THE UNIVERSALITY OF VARIOUS TYPES OF SIMD MACHINE INTERCONNECTION NETWORKS

Howard Jay Siegel
School of Electrical Engineering
Purdue University
West Lafayette, Indiana 47907

Abstract

SIMD machine architects must choose an interconnection network to provide interprocessor communication. The universality of a network is its ability to simulate arbitrary interconnections of the processing elements. We examine the universality of five particular networks which cover the types used in the Illiac IV, STARAN, Omen, SIMDA, and RAP machines. They also cover the types discussed by Feng, Lang, Lawrie, Orcutt, Siegel, and Stone. We give $O((\log_2 N)^2)$ algorithms, where N is the number of processing elements, for the Perfect Shuffle, PM2I, WPM2I, and Cube networks to simulate arbitrary interconnections (Orcutt has given an $O(N^{1/2}\log_2 N)$ algorithm for the Illiac network). We analyze Batcher's bitonic sorting method and show how each network can implement it on an SIMD machine. We discuss how sorting destination tags is equivalent to simulating interconnections.

I. Introduction

The feasibility of SIMD (single instruction stream-multiple data stream) machines has greatly increased with the microprocessor revolution. Machine architects must choose an interconnection network to implement when designing an SIMD computer system. The network must provide interprocessor communication facilities for the processors (or microprocessors) of which the system is composed. The number of distinct interconnections that the system architect can include is limited by cost considerations. The architect must therefore choose a small set of interconnections and use them to simulate the actions of the interconnections that were not included in the system. In this paper we consider the universality of various types of interconnection networks; that is, the ability of different types of networks to simulate arbitrary interconnections. We show an upper bound on the time required by various types of networks to simulate arbitrary interconnections by presenting algorithms to perform the simulations. The simulation algorithms are based on sorting algorithms, in particular, bitonic sorting. If a destination tag is paired with the data to be moved then the interconnection network can sort the destination tags, moving the associated data long with them. Thus, by sorting the destination tags, the interconnection represented by that set of tags will be simulated.

Orcutt gives an algorithm to show the universality of the Illiac IV interconnection network in reference 21. Stone shows that the shuffle interconnection connected to compare-exchange modules can implement bitonic sorting in reference 27. We show how various types of interconnection networks that have been discussed in the literature can simulate arbitrary interconnections efficiently. In this paper we discuss five particular interconnection networks, including the Perfect Shuffle and the network used in the Illiac IV system. These networks cover the types used in the Illiac IV[1,6], STARAN[4], Omen[13], SIMDA[28], and RAP[7] machines. They also cover the types discussed by Feng[9,10], Lang[16], Lawrie[18], Orcutt[21], Siegel[23,24], and Stone[27]. The sorting algorithms presented here are based on Batcher's bitonic sorting network[2].

Algorithms to perform the sorting operation are presented and the time complexity of each algorithm is analyzed. A model developed in reference 24 is used to characterize SIMD machines. For the sorting algorithms very few assumptions about the attributes of the SIMD machines are made. The time complexity results hold for all SIMD architectures which satisfy the few assumptions. In addition, the algorithms can be implemented directly for such machines. This, therefore, provides the system designer with information about one facet of the abilities of various types of interconnection networks.

In section II we define our model of SIMD machines and state any assumptions made about SIMD machines. The various interconnection networks are defined in section III. Batcher's bitonic sorting method is reviewed and analyzed in section IV. In section V we give the algorithms for the Perfect Shuffle, PM2I, WPM2I, and Cube interconnection networks to simulate arbitrary interconnections in $O((\log_2 N)^2)$ time, where N is the number of processing elements in the SIMD machine.

II. The Model

An SIMD (single instruction stream-multiple data stream[11]) machine is a computer system consisting of a control unit, a set of processing elements, and an interconnection network. The control unit broadcasts instructions to the processing elements and all active processing elements will execute the same instruction at the same time. Thus, there is a single instruction stream. Each active processing element will execute the instructions on data in its own memory. Thus, there is a multiple data stream. The interconnection network allows processing elements to communicate with each other. These machines are designed to exploit the parallelism of tasks such as vector and matrix operations. They are also referred to as array machines.

