

Computation of the Fast Fourier Transform on the Star-Connected Cycle Networks

S. Shoari and N. Bagherzadeh
University of California
Department of Electrical and Computer Eng.
Irvine, CA 92717

e-mail: simin@ece.uci.edu

Abstract

Two parallel algorithms for computing Fast Fourier Transform(FFT) on a new type of architecture called the n -Star-Connected Cycles (n -SCC) have been considered. The n -SCC architecture is based on the n -star interconnection graph and has all the desirable features which exist in the n -star interconnection graph. To demonstrate the main properties of an n -SCC network, we present two parallel algorithms for computing the FFT. These algorithms are based on the Cooley-Tukey FFT algorithm and combine the principles of parallelism and pipelining. Considering the same number of inputs, the second algorithm in this paper has better performance in compare to the algorithm implemented on star graph.

Key Words: FFT, parallel algorithm, n -star graph, and n -SCC network.

1 Introduction

Since the introduction of Discrete Fourier Transform (DFT) and Fast Fourier Transform (FFT) in 1965 [3], FFT algorithms have been extensively studied. In order to obtain higher throughput, researchers have been putting their efforts into two aspects: effective algorithm design and faster architecture. During the 1970s, array processors were produced and widely used to execute

different algorithms [4]. Later, many parallel FFT algorithms on SIMD and MIMD architectures were developed. Thus, FFT has become a necessary tool in many applications for parallel architectures. Fragopoulou, et al. [4] considered the FFT algorithm on an $n - star$ architecture.

In this paper we consider a new type of architecture called $n - SCC$ [5]. This architecture is based on $n - star$ graphs. To demonstrate the main properties of an $n - SCC$ network, we present two different parallel algorithms for computing the FFT. These algorithms combine the principles of parallelism and pipelining.

In Section 2, the $n - SCC$ architecture is described. Section 3 considers basic properties of FFT. In Section 4, Concurrent Pipeline Algorithm and Parallel Pipeline Algorithm for computing FFT on an $n - SCC$ are described. The conclusion is presented in Section 5.

2 Description of the $n - SCC$ Network

The $n - SCC$ network is a new type of interconnection network for multiprocessors.

Definition 1: The $n - SCC$ network is constructed from an $n - star$ graph S_n by replacing each node by $n - 1$ nodes connected in a form of ring [5].

In order to explain this architecture, $n - star$ will be described in the following section.

2.1 The *Star* Graph

An $n - star$ graph, S_n , defined by Aker et al. [1, 2] as an alternative to the hypercube interconnection. It contains $n!$ nodes corresponding to the $n!$ permutations of n different symbols. We use numbers 1 through n to represent these symbols. Two nodes $u, v \in S_n$ are connected if and only if their addresses differ exclusively in the first and any other symbol. Therefore, each node has direct communication links to $n - 1$ other nodes. Figure 1 depicts the structure of the $4 - star$. The edge connecting two nodes with addresses resulting from interchanging between the first and i^{th} symbols is called edge in the i^{th} dimension. In the Figure 1, the number next to each edge represents the dimension. For example the edge connecting nodes with labels 1234 and 3214 is in the 3^{rd} dimension.

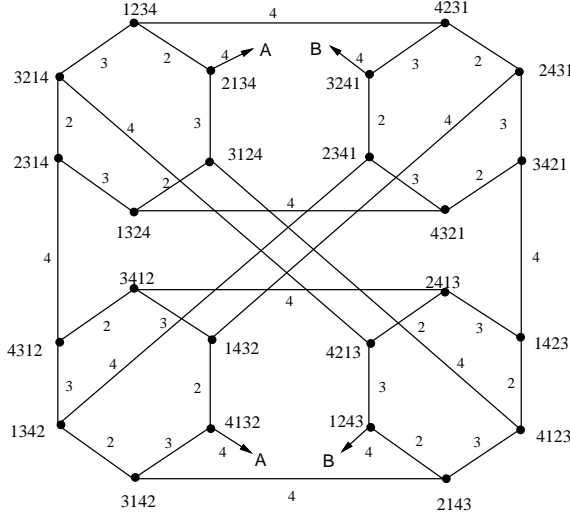


Figure 1: The 4 – star interconnection graph

An n – star graph can be decomposed into different forms. One form is decomposing an n – star into $(n - 2)!$ disjoint cycles of length $(n - 1)n$ each [8]. Each cycle is defined by its initial permutation. Assume the initial permutation of a cycle is $1a_2a_3\dots a_{n-1}n$. The other permutations are generated by interchanging a_2, a_3, \dots, a_{n-1} , and n until the initial permutation is repeated.

2.2 n – Star – Connected Cycles

In an n – Star – Connected Cycles (n – SCC) each node of an n – star is replaced by $(n - 1)$ nodes, thus the number of nodes in an n – SCC network is $N = (n - 1)n!$, for $n > 2$.

