications is reduced to $N/2 \times \log_2 N$ from N^2 for N point DFT while keeping the data flow to be regular.

4 Mapping of the FFT computation onto processor array

First, we will show a mapping which results directly from the FFT formula[1], and discuss the processing time and the utilisation of the mesh connections. Then we propose a scheme to overlap E/W communication with N/S communication and discuss the time reduction and utilisation of the mesh connections.

Implementation of the FFT can be divided into three parts.

Preprocessing: This is an I/O operation in which unloading of the results of the previous block and loading of new input block is performed.

Main processing: For K point FFT, the main processing is done in $\log K$ stages. For each stage, $\frac{K}{2}$ butterflies are computed. Each butterfly computation comprises of one complex addition, one complex subtraction and one complex multiplication.

Postprocessing: The output of the FFT algorithm is in bit-reversal order. This is rearranged into a lexicographic order.

If the local memory of processing elements is sufficiently large the three phases, mentioned above, can be pipelined. In this case the improvements resulting from the overlap may not be significant.

4.1 A direct mapping

First, recall that an N-point DFT is represented by the following:

$$X(k) = \sum_{n=0}^{N-1} x(n)exp(-\frac{j2\pi nk}{N}) \qquad (1)$$

Let's consider one example where $N = 64$. Then n and k can be represented as follows:

$$\begin{aligned} n &= 32n_5 + 16n_4 + 8n_3 \\ &\quad +4n_2 + 2n_1 + n_0 \qquad (2) \\ k &= 32k_5 + 16k_4 + 8k_3 \\ &\quad +4k_2 + 2k_1 + k_0 \qquad (3) \end{aligned}$$

where $n_i = 0, 1$ and $k_i = 0, 1$.

The $exp(-\frac{j2\pi nk}{N})$ in equation 1 can be rewritten as follows using the above binary representation

for n:

$$exp(-\frac{j2\pi 32n_5k}{N})exp(-\frac{j2\pi 16n_4k}{N})$$
$$exp(-\frac{j2\pi 8n_3k}{N})exp(-\frac{j2\pi 4n_2k}{N})$$
$$exp(-\frac{j2\pi 2n_1k}{N})exp(-\frac{j2\pi n_0k}{N}) \quad (4)$$

Then, we have the following [3]:

$$X_0(n_0, n_1, \ldots, n_5) = x(n) \quad (5)$$

$$X_1(n_0, n_1, \ldots, n_4, k_0) = \sum_{n_5=0}^{1}[X_0(n_0, n_1, \ldots, n_5)] * exp(-\frac{2\pi 32n_5k_0}{64}) \quad (6)$$

$$X_2(n_0, n_1, \ldots, n_3, k_1, k_0) = \sum_{n_4=0}^{1}[X_1(n_0, \ldots, n_4, k_0)] * exp(\frac{-j2\pi 16n_4(k_0+2k_1)}{64}) \quad (7)$$

$$X_3(n_0, n_1, n_2, k_2, k_1, k_0) = \sum_{n_3=0}^{1}[X_2(n_0, \ldots, n_3, k_1, k_0)] * exp(\frac{-j2\pi 8n_3(k_0+2k_1+4k_2)}{64}) \quad (8)$$

$$X_4(n_0, n_1, k_3, k_2, k_1, k_0) = \sum_{n_2=0}^{1}[X_3(n_0, \ldots, n_2, k_2, k_1, k_0)] * exp(\frac{-j2\pi 4n_2(k_0+\ldots+8k_3)}{64}) \quad (9)$$

$$X_5(n_0, k_4, k_3, k_2, k_1, k_0) = \sum_{n_1=0}^{1}[X_4(n_0, n_1, k_3, k_2, k_1, k_0)] * exp(\frac{-j2\pi 2n_1(k_0+\ldots+16k_4)}{64}) \quad (10)$$

$$X_6(k_5, k_4, k_3, k_2, k_1, k_0) = \sum_{n_0=0}^{1}[X_5(n_0, k_4, k_3, k_2, k_1, k_0)] * exp(\frac{-j2\pi n_0(k_0+\ldots+32k_5)}{64}) \quad (11)$$

The signal flow graph resulting from the above expansion is as shown in figure 3.

If we map the 64-point FFT onto a 4 × 4 subarray, we have four data layers[1]. One layer is composed of size 4 × 4 input points. The data in each layer are arranged as follows.