One way to view the physical structure of an SIMD machine is as a set of N processing elements (where each processing element consists of a processor with its own memory), interconnected by a network, and fed instructions by a control unit. This type of configuration is shown in Figure 1. The network connects each processing element to some subset of the other processing elements. A transfer instruction causes data to be moved from each processing element to one of the processing elements to which the element is connected by the network.

An alternative structure is to position the network between the processors and the memories. This type of configuration is shown in Figure 2.

This work was supported by NSF Grant DCR74-21939 at Princeton University and by the School of Electrical Engineering at Purdue University.

The processing element to processing element configuration is used in our model. However, the results presented here are also valid for the processor to memory type of structure, assuming that processor$_i$ can directly access memory$_i$, $0 \leq i < N$.

Our model of an SIMD machine consists of four parts: processing elements, interconnection functions, machine instructions, and masking schemes. By specifying these four components a particular SIMD machine architecture can be described. The model was designed to reflect all of the flexibility of actual SIMD computer systems. This model was first presented in reference 24.

Each processing element (PE) is a processor together with its own memory, a set of fast access registers, and a data transfer register (DTR). We assume there are at least two fast access registers, which will be referred to as A and B. The DTR of each PE is connected to the DTR's of other PE's via the interconnection network. When data is transferred among processing elements it is the contents of the DTR of each PE that is transferred.

There are N PE's, each assigned an address from 0 to N-1. We assume that N equals 2^m; that is, $\log N = m$. We also assume that PE$_i$ has a register ADDRESS that has the value i, $0 \leq i < N$. We define ADDRESS(j) to be the jth bit of ADDRESS.

Each PE is either in the active or the inactive mode. If a PE is active it executes the instructions broadcast to it by the control unit. If a PE is inactive it will not execute the instructions broadcast to it. We use enabled and disabled as synonyms for active and inactive, respectively.

An interconnection network is a set of interconnection functions, each a bijection on the set of PE addresses. When an interconnection function f is applied, PE$_i$ copies the contents of its DTR into the DTR of PE$_{f(i)}$. This occurs for all i simultaneously, for $0 \leq i < N$ and PE$_i$ active. In section III five particular interconnection networks will be defined.

Note that an inactive PE may receive data from another PE if an interconnection function is executed, but the inactive PE cannot send data. For example, if we apply the interconnection function $f(i) = i+1 \bmod N$, $0 \leq i < N$, when PE$_0$ is active and PE$_1$ is inactive, the data in the DTR of PE$_1$ is overwritten by the data transferred from PE$_0$. The data that was originally in the DTR of PE$_1$ is lost, unless a copy was saved. Thus, whenever data is transferred into disabled PE's, care must be taken to preserve copies of any necessary data that might be overwritten.

The machine instructions are those operations that each processor can perform on data in its individual memory or registers. We assume there is a separate control unit (CU) computer which stores programs and broadcasts instructions and data. All enabled PE's execute the same instruction at the same time, but on possibly different data. The notation $x \leftarrow y$ means the contents of register y are copied into register x. The abreviated notation $x \leftrightarrow y$ indicates that two registers switch their contents by using a third register as temporary storage.

A masking scheme is a method for determining which PE's will be active at a given point in time. Data conditional masks are the implicit result of executing conditional statements in an SIMD machine environment. This type of masking is used in such machines as the Illiac IV[1,6,20,26] and PEPE[29].

When a conditional statement is executed, each PE may be executing it with different data, so the outcome of the conditional may differ from one PE to the next. Thus, as a result of the conditional, each PE will set an internal flag so that it will be active for either the "then" or the "else," but not both. The execution of the "else" statements must follow the "then" statements; that is, they cannot be executed simultaneously. For example, as a result of executing the statement:

if A > B then C ← A
else C ← B

each PE will load its C register with the maximum of its A and B registers; that is, some PE's will execute "C ← A," and then the rest will execute "C ← B."

Formally, an SIMD machine can be represented as the 4-tuple (N,F,I,M), where:

(1) N is a positive integer, representing the number of processing elements in the machine;

(2) F is the set of interconnection functions, where each function is a bijection on the set {0,1,...N-1};

(3) I is the set of machine instructions, instructions that are executed by each active PE and act on data within that PE;

(4) M is the set of masking schemes, where each mask enables certain PE's and disables the others.