The address of each node in an n – SCC is represented by a 2–tuple (i, R_j) , where R_j is a permutation of integers $1, 2, \dots, n$ and represents the address of the ring containing the node. The value of i is the address of the node within the ring, $2 \leq i \leq n$. The nodes are numbered clockwise. Figure 2 shows the n – SCC network for $n = 4$.

In an n – SCC, each node with address (i, R_j) is connected to three other nodes. Thus the degree of each node is 3. Two of these links are called *local links* which connect nodes within a ring (right and left neighbors). The third link is called *lateral link* and connects a node in one ring to the other node

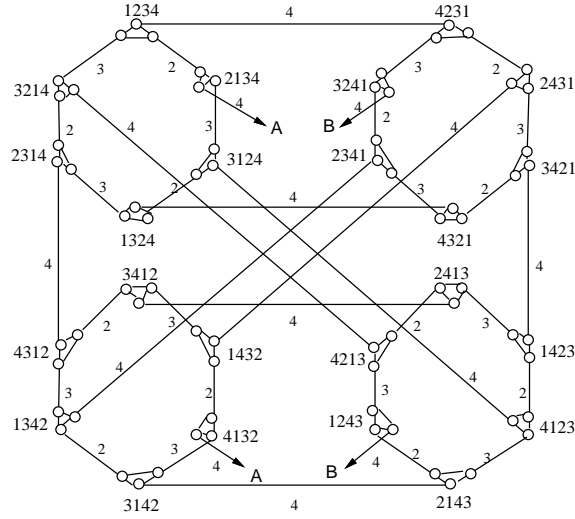


Figure 2: The 4 – *SCC* interconnection network

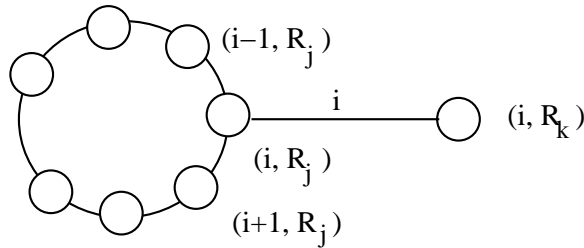


Figure 3: The node (i, R_j) and its adjacent nodes.

belonging to the adjacent ring. On the other word, lateral links connect rings (Figure 3).

3 Description of the Fourier Transform Algorithm

Assume that N , the number of input elements, is a composite number with only two factors r_1 and r_2 , $N = r_1 * r_2$. The FT can be written as (see Appendix A):

$$X(i_1, i_2) = \sum_{k_2=1}^{r_2} \omega^{i_1 k_2 r_1} \omega^{i_2 k_2} \sum_{k_1=1}^{r_1} x(k_1, k_2) \omega^{i_2 (k_1-1) r_2}$$

Therefore, $X(i_1, i_2)$ can be computed in the following steps [3]:

- **Step 1**

Part a.

$$X'(i_2, k_2) = \sum_{k_1=1}^{r_1} x(k_1, k_2) \omega^{i_2 (k_1-1) r_2}$$

Part (a) computes FT of each column of the array.

Part b. $X''(i_2, k_2) = \omega^{i_2 k_2} X'(i_2, k_2)$

Part (b) performs local multiplication.

- **Step 2**

$$X'''(i_1, i_2) = \sum_{k_2=1}^{r_2} X''(i_2, k_2) \omega^{i_1 k_2 r_1}$$

In Step 2, a FT of each row of the array is computed.

This method of computing the FT is referred to as a Fast Fourier Transform (FFT) and will be used in the following section. Computing FFT requires Nr_1 operations for calculating X'' and Nr_2 operations for computing X from X'' . The total number of operations is $N(r_1 + r_2)$.

4 The FFT on the $n - SCC$ Network

In this section we describe two different algorithms for computing the FFT on an $n - SCC$ network. Both algorithms use the decomposition scheme. These two algorithms have several steps. At each step after some local computations, nodes in the adjacent rings in the i^{th} dimension exchange their data. This exchange of data is referred to as *data transfer over the i^{th} dimension*; it means that all nodes connected through edges in the i^{th} dimension, exchange their data.

<i>Step</i>	<i>no. of cycles</i>	<i>Length of cycle</i>	<i>Length of computed FFT</i>
1	$(n-2)!$	$(n-1) * n$	$n - \text{length FFT}$
2	$(n-3)! * n$	$(n-2) * (n-1)$	$(n-1) - \text{length FFT}$
3	$(n-4)! * (n-1) * n$	$(n-3) * (n-2)$	$(n-2) - \text{length FFT}$
...
i	$(n-i-1)! * (n-i+2) * \dots * n$	$(n-i) * (n-i+1)$	$(n-i+1) - \text{length FT}$
...
$n-2$	$4 * 5 * \dots * (n-1) * n$	$2 * 3$	$3 - \text{length FFT}$
$n-1$	$3 * 4 * \dots * (n-1) * n$	$1 * 2$	$2 - \text{length FFT}$

Table 1: Steps of FFT computation on an $n - SCC$.