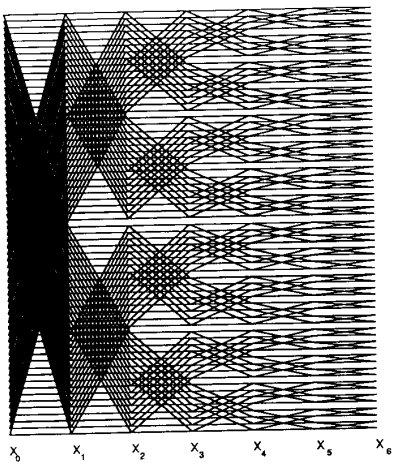

Figure 3: The signal flow graph of a 64 point FFT used in the simple implementation

- Layer 0 has inputs $x(0), \ldots, x(15)$ in row-major order.
- Layer 1 has inputs $x(16), \ldots, x(31)$ in row-major order.
- Layer 2 has inputs $x(32), \ldots, x(47)$ in row-major order.
- Layer 3 has inputs $x(48), \ldots, x(63)$ in row-major order.

For example, the $PE(0,0)$ has four inputs $x(0), x(16), x(32), x(48)$ and the $PE(0,1)$ has four inputs $x(1), x(17), x(33), x(49)$ where $PE(i,j)$ represents the Processing Element at the intersection of i^{th} row and j^{th} column of the subarray. The main processing part has $\log 64 = 6$ stages and 32 butterflies are computed in each stage. At t^{th} stage, 32 pairs of two data points whose indices are separated by $\frac{64}{2^{t+1}}$ are used for butterfly computation. If the two points are apart by (a, b) in the processor array, the $2b$ E/W communication and $2a$ N/S communication are needed before the computation of the butterfly. The detailed mapping for the main processing part is as follows:

stage 0 and 1: Processing of equation 6 and 7. No transfer of data points are necessary.

stage 2: Processing of equation 8. Two North transfer is followed by two South transfer for layer 0,1,2 and 3.

stage 3: Processing of equation 9. One North transfer is followed by one South transfer for layer 0,1,2 and 3.

stage 4: Processing of equation 10. Two East transfer is followed by two West transfer for layer 0,1,2 and 3.

stage 5: Processing of equation 11. One East transfer is followed by one West transfer for layer 0,1,2 and 3.

Let us define the following:

The definition for the times is as follows:

- T_{EW}: The time needed for one East transfer of one data point and one West transfer of one data point using E/W connection.
- T_{NS}: The time needed for one North transfer of one data point and one South transfer of one data point using N/S connection.
- T_{comp}: Time for computing one butterfly between two data points in a PE.

Let us assume for simplicity that $T_{NS} = T_{EW} = T_{comp}$. The timing diagram for the simple implementation is shown in figure 4. One narrow block stands for simple transfer or computation step. The double blocks are where data are moved by distance of two processors. The shaded blocks appear where the actual transfer or computation are taking place. The blank blocks are only placeholders and indicate idle cycles of a given resource i.e. E/W or N/S communication links or functional units within the processors.

The main processing time will be $4T_{comp} + 12(T_{EW} + T_{NS})$ if $T_{EW} = T_{NS} \geq T_{comp}$. The idle times for a few hardware resources are as follows(Percentage is calculated against the main processing time).

- E/W connection: $4T_{comp} + 12T_{NS}$, 58%
- N/S connection: $4T_{comp} + 12T_{EW}$, 58%
- ALU: $(8T_{EW} + 8T_{NS} - 8T_{comp})$, 29%

4.2 Overlapping E/W communication with N/S communication

After using equations 10, 9, 8 in equation 11, the resulting equation can be divided into four classes depending on four combinations of (k_1, k_0).

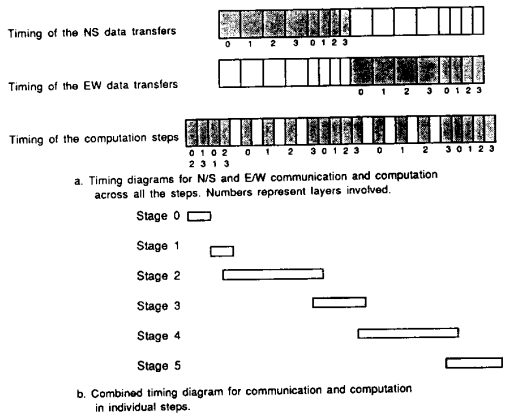

Figure 4: Timing diagram for a simple implementation of a 64-point FFT on a 4×4 processor array