By specifying N, F, I, and M, a particular SIMD machine architecture can be modeled.

The only assumptions made about the SIMD machine model in this paper are:

(1) $N = 2^m$;

(2) F will vary:

(3) I contains instructions to (a) move data among the DTR and fast access registers of the same PE, (b) perform the "or," "and," and "exclusive-or" boolean operations, and (c) perform "greater than," "less than," and "equal" data comparisons; and

(4) M = {data conditional masks}.

III. The Interconnection Networks

The five networks we will discuss are the Perfect Shuffle, the Illiac, the Cube, the Plus-Minus 2^i, and the Wrap-around Plus-Minus 2^i. These networks cover the types used in the Illiac IV[1,6], STARAN[4], Omen[13], SIMDA[28], and RAP[7] machines. They also cover the types discussed by Feng[9,10], Siegel[23,24], Orcutt[21], Lawrie[18], Stone[27], and Lang[16].

The following notation will be used: let the binary representation of a PE address be $p_{m-1}p_{m-2}\cdots p_1p_0$, let $\bar{p}_i$ be the complement of p_i, and let the integer n be the square root of N. Recall that $N = 2^m$ is the number of PE's.

(1) Perfect Shuffle (PS). This network consists of a shuffle function and an exchange function. The shuffle is defined by:

$$s(p_{m-1}p_{m-2}\cdots p_1p_0) = p_{m-2}p_{m-3}\cdots p_1p_0p_{m-1}$$

and the exchange is defined by:

$$e(p_{m-1}p_{m-2}\cdots p_1p_0) = p_{m-1}p_{m-2}\cdots p_1\bar{p}_0.$$

For example, $s(3) = 6$ and $e(6) = 7$, for $N \geq 8$.

The shuffle function is discussed in references 12 and 14. The usefulness of this network is demonstrated by Stone and Lang[16,17,27]. The network is further analyzed in references 23 and 24. Lawrie's Omega network is shown to be a series of Perfect Shuffles[18,19]. The Perfect Shuffle is also included in the networks of the Omen and RAP systems[7,13].

(2) Illiac. This network has four functions defined as follows (recall n is the square root of N):

$$I_{+1}(x) = x+1 \bmod N$$
$$I_{-1}(x) = x-1 \bmod N$$
$$I_{+n}(x) = x+n \bmod N$$
$$I_{-n}(x) = x-n \bmod N$$

For example, if $N = 16$, $I_{+n}(0) = 4$. Note that when we discuss the Illiac we shall assume m is even, that is, $n = 2^{m/2}$ is an integer.

The Illiac is the network implemented in the Illiac IV system. The ability of this network to perform various tasks, as well as other features of this network, are discussed in references 1, 6, 21, 23, 24 and 25.

(3) Cube. This network consists of m functions defined by:

$$c_i(p_{m-1}\cdots p_{i+1}p_ip_{i-1}\cdots p_0) = p_{m-1}\cdots p_{i+1}\bar{p}_ip_{i-1}\cdots p_0$$

for $0 \leq i < m$. The Cube function c_i complements the ith bit position of the addresses. For example, $c_2(7) = 3$. When the PE addresses are considered as the corners of an m-dimensional cube this network connects each PE to its m neighbors.

The interconnection network in the STARAN is a wired series of Cube functions[4]. In reference 3 and 5 the applicability of the Cube type of network to practical problems is discussed. The Cube network is analyzed in references 23 and 24. A version of the Cube was used as part of a parallel machine simulation in reference 22.

(4) Plus-Minus 2^i (PM2I). This network consists of 2m functions defined by:

$$t_{+i}(j) = j+2^i \bmod N$$
$$t_{-i}(j) = j-2^i \bmod N$$

for $0 \leq i < m$. For example, $t_{+1}(2) = 4$ if $N > 4$.