4.1 Concurrent Pipeline Algorithm

In this algorithm for computing $(n-1)$ -dimensional FFT on an $n - SCC$, $(n-1)$ steps are required. The sequence of numbers $x(1), x(2), \dots, x(n!)$ are distributed among $n!$ rings of the $n - SCC$ network such that, node (n, R_j) , where $1 \leq j \leq n!$, holds one element of the sequence in its local memory. Step i of computation corresponds to computing the $(n-i)$ dimension of FFT.

4.1.1 Steps of Concurrent Pipeline Algorithm

An $n - SCC$ can be decomposed into $(n-i-1)!(n-i+2) * \dots * n$ disjoint cycles of $(n-i)(n-i+1)$ rings, where $1 \leq i \leq n-1$. Therefore, there are $(n-1)$ ways to partition an $n - SCC$ network. Each of $(n-1)$ steps of the FFT algorithm uses one of the partitions. Table 1 illustrates the steps of the algorithm and views of disjoint cycles of different length at different steps.

This algorithm consists of two major parts: *Computing* and *transferring data*. At Step i , a $(n-i+1)$ -length FFT is calculated using $(n-i+1)$ adjacent rings. At this step, node $(n-i+1)$ of each ring starts to compute $(n+i-1)$ -length FFT. This node is called *active node*. A special initial arrangement of the data is required in order to have the appropriate data in the adjacent positions in the cycles [4]. Step i is followed by a transfer of the data over the $(n-i+1)^{th}$ dimension. Following is the pseudo-code for this algorithm.

```

for  $i = 1$  to  $n - 1$  do
    { COMPUTING FFT }
    for all  $\frac{n!}{n-i+1}$  pipelines made by  $(n - i + 1)$  nodes which are
        adjacent on  $(n - i)(n - i + 1)$ -length cycles do
        compute  $(n - i + 1)$ -length FFT
        feedback the computed data to appropriate node in the pipeline
        multiply the computed data by an appropriate coefficient
    od
    { TRANSFERRING DATA }
for all nodes
    transfer data over  $(n - i + 1)^{th}$  dimension od

```

By using an $n - SCC$ structure, we can decrease the number of operations. Each ring contains more than one node, therefore, at the same time we can have more than one active node to activate the next step of the algorithm. Thus, it is not necessary to wait for the pipeline to complete its operations and send back its outputs. When the first output exits from the last cell of the pipeline, the result will be sent to the corresponding node. Such a node is called (i, R_j) . When those nodes which form the pipeline receive the data, they become active and start the computation of $(i - 1)$ -length FFT. Some pipelines start computing $(i - 1)$ -length FFT later, because their inputs are from the pipeline which is computing i -length FFT.

Lemma 1: The result of Concurrent Pipeline Algorithm for computing FFT on an $n - SCC$ is available after k steps, where $n(n + 1)/2 \leq k \leq n^2 - 1$.

Proof: In this algorithm, the operations of pipelines are overlapped, therefore, computed data become available in different steps, and the final data can be read from the network step by step. The first and last computed data in this algorithm are ready after $n(n + 1)/2$ and $n^2 - 1$ operations respectively.

Example 1 : Table 2 illustrates the steps of the algorithm for calculating FFT on the $4 - SCC$. At the first step, a 4-length FFT is computed. An $n - SCC$ is partitioned into two disjoint cycles of length 12. These two cycles are:

1234 2134 3124 4123 1423 2413 3412 4312 1342 2341 3241 4231, and
 3214 2314 1324 4321 3421 2431 1432 4132 3142 2143 1243 4213.

Each cycle can be defined by its initial permutation. Assume that the

<i>Step</i>	<i>Number of cycles</i>	<i>Length of cycle</i>	<i>Length of computed FFT</i>
1	2	12	$4 - \text{length}FFT$
2	4	6	$3 - \text{length}FFT$
3	12	2	$2 - \text{length}FFT$

Table 2: Steps of computing FFT on the 4 – *SCC* and views of disjoint cycles with different lengths.

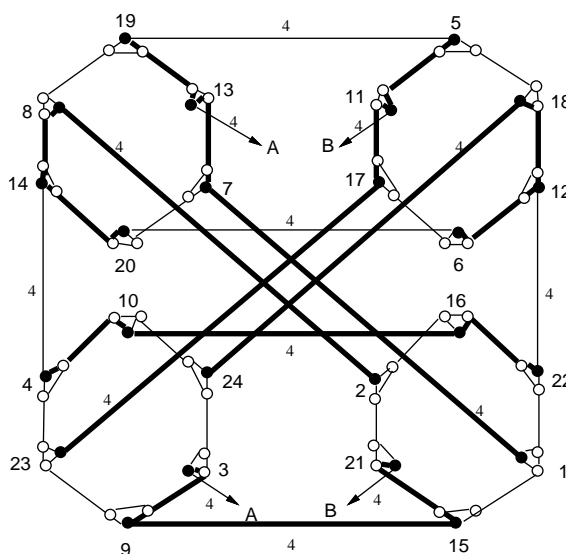


Figure 4: Processing elements (in bold) involved in computing 4-length FFT at Step 1.

initial permutation of a cycle is given, the other permutations can be generated by visiting other dimensions until the initial permutation of the cycle is reached. In these two cycles 1234 and 3214 are the initial permutations. The six pipelines of length four, which are computing a 4-length FFT, are in the path of these two cycles.