$$
\begin{aligned}
& X_6(k_5, k_4, k_3, k_2, 0, 0) \\
= \; & \textstyle\sum_{n_0=0}^{1}[\sum_{n_1=0}^{1}[\sum_{n_2=0}^{1}[\sum_{n_3=0}^{1} \\
& X_2(n_0, n_1, n_2, n_3, 0, 0) \\
& *exp(-\tfrac{j2\pi 8n_3(4k_2)}{64})] \\
& *exp(-\tfrac{j2\pi 4n_2(4k_2+8k_3)}{64})] \\
& *exp(-\tfrac{j2\pi 2n_1((4k_2+\ldots+16k_4)}{64})] \\
& *exp(-\tfrac{j2\pi n_0(4k_2+\ldots+32k_5)}{64})] \qquad (12)
\end{aligned}
$$

$$
\begin{aligned}
& X_6(k_5, k_4, k_3, k_2, 0, 1) \\
= \; & \textstyle\sum_{n_0=0}^{1}[\sum_{n_1=0}^{1}[\sum_{n_2=0}^{1}[\sum_{n_3=0}^{1} \\
& X_2(n_0, n_1, n_2, n_3, 0, 1) \\
& *exp(-\tfrac{j2\pi 8n_3(1+4k_2)}{64})] \\
& *exp(-\tfrac{j2\pi 4n_2(1+4k_2+8k_3)}{64})] \\
& *exp(-\tfrac{j2\pi 2n_1(1+4k_2+\ldots+16k_4)}{64})] \\
& *exp(-\tfrac{j2\pi n_0(1+4k_2+\ldots+32k_5)}{64})] \qquad (13)
\end{aligned}
$$

$$
\begin{aligned}
& X_6(k_5, k_4, k_3, k_2, 1, 0) \\
= \; & \textstyle\sum_{n_0=0}^{1}[\sum_{n_1=0}^{1}[\sum_{n_2=0}^{1}[\sum_{n_3=0}^{1} \\
& X_2(n_0, n_1, n_2, n_3, 1, 0) \\
& *exp(-\tfrac{j2\pi 8n_3(2+4k_2)}{64})] \\
& *exp(-\tfrac{j2\pi 4n_2(2+4k_2+8k_3)}{64})]
\end{aligned}
$$

$$*exp(-\frac{j2\pi 2n_1(2+4k_2+...+16k_4)}{64})]$$
$$*exp(-\frac{j2\pi n_0(2+4k_2+...+32k_5)}{64})] \quad (14)$$

$$X_6(k_5,k_4,k_3,k_2,1,1)$$
$$= \sum_{n_0=0}^{1}[\sum_{n_1=0}^{1}[\sum_{n_2=0}^{1}[\sum_{n_3=0}^{1}$$
$$X_2(n_0,n_1,n_2,n_3,k_1,k_0)$$
$$*exp(-\frac{j2\pi 8n_3(1+2+4k_2)}{64})]$$
$$*exp(-\frac{j2\pi 4n_2(1+2+4k_2+8k_3)}{64})]$$
$$*exp(-\frac{j2\pi 2n_1(1+2+4k_2+...+16k_4)}{64})]$$
$$*exp(-\frac{j2\pi n_0(1+2+4k_2+...+32k_5)}{64})] \quad (15)$$

We change the order of summation in equations 13 and 15 without affecting the output. Now equations 12 thru 15 will be as follows:

$$X_6(k_5,k_4,k_3,k_2,0,0)$$
$$= \sum_{n_0=0}^{1}[\sum_{n_1=0}^{1}[\sum_{n_2=0}^{1}[\sum_{n_3=0}^{1}$$
$$X_2(n_0,n_1,n_2,n_3,0,0)$$
$$*exp(-\frac{j2\pi 8n_3(4k_2)}{64})]$$
$$*exp(-\frac{j2\pi 4n_2(4k_2+8k_3)}{64})]$$
$$*exp(-\frac{j2\pi 2n_1((4k_2+...+16k_4)}{64})]$$
$$*exp(-\frac{j2\pi n_0(4k_2+...+32k_5)}{64})] \quad (16)$$

$$X_6(k_5,k_4,k_3,k_2,0,1)$$
$$= \sum_{n_2=0}^{1}[\sum_{n_3=0}^{1}[\sum_{n_0=0}^{1}[\sum_{n_1=0}^{1}$$
$$X_2(n_0,n_1,n_2,n_3,0,1)$$
$$*exp(-\frac{j2\pi 2n_1(1+4k_2+...+16k_4)}{64})]$$
$$*exp(-\frac{j2\pi n_0(1+4k_2+...+32k_5)}{64})]$$
$$*exp(-\frac{j2\pi 8n_3(1+4k_2)}{64})]$$
$$*exp(-\frac{j2\pi 4n_2(1+4k_2+8k_3)}{64})] \quad (17)$$