To implement data manipulating functions Feng recommended a network which is a wired series of PM2I interconnection functions[9,10]. A network similar to the PM2I is also included in the network of the Omen computer[13]. The t_{+0} and t_{-0} functions are included in the RAP network[7]. The concept underlying the SIMDA array machine's interconnection network is the same as that of the PM2I, but the SIMDA uses hardware multiplexing to combine the effects of sets of PM2I functions[28]. The various data manipulations that the PM2I can perform, and other features of this network are discussed in references 9, 10, 23 and 24.

(5) Wrap-around Plus-Minus 2^i (WPM2I). This network consists of 2m functions defined by:

$$w_{+i}(p_{m-1}\cdots p_i\cdots p_0) = q_{m-1}\cdots q_i\cdots q_0,$$

where

$$q_{i-1}\cdots q_0q_{m-1}\cdots q_{i+1}q_i = (p_{i-1}\cdots p_0p_{m-1}\cdots p_{i+1}p_i) + 1 \bmod N,$$

and

$$w_{-i}(p_{m-1}\cdots p_i\cdots p_0) = q_{m-1}\cdots q_i\cdots q_0,$$

where

$$q_{i-1}\cdots q_0q_{m-1}\cdots q_{i+1}q_i = (p_{i-1}\cdots p_0p_{m-1}\cdots p_{i+1}p_i) - 1 \bmod N,$$

for $0 \leq i < m$. WPM2I is like PM2I, except any "carry" or "borrow" will "wrap-around" through the p_{i+1} bit position. For example, if N=8, m=3, and we write the PE addresses in binary then: $w_{-1}(001) = 110$, whereas $t_{-1}(001) = 111$. Thus, the function w_{+i} can be thought of as adding 2^i to the PE addresses, with any "carry" wrapping around up to and including p_{i-1}. Note that a carry cannot affect p_i. The function w_{-i} can be described in a similar way.

The WPM2I network, was introduced in reference 23. It has the ability to simulate any other interconnection function when the networks are treated as sets of permutations on the integers from 0 to N-1. In terms of group theory, WPM2I can generate the entire group of permutations on N elements. Of the five networks presented here, only WPM2I has this ability[23]. The WPM2I network is also analyzed in reference 24.

IV. Bitonic Sorting

In this section we review Batcher's bitonic sorting method. We show which pairs of elements should be compared at each step of the process and which comparisons should produce ascending or descending sequences. This information is needed to construct the algorithms to implement bitonic sorting on SIMD machines which will be presented in section V.

Batcher's bitonic sorting algorithm was originally designed as a network of comparator modules[2]. Each comparator has two inputs and two outputs. The comparator routes the maximum input to the lower output port and the minimum to the upper port. This is shown in Figure 3. We shall first review how Batcher uses these comparators to sort "bitonic" sequences and then show how he uses "bitonic" sorters to sort arbitrary sequences.

A sequence of numbers $x_0, x_1, \ldots x_{r-1}$ is a bitonic sequence if the condition $x_0 \geq \ldots \geq x_k \leq \ldots \leq x_{r-1}$, for some k, $0 \leq k < r$, holds, or if $x_0, x_1, \ldots x_{r-1}$ can be shifted cyclically so that the condition holds. A bitonic sequence is the concatenation and cyclic shifts of two monotonic sequences, one increasing and the other decreasing.

A single comparator is a sorter for bitonic sequences of size 2. To construct larger bitonic sequence sorters the iterative technique shown in Figure 4 is used. Thus, a bitonic sequence of size 2^j would require j ranks of comparators. For example, Figure 5 shows a 4-element bitonic sequence sorter.

The following theorem guarantees the correctness of this method.

Theorem 1: (Batcher) if $x_0, x_1, \ldots x_{r-1}$ is a bitonic sequence then the sequences

$$\min(x_0, x_{r/2}), \min(x_1, x_{(r/2)+1}), \ldots \min(x_{(r/2)-1}, x_{r-1})$$

and

$$\max(x_0, x_{r/2}), \max(x_1, x_{(r/2)+1}), \ldots \max(x_{(r/2)-1}, x_{r-1})$$

are both bitonic and no element of the former sequence is greater than any element of the later sequence.