Figure 4 shows the calculation of Step 1 of the algorithm among the processing elements connected by bold lines. Each of those 4-connected rings acts as a linear array and performs the computation in a pipeline fashion. In Figure 4, node number 4 in each ring (also called the active node) specified by a black circle, performs the execution of the algorithm. The four outputs are immediately feedback to the same linear array to node number 4 in each ring,

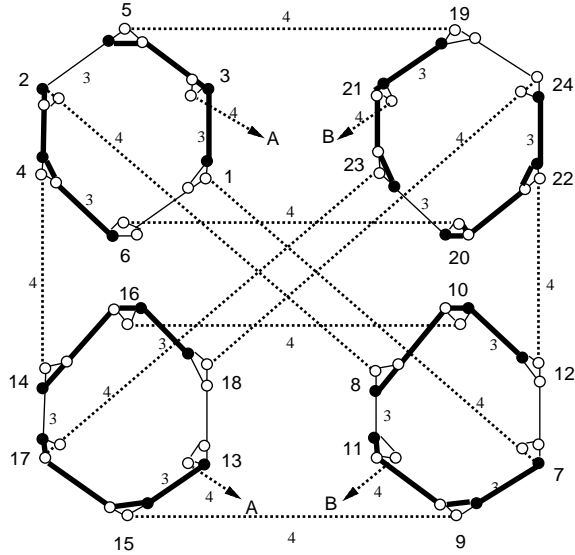


Figure 5: At Step 2, 3-length FFT are computed by the elements connected by bold lines.

which become inputs to the new linear array in the next step of the algorithm (Figure 5).

At Step 2, the $n - SCC$ is decomposed into four disjoint cycles of length six. Cycles are generated by the initial permutations visiting dimensions 2 and 3. When a node number 4 receives the result, it transfers the value to the node number 3 in the adjacent ring. Meanwhile, the pipelines illustrated in Figure 4 are computing the next output. Therefore, there are three different types of pipelines computing different steps of FFT algorithm. In the first pipeline, node 4 of each ring are calculating the 4-length FFT (Figure 4). In the second pipeline, node number 3, the active node, in each ring computes 3-length FFT (Figure 5). Third pipeline represented by the processing elements connected by bold lines (Figure 6). This pipeline computes a 2-length FFT.

At Step 3, node number 3 in each ring transfers the result to the adjacent ring connected by the 3^{rd} dimension (dotted lines in Figure 6), at this point node number 2 in each ring is responsible for executing the 2-length FFT. Figure 7 shows the final arrangement of the data at the end of Step 3. In this figure the transfer of data over the 2^{nd} dimension is represented by dotted lines.

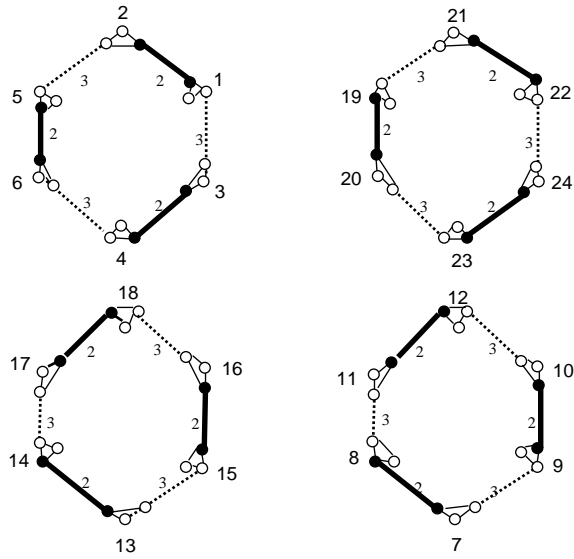


Figure 6: Processors (in bold) involved in computing 2-length FFT at Step 3.

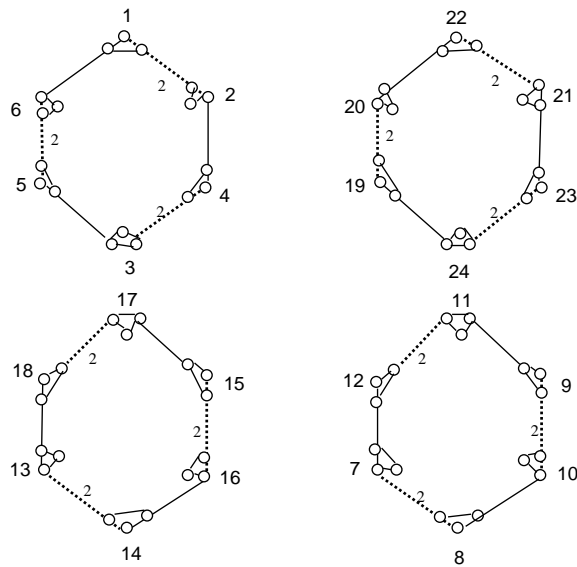


Figure 7: The final arrangement of elements.