$$X_6(k_5,k_4,k_3,k_2,1,0)$$
$$= \sum_{n_0=0}^{1}[\sum_{n_1=0}^{1}[\sum_{n_2=0}^{1}[\sum_{n_3=0}^{1}$$
$$X_2(n_0,n_1,n_2,n_3,1,0)$$
$$*exp(-\frac{j2\pi 8n_3(2+4k_2)}{64})]$$
$$*exp(-\frac{j2\pi 4n_2(2+4k_2+8k_3)}{64})]$$
$$*exp(-\frac{j2\pi 2n_1(2+4k_2+...+16k_4)}{64})]$$
$$*exp(-\frac{j2\pi n_0(2+4k_2+...+32k_5)}{64})] \quad (18)$$

$$X_6(k_5,k_4,k_3,k_2,1,1)$$

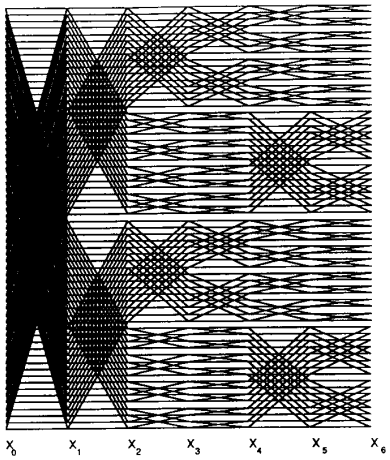

Figure 5: The modified signal flow graph for a 64-point FFT used in the overlapped implementation

$$= \sum_{n_2=0}^{1}[\sum_{n_3=0}^{1}[\sum_{n_1=0}^{1}[\sum_{n_0=0}^{1}$$
$$X_2(n_0,n_1,n_2,n_3,k_1,k_0)$$
$$*exp(-\frac{j2\pi 2n_1(1+2+4k_2+...+16k_4)}{64})]$$
$$*exp(-\frac{j2\pi n_0(1+2+4k_2+...+32k_5)}{64})]$$
$$*exp(-\frac{j2\pi 8n_3(1+2+4k_2)}{64})]$$
$$*exp(-\frac{j2\pi 4n_2(1+2+4k_2+8k_3)}{64})] \quad (19)$$

The modified signal flow graph resulting from the above conversion of the formula is shown in figure 5. Comparing this figure with figure 3, we can note the difference. Starting from the stage $2(X_2)$ thru the end of the stage $5(X_5)$, the signal flow of the first quarter and and third quarter remains same but those of second quarter and fourth quarter have been changed. The transfer for stage 2 of first and third quarters is two north transfer followed by two south transfer while the transfer for stage 2 of second and fourth quarter is two east transfer followed by two west transfer. Note that each quarter represents one data layer. Since in stage 2 the transfer in layer 0 uses N/S connection while transfer in layer 1 uses E/W connection, they can be overlapped. Then the transfer for layer 2 and 3 can be concurrently done while computation is performed on layer 0 and 1. By the same reason, N/S communication can be overlapped with E/W communication in the remaining stages.

stage 0 and 1: same as before.

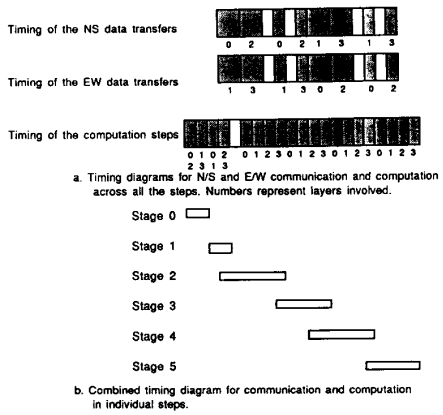

Figure 6: Timing diagram for an overlapped implementation of a 64-point FFT on a 4×4 processor array

stage 2: Two north transfer is followed by two south transfer for layer 0 and 2. Two east transfer is followed by two west transfer for layer 1 and 3.

stage 3: One north transfer is followed by one south transfer for layer 0 and 2. One east transfer is followed by one west transfer for layer 1 and 3.

stage 4: Two east transfer is followed by two west transfer for layer 0 and 2. Two north transfer is followed by two south transfer for layer 1 and 3.

stage 5: One east transfer is followed by one west transfer for layer 0 and 2. Two north transfer is followed by two south transfer for layer 1 and 3.

The timing diagram is shown in figure 6, here again, $T_{EW} = T_{NS} = T_{comp}$ is assumed.