Proof: See references 2 or 27. □

Batcher shows that bitonic sorters can be used to sort arbitrary sequences into ascending order. To do this two types of comparators will be needed. We will refer to the type of comparator shown in Figure 3 as an ascending comparator, since it sends the maximum input value to the lower output port. We will also use descending comparators, which send the maximum input value to the upper output port. These are shown in Figure 6.

To construct an ascending order sorter for an arbitrary sequence of $r = 2^l$ elements use the following recursive method:

(1) sort the first 2^{l-1} elements using a 2^{l-1}-element sorter to produce an ascending sequence

(2) change all ascending comparators to descending comparators, and vice versa, in a 2^{l-1}-element ascending order sorter to create a 2^{l-1}-element sorter that produces a descending sequence and use this to sort the second 2^{l-1}-elements;

(3) sort the resulting bitonic sequence of 2^l elements with a 2^l-element bitonic sorter;

(4) when $l=1$ use a single ascending comparator to produce an ascending sequence and a single descending comparator to produce a descending sequence.

For example, consider the bitonic sorter for an arbitrary sequence of 8 elements which is shown in Figure 7. In general, to sort N elements, where $m = \log_2 N$:

$$1 + 2 + 3 + \ldots + m = (1/2)m(m+1)$$

ranks of comparators will be needed.

Let us view the interior of the comparator module and the labels of the input and output as shown in Figure 8. We want to simulate bitonic sorters with SIMD machines and this labeling convention allows us to find the addresses of the pairs of PE's whose data are to be compared. The input labels correspond to the address of the PE's from which the data to be compared is taken; the output numbers indicate which PE receives the maximum value and which PE receives the minimum value. Both ascending and descending comparators can be implemented in this manner; the internal links perform the appropriate exchanges of input values for the type of comparator being implemented. The action of a single rank of N/2 comparators corresponds to N/2 PE's performing N/2 comparisons simultaneously.

The following theorem, based on an observation by Stone[27], formally proves the correctness of a method for determining the addresses of pairs of PE's whose data will be compared during each step of the bitonic sequence sorting process.

Theorem 2: In an N-element bitonic sequence sorter, if we label the inputs $x_0, x_1, \ldots x_{N-1}$, then the element labeled x_i is compared to the element x_j on rank k and only if the bit representations of i and j differ only in their kth bit positions.

Proof: We shall prove this by induction on the $\log_2$ of the number of inputs, using the labeling convention shown in Figure 8.

Basis: Bitonic sequence sorter of size 2^1. Consider Figure 9. There is only one rank of comparators, the 0th. The labels 0 and 1 differ in their 0th bit position.

Induction step: Assume true for bitonic sequence sorters of size $2^i = N/2$, show true for bitonic sequence sorters of size $2^{i+1} = N$. Consider Figure 10. The ith rank of comparators, that is, the input comparators, compare inputs j and $j+2^i$, where $0 \le j < 2^i$. The inputs to the next rank are in linear order, where the next rank is the beginning of two N/2-element bitonic sequence sorters. The inputs to each of the two N/2-element sorters are of the form $x_0, x_1, \ldots x_{(N/2)-1}$, where subscripts are mod N/2. From the induction hypothesis we know that the N/2-element sorters have the desired property. The fact that we are considering the subscripts mod (N/2) does not violate our induction hypothesis, since within each N/2-element bitonic sequence sorter the ith bit of the label scripts are the same.

As we have observed earlier, a bitonic sorting network for arbitrary sequences of N elements first acts as N/2 2-element bitonic sequence sorters, then as N/4 4-element bitonic sequence sorters, etc. Let us call the set of ranks of comparators that are acting as 2^j-element bitonic sequence sorters the jth stage of the network. Let $x_{m-1} \ldots x_1 x_0$ be the binary representation of the smaller of the labels of the two inputs to an arbitrary comparator. Let ⊕ be the exclusive-or logical operator. The following theorem, based on an observation by Stone[27], proves the correctness of a method for determining which comparators should be ascending and which should be descending to sort an arbitrary sequence.

Theorem 3: In the jth stage, $1 \le j < m$, of an N-element bitonic sorting network for arbitrary sequences a comparator is ascending if and only if $x_{m-1} \oplus \ldots \oplus x_{j+1} \oplus x_j = 0$. For the mth stage all comparators are ascending.