<i>Step</i>	<i>Number of cycles</i>	<i>Length of cycle</i>	<i>Length of computed FFT</i>
1	$(n-2)!$	$(n-1) * n$	$n + 1 - \text{length FFT}$
2	$(n-3)! * n$	$(n-2) * (n-1)$	$n - \text{length FFT}$
3	<i>local ring computation</i>	<i>local ring computation</i>	$(n-1) - \text{length FFT}$
4	$(n-4)! * (n-1) * n$	$(n-3) * (n-2)$	$(n-2) - \text{length FFT}$
...
<i>i</i>	$(n-i)! * (n-i+3) * \dots * n$	$(n-i+1) * (n-i+2)$	$(n-i+2) - \text{length FT}$
...
$n-1$	$4 * 5 * \dots * (n-1) * n$	$2 * 3$	$3 - \text{length FFT}$
n	$3 * 4 * \dots * (n-1) * n$	$1 * 2$	$2 - \text{length FFT}$

Table 3: Steps of FFT computation on the $n - SCC$.

4.2 Parallel Pipeline Algorithm

This algorithm computes n -dimensional FFT on an $n - SCC$, where $N = (n-1)n! = 2 * 3 * \dots * (n-2) * (n-1) * (n-1) * n$. The sequence of numbers $x(1), x(2), \dots, x((n-1)n!)$ are distributed among $(n-1)n!$ nodes. Each node in the network holds one data of the sequence in its local memory.

For calculating the n -dimensional FFT, n computational steps are required. At Step i , the $(n-i+2)$ -length FFT is computed ($1 \leq i \leq n$).

4.2.1 Steps of Parallel Pipeline Algorithm

An $n - SCC$ can be decomposed into $(n-i-1)!(n-i+2)*\dots*n$ disjoint cycles of $(n-i)(n-i+1)$ rings ($1 \leq i \leq n-1$). Therefore, there are $(n-1)$ ways to partition an $n - SCC$ network. Each $(n-1)$ steps of the FFT algorithm considers one way of partitioning an $n - SCC$. Table 3 illustrates the steps of the algorithm and views of disjoint cycles of different lengths at different steps.

This algorithm consists of two major parts: *Computing* and *Transferring data*. At Step 1, $(n-1)n!$ nodes form $(n-1)(n-1)!$ pipelines of size n . These pipelines used to compute $(n+1)$ -length FFT. Step 2 is followed by a transfer of the computed data over the n^{th} dimension of an $n - SCC$. At this step, n -length FFT is calculated by nodes which form pipelines of size $n-1$ and are adjacent to the $(n-2)(n-3)$ -length cycles. At Step 3, $(n-1)$ nodes in each ring form a pipeline and compute a $(n-1)$ -length FFT. Similar to the first and second steps, at the i^{th} step of the algorithm, ($i > 3$), $(n-1)$ nodes in

$(n - i + 2)$ adjacent rings perform the computation of $(n + i - 2)$ -length FFT. Step i , where $i > 2$, is followed by a transfer of the data over the $(n - i + 2)^{th}$ dimension. In each transfer of data, there is a change from one dimension to the next. Following is the pseudo-code for this algorithm.

```

for  $i = 1$  to  $n$  do
    {COMPUTING FFT}
    if  $i < 3$  then
        for all  $\frac{(n-1)n!}{n-i+1}$  pipelines made by  $(n - i + 1)$  nodes which
            are adjacent on  $(n - i)(n - i + 1)$ -length cycles do
            compute  $(n - i + 2)$ -length FFT
            feedback computed data to the first node in the pipeline
            multiply the computed data by an appropriate coefficient
        od
    else if  $i > 3$  then
        for all  $\frac{(n-1)n!}{n-i+1}$  pipelines made by  $(n - i + 2)$  nodes which
            are adjacent on  $(n - i + 1)(n - i + 2)$ -length cycles do
            compute  $(n - i + 1)$ -length FFT
            feedback the computed data to the first node in the pipeline
            multiply the computed data by an appropriate coefficient
        od
    else if  $i = 3$ 
        for all  $n!$  pipelines made by  $(n - 1)$  nodes in rings do
            compute  $(n - 1)$ -length FFT
            feedback the computed data to the first node in the pipeline
            multiply the computed data by an appropriate coefficient
        od
    {TRANSFERRING DATA}
    if  $i = 1$ 
        for all nodes
            transfer the computed data over  $(n - i + 1)^{th}$  dimension
            { Not transferring data if  $i = 2$  }
    else if  $i > 2$ 
        transfer the computed data over  $(n - i + 2)^{th}$  dimension
    od

```

This algorithm consists of n steps. Pipelines at each step have the same size

<i>Step</i>	<i>no. of cycles</i>	<i>Length of cycle</i>	<i>no. of pipelines</i>	<i>Length of pipeline</i>	<i>Computed FFT</i>
1	2	12	12	4	5 – length FFT
2	4	6	24	3	4 – length FFT
3	– – –	– – –	24	3	3 – length FFT
4	12	2	36	2	2 – length FFT

Table 4: Steps of computing FFT on the 4 – SCC.

and they are computing FFT simultaneously. The total number of operations in one particular step is equal to the number of operations required for one pipeline.