The processing time will be $8T_{EW} + 8T_{NS} + 5T_{comp}$ if $T_{EW}, T_{NS} \geq T_{comp}$. The percentage of the idle cycle of the mesh connections will converge to 0% with large N, since the transfer time increases with the size of the FFT while the computation time for each PE is fixed. So the utilisation of the mesh connections can be maximised with this scheme.

Table 1 shows the savings in processing time by our scheme for various sizes of FFT when $T_{NS} = T_{EW} = T_{comp}$ is assumed.

We can generalise the discussion as follows:

- N-point FFT where $N = 2^n$.
- $P \times P$ processor array where $P = 2^p$.
- We have $R = N/P^2$ layers and $R \geq 4$.

N point FFT	mapped onto	savings in time
64	4×4	28%
256	8×8	40%
1024	16×16	45%
4096	32×32	47%

Table 1: Savings in processing time by the overlapping of E/W communication with N/S communication

Then, the reduction in the constant factor of the time complexity(T) will be as follows:

1. If $T_{EW} \geq T_{NS}$, the reduction is:

$$\frac{R(P-1)T_{NS} - 4T_{comp}}{4T_{comp} + R(P-1)(T_{EW} + T_{NS})}$$

2. If $T_{EW} \leq T_{NS}$, the reduction is:

$$\frac{(P-1)T_{EW} - 4T_{comp}}{4T_{comp} + R(P-1)(T_{EW} + T_{NS})}$$

Note that the reduction ratio converges to 0.5 when we assume large P and $T_{EW} = T_{NS}$.

5 Conclusion

We described modification of a FFT implementation scheme for mesh-connected arrays presented in [1]. The modification leads to significant reduction of the processing time resulting from cutting data transfer time by half. This is accomplished by reorganization of the FFT flow diagram of the original implementation so that horizontal and vertical data transfers in the processor mesh can be overlapped.

References

[1] K. Wojtek Przytula et al, *Fast Fourier transform algorithm for two-dimensional array of processors*, SPIE Symposium on Optical and Optoelectronic Applied Science and Engineering, San Diego, August, 1987.

[2] K. Wojtek Przytula and J. Greg Nash, *A Special Purpose Coprocessor For Robotics And Signal Processing*, International Conference on Robotics and Automation, Philadelphia, April 1988.

[3] Tran Thong, *FFT with Reduced Coefficient Storage Requirement*, Int. Conf. on Parallel Processing 1990.

[4] G. Bilardi and M. Sarrafsadeh, *Optimal Discrete Fourier Transform in VLSI*, Advances in Computing Research, PP. 87-101.

[5] C. Chakrabarti and J. Ja'Ja', *A Family of Optimal Architectures for Multidimensional Transforms*, 26th Allerton conf. on Communication, Control, and Computing, 1988.

[6] I. Gertner and M. Shamash, *Architecture for Multidimensional Fourier Transform Processing*, IEEE Transactions on Computers, c-36, Nov. 1987.

[7] K. Keutser and E. Rovertson, *The M-Shuffle as an Interconnection Network for SIMD Machines*, 20th Allerton Conf. on Communication, Control, and Computing, 1982.

[8] J. Vuillemin, *A combinatorial limit to the computing power of VLSI circuits*, in Proc. 21st Symp. Foundations of Comput. Sci., IEEE Comput. Soc., Oct. 1980, pp. 294-300.

[9] F. Preparata and J. Vuillemin, *The cube-connected cycles: A versatile network for parallel computation*, in Proc. 20th Annu. Symp. Foundations of Comput. Sci., IEEE Comput. Soc., Oct. 1979, pp. 140-147.

[10] Stevens, *A fast Fourier transform subroutine for Illiac IV*, Cen. Advanced Comput. , Univ. Illinois. Tech. Rep., 1971.

[11] C. D. Thompson, *Fourier Transforms in VLSI*, IEEE Trans. Comp. C-32, No.32 pp. 1047-1057 November 1983.

[12] Nam Ling and M. A. Bayoumi, *Algorithms for High speed Multi-dimensional Arithmetic and DSP Systollic Arrays* Proc. Int. Conf. Parallel Processing 1988.

[13] Cooley, J. W., and Tuckey, J. W., *An algorithm for the machine calculation of complex Fourier series*, Math. Comput., 19 pp. 297-301 1965.

[14] Good, I. J., *The Relationship between Two Fast Fourier Transforms*, IEEE Trans. Comp. c-20 pp. 310-317 1971.

[15] Goertsel, G., *An Algorithm for the Evaluation of Finite Trigonometric Series*, Amer. Math. Monthly 65 pp. 34-35 1968

[16] Winograd S., *On Computing the discrete Fourier Transform*, Math Comp. 32 pp. 175-199 1978.