Proof: Proof by induction on m.

Basis: m=2 (N=4). Figure 11 shows a 4-element sorter as produced by construction method which was described earlier. All input labels in the figure are in binary. For stage m=2 all comparators are ascending. For stage 1 the only comparators that are ascending are those for which the x_1 position of the smaller label is a 0.

Induction step: Assume true for $N=2^{i-1}$. Show true for 2^i. Since stage m=i is to be a sorter for a bitonic sequence, all comparators are ascending. All the comparators in the 2^{i-1} arbitrary element

sorter portion which produces the ascending sequence should be as they are in the case of sorting 2^{i-1} elements. Since the ith bit on the input labels for all of these comparators is 0 the result of the exclusive-or operation will be the same as it was for the low-order i-1 bits. The 2^{i-1}-element sorter that produces the descending sequence should be identical to the ascending 2^{i-1}-element sorter, except that the descending and ascending comparators must be switched. But for each of these comparators the Ith bit on the input label will be 1, so the exclusive-or operation will give the complement of the result obtained when only the low-order i-1 bits are used. □

These theorems are the basis for the SIMD machine sorting algorithms to be presented in the next section.

V. Implementing Bitonic Sorting with SIMD Machines

In this section we shall show how the Perfect Shuffle, the Cube, the PM2I, and the WPM2I networks can sort N elements in $O(m^2)$ time, and how the Illiac network can sort in O(nm) time. All of the sorting algorithms are based on Batcher's bitonic sorting algorithm[2,15,27]. The ability of an interconnection network to sort is important since that implies the ability to simulate arbitrary interconnection functions. If interconnection function f is to be simulated then the data in PE_i that is to be transferred is paired with a destination tag of f(i), $0 \leq i < N$. The network then sorts the destination tags, moving the tag and its associated data together as a pair. When the sorting is completed the data that was orlginally in the DTR of PE_i should be in the DTR of $PE_{f(i)}$, $0 \leq i < N$. All of the algorithms presented in this section could be directly implemented on any system which meets the assumptions we make about SIMD machines. Without loss of generality, we simplify our algorithms by letting the data in the DTR's be the destination tags themselves.

Consider using the Cube network to simulate a sorter for bitonic sequences on an SIMD machine. From Theorem 2 we know that the kth rank of comparators, corresponding to the kth set of comparisons, should compare inputs that differ only in their kth bit position. To implement this the "c_k" function is used.

The following lemma shows how the Cube network can sort bitonic sequences. The PE's whose kth bit is 0 act as the comparators.

Lemma 1: The Cube network can sort bitonic sequences of N elements in m steps.

Proof: The following algorithm directly implements a bitonic sequence sorter constructed of ascending comparators.

```
    begin
(S1) for i=m-1 step -1 until 0 do
        begin
(S2)    if ADDRESS(i) = 0 then A ← DTR
(S3)    if ADDRESS(i) = 1 then c_i
(S4)    if DTR < A and ADDRESS(i) = 0 then A ↔ DTR
(S5)    if ADDRESS(i) = 0 then c_i
(S6)    if ADDRESS(i) = 0 then DTR ← A
        end
    end
```

To implement a sorter for arbitrary sequences based on the bitonic sorting algorithm the SIMD machine must implement both ascending and descending comparators. To determine which PE's should act as ascending and which as descending comparators we use the results of Theorem 3. The next lemma gives an algorithm for implementing such a sorter using the Cube network.

Lemma 2: The Cube network can sort arbitrary sequences of N elements using m(m+1) interprocessor data transfers.