Lemma 2 : The number of operations in Parallel Pipeline Algorithm is $n^2 + 2n - 4$.

Proof : The number of operations of a pipeline of size n with n inputs is $2n - 1$. At Steps 1 and 2 the size of the pipelines are n and $n - 1$ respectively, therefore, the number of operations is :

$$(2n - 1) + (2n - 3) = 4n - 4.$$

At Step i , ($3 \leq i \leq n$), the size of pipeline is $(n - i + 2)$. Therefore, the total number operations is:

$$4n - 4 + \sum_{i=3}^n 2(n - i + 2) - 1 = n^2 + 2n - 4$$

Example 2: Table 4 represents the computation steps of the algorithm on the 4 – SCC. An $n - SCC$ is partitioned into two disjoint cycles of length 12:

1234 2134 3124 4123 1423 2413 3412 4312 1342 2341 3241 4231, and

3214 2314 1324 4321 3421 2431 1432 4132 3142 2143 1243 4213.

In these two cycles 1234 and 3214 are the initial permutations, and the rest of the permutations are generated by visiting dimensions 2, 3, and 4 repeatedly until the initial permutations are repeated. In the path of these two cycles there are six pipelines of length four and they are used to compute a 5-length FFT.

Figure 8 shows the calculation of Step 1 by processing elements connected by bold lines. Each bold line represents three pipelines. Each of those 4-connected rings acts as a linear array and performs the computation in a pipeline fashion. Node j in each ring ($2 \leq j \leq 4$) defines a pipeline and performs the execution of the algorithm. The four outputs are immediately

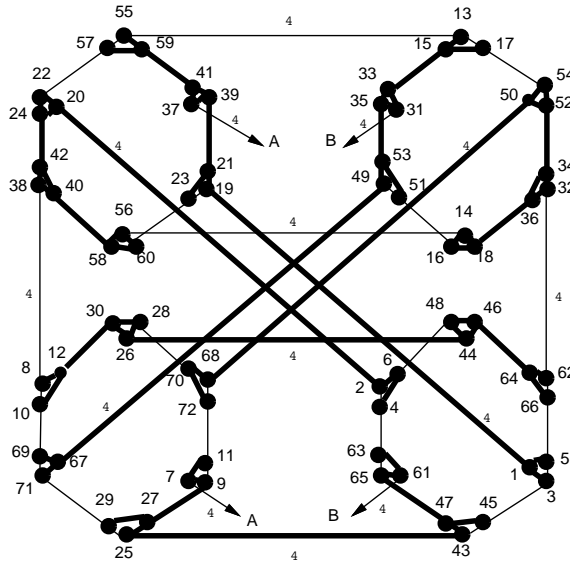


Figure 8: Processing elements involved in computing 5-length FFT (Step 1).

feedback to the starting node (Figure 9). The received output is transferred to the node j of the adjacent ring connected by the 4th dimension (dotted lines in Figure 9). At Step 2, processing elements (connected by bold lines) compute a 4-length FFT. At Step 3, nodes in each ring form a pipeline and compute a 3-length FFT (Figure 10). Figure 11 depicts Step 4 of the algorithm. In this figure elements connected by bold lines, compute a 2-length FFT. Figure 12 shows the final arrangement of the elements at the end of Step 4.

5 Conclusion

Two FFT algorithms on an $n - SCC$ are considered. Table 5 represents the comparison among different algorithms for the FFT [4, 7, 9, 6]. In the first five algorithms, the number of input elements is $n!$. For the algorithms presented in this paper, although this number is increased by a factor of $(n-1)$, still the number of operations is in the order of $O(n^2)$.

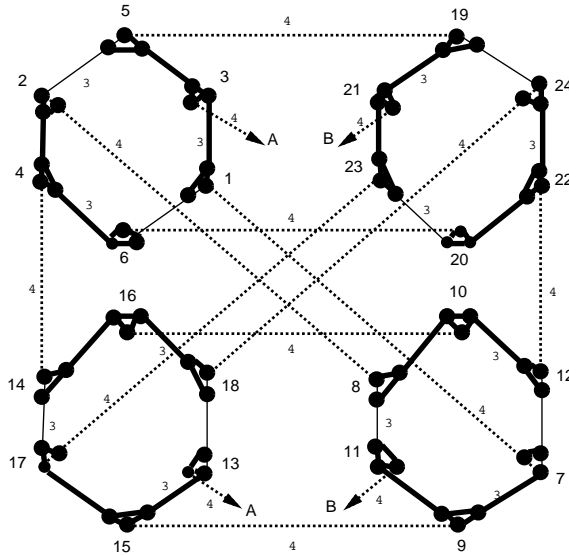


Figure 9: A 4-length FFT are computed by the elements connected by bold lines (Step 2).

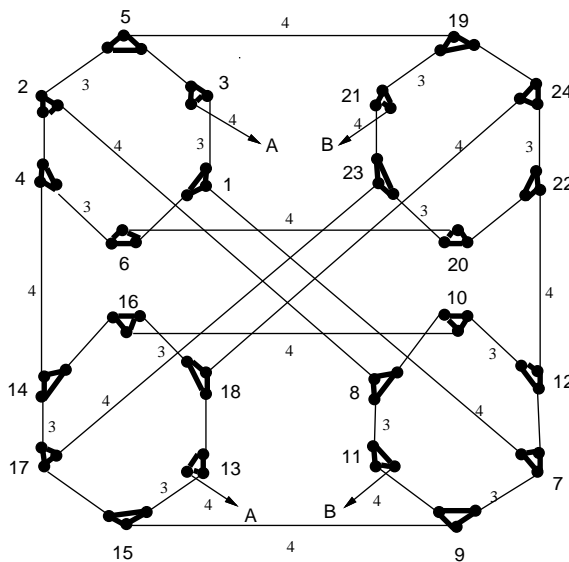


Figure 10: A 3-length FFT are computed by the elements connected by bold lines (Step 3).

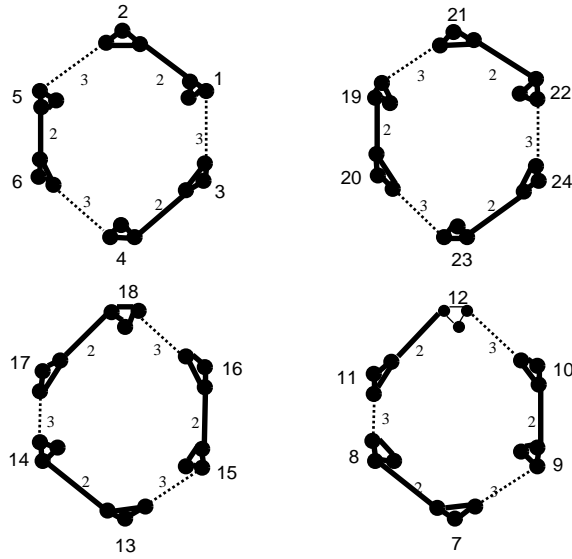


Figure 11: Processors (in bold) involved in computing 2-length FFT at the Step 4.

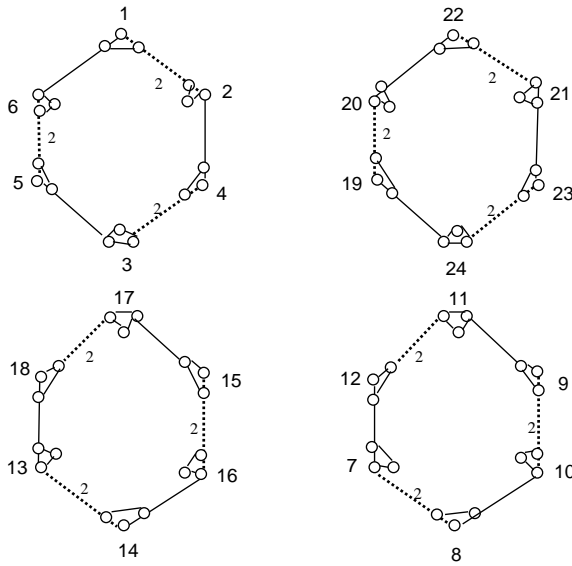


Figure 12: The final arrangement of elements.

<i>Algorithm</i>	<i>no. of input sequence</i>	<i>no. of processors</i>	<i>no. of operations</i>
<i>sequential</i>	$n!$	1	$n(n+1)!/2$
<i>star</i>	$n!$	$n!$	$n^2 - 1$
<i>hypercube</i>	$n!$	$n!$	$O(\log n!) = O(n \log n)$
<i>perfect shuffle</i>	$n!$	$n!$	$O(\log n!) = O(n \log n)$
<i>CCC</i>	$n!$	$O(n! \log n!)$	$O(\log n!) = O(n \log n)$
<i>SCC(Concurrent)</i>	$n!$	$(n-1)n!$	$n(n-1)/2$
<i>SCC(Parallel)</i>	$(n-1)n!$	$(n-1)n!$	$n^2 + 2n - 4$

Table 5: Comparison of different algorithms for computing FFT.