Proof: The following algorithm implements the bitonic sorting algorithm for arbitrary sequences as was discussed in the previous section.

```
    begin
(S1) for j=1 until m do
        begin
(S2)    for i=j-1 step-1 until 0 do
(S3)        begin
(S3)        if ADDRESS(i) = 0 then A ← DTR
(S4)        if ADDRESS(i) = 1 then c_i
(S5)        if ((j=m) or ((ADDRESS(m-1) ⊕ ...
                ⊕ ADDRESS(j+1) ⊕ ADDRESS(j)) = 0))
(S6)            then if DTR < A and ADDRESS(i) = 0
                        then A ↔ DTR
(S7)            else if DTR > A and ADDRESS(i) = 0
                        then A ↔ DTR
(S8)        if ADDRESS(i) = 0 then c_i
(S9)        if ADDRESS(i) = 0 then DTR ← A
            end
        end
    end
```

This algorithm executes m(m+1) Cube functions. S1 sets the variable j such that we first implement bitonic sequence sorters of size 2, then 4, then 8, ..., then N. S2 through S9 implement the bitonic sequence sorters. S5 is used to determine which type of comparator each PE should simulate. Thus, the algorithm implements an N-element bitonic sorting network for arbitrary sequences as described in the recursive construction in the previous section. Note that if j=m in S5, then the remaining portion of the boolean expression controlling the conditional is ignored. Also note that as an alternative to computing the exclusive-or in S5, its value may be computed immediately before S2, outside the scope of the S2 to S9 loop, and stored in a variable which could be used in S5. Another alternative is for each PE to precompute the values computed in S5, store them in a bit vector of size m, and access that vector whenever sorting is to be performed. If either of these alternatives is used the entire algorithm requires only a constant times m(m+1) operations. □

The algorithm in the previous lemma can sort $2^{m'}$ elements, where m' < m, by changing m to m' in S1. Thus, the algorithm for $2^{m'}$ elements would require only m'(m'+1) interprocessor data transfers.

The next two lemmas describe how the PM2I and WPM2I networks can be used to implement the bitonic sorting algorithm for arbitrary sequences.

Lemma 3: The PM2I network can sort arbitrary sequences of N elements using m(m+1) interprocessor data transfers.

Proof: Consider the algorithm presented in Lemma 2.

Replace "if ADDRESS(i) = 0 then c_i"
with "if ADDRESS(i) = 0 then t_{+i}"
and replace "if ADDRESS(i) = 1 then c_i"
with "if ADDRESS(i) = 1 then t_{-i}". □

Lemma 4: The WPM2I network can sort arbitrary sequences of N elements using m(m+1) interprocessor data transfers.

Proof: Consider the algorithm presented in Lemma 2.

Replace "if ADDRESS(i) = 0 then c_i"
with "if ADDRESS(i) = 0 then w_{+i}"
and replace "if ADDRESS(i) = 1 then c_i"
with "if ADDRESS(i) = 1 then w_{-i}."

Like the Cube network, the PM2I and WPM2I networks can sort $2^{m'}$ elements, m' < m, using only m'(m'+1) interprocessor data transfers.

The next lemma gives an algorithm for implementing a bitonic sorter for arbitrary sequences using the Perfect Shuffle network. The algorithm is similar to that for the Cube. It differs in that to compare the data that were originally in PE's that differ in the kth bit position, the data must be shuffled m-k times and then the exchange applied. To compare PE's that differ in their 0th bit position only the exchange need be used. To determine which type of comparator each PE should simulate we again use the results of Theorem 3. In this case, however, the data to be sorted will be shuffled before the exchange can be applied to simulate the action of the comparator. Thus, when taking the exclusive-or of the bits of the PE address we must consider the locations of the data at the beginning of that stage of the sorter, before the shuffles were applied. Stone[27] presents a similar algorithm using the shuffle interconnection and compare-exchange modules.

Lemma 5: The Perfect Shuffle network can sort arbitrary sequences of N elements using $2m^2$ interprocessor data transfers.

Proof: The algorithm at the top of the next column implements a bitonic sorter for arbitrary sequences of N elements.

This algorithm uses m(m-1) shuffles and m(m+1) exchanges. S1 through S7 implement the N/2 2-element bitonic sequence sorters. This is stage 1 of the arbitrary sequence sorter. S3 determines which PE's should simulate which type of comparator, based on the results of the previous theorem. S8 sets the variable j such that we next implement bitonic sequence sorters of size 4, then 8, then 16, ..., then N. S9 through S18 implement the bitonic sequence sorters. S14 is used to determine which type of comparator each PE should simulate. The indices of the ADDRESS register are set in such a way as to compensate for the fact that the data is shuffled before the exchanges occur. Thus, the algorithm