References

- [1] S. B. Akers, D. Harel, and B. Krishnamurthy, "The Star Graph: An Attractive Alternative to the n-cube," *International Conf. on Parallel Processing*, 1987 August, pp. 393-400.
- [2] S. B. Akers, and B. Krishnamurthy, "A Group Theoretic Model for Symmetrical Interconnection Networks," *IEEE Trans. Computers*, vol. 38, no. 4, 1989 April, pp. 555-566.
- [3] J. W. Cooley and J. W. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series," *Math. of Comput.*, Vol 19, April 1965, pp. 297-301.
- [4] P. Fragopoulou and S. G. Akl, "A Parallel Algorithm for Computing Fourier Transforms on the Star Graph," *International Conf. on Parallel Processing*, 1991, pp. 100-106.
- [5] S. Latifi, M. Azevedo, and N. Bagherzadeh, "The Star-Connected Cycles: A fixed-degree interconnection network for parallel processing," *International Conf. on Parallel Processing*, 1993.
- [6] D. S. Parker, "Notes on Shuffle/Exchange-Type Switching Networks," *IEEE Trans. Computers*, vol. 29, no. 3, 1980 March, pp. 213-223.
- [7] F. P. Preparata, and J. Vuillemin, "The Cube-Connected Cycles," *Communications of the ACM*, vol. 24, no. 5, 1981 May, pp. 300-309.

- [8] K. Qui, H. Meijer, and S. G. Akl, "Decomposing a Star Graph into Disjoint Cycles," *Information Processing Letters*, vol. 39, no. 3, 1991 August, pp. 125-129.
- [9] H. S. Stone, "Parallel Processing with the Perfect Shuffle," *IEEE Trans. Computers*, vol. 20, no. 2, 1971 February, pp. 153-161.
- [10] C. Nian Zhang and D. Y. Y. Yun, "Multi-Dimensional Systolic Networks for Discrete Fourier Transform," *The 11th Annual Int. Symp. on Comp. Arch.*, June 5-7, 1984, Ann Arbor, Michigan, pp. 215-222.

A Appendix

The FT of a sequence of numbers, $x(1), x(2), \dots, x(N)$, is computed by the following expression:

$$X(i) = \sum_{j=1}^N x(j)\omega^{(j-1)(i-1)}$$

for $1 \leq i \leq N$ where $\omega = e^{2\pi k/N}$ with $k = \sqrt{-1}$

The N-point FT can be viewed as a polynomial

$$x(N)A^{N-1} + x(N-1)A^{N-2} + \dots + x(2)A + x(1)$$

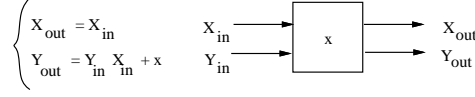
at $A = 1, \omega, \omega^2, \dots, \omega^{N-1}$

By using Horner's rule, this formula can be written as:

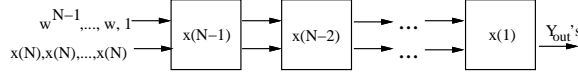
$$(\dots(x(N)A + x(N-1))A + x(N-2))A + \dots + x(2))A + x(1) \tag{1}$$

Formula (1) can be computed by the Linear Systolic Array Algorithm. This algorithm uses basic cells. A full array consists of $N - 1$ basic cells.

Figure 13 illustrates the structures of the basic cell and the Linear Systolic Array. The Y_{out} 's from the right-most cell are the FT of the input sequence. These values are computed in a pipeline fashion in $O(N)$ multiply-add steps.



(a) Basic cell for Linear Systolic Array.



(b) Linear Systolic Array.

Figure 13: The structure of basic cell and Linear Systolic Array.

Assume N , the number of input elements, is a composite number with only two factors r_1 and r_2 , $N = r_1 * r_2$. The FFT of $X(1), X(2), \dots, X(N)$ can be obtained as a two-dimensional FFT by arranging the N input elements in row-major order as an $r_1 * r_2$ array, $x(k_1, k_2)$ refers to one or more elements in row k_2 and column k_1 , where $k_1 = 1, 2, \dots, r_1$ and $k_2 = 1, 2, \dots, r_2$. The FFT can be written as:

$$\begin{aligned}
 X(i_1, i_2) &= \sum_{k_2=1}^{r_2} \sum_{k_1=1}^{r_1} x(k_1, k_2) \omega^{i[(k_1-1)r_2+k_2]} \\
 &= \sum_{k_2=1}^{r_2} \sum_{k_1=1}^{r_1} x(k_1, k_2) \omega^{i(k_1-1)r_2} \omega^{ik_2} \quad (2)
 \end{aligned}$$

$$i = (i_1 - 1)r_1 + i_2 \quad \text{where } i_2 = 1, 2, \dots, r_1 \quad \text{and} \quad i_1 = 1, 2, \dots, r_2$$

Since $\omega^{(i_1-1)(k_1-1)r_1r_2} = \omega^{(i_1-1)(k_1-1)N} = (e^{2\pi i})^{(i_1-1)(k_1-1)} = 1$, Formula (2) can be written as:

$$X(i_1, i_2) = \sum_{k_2=1}^{r_2} \omega^{i_1 k_2 r_1} \omega^{i_2 k_2} \sum_{k_1=1}^{r_1} x(k_1, k_2) \omega^{i_2 (k_1-1) r_2}$$

$X(i_1, i_2)$ is computed in three steps. This method of computing FT is referred to as Fast Fourier Transform (FFT). Formula (2) can be executed by a Linear Systolic Array using basic cells [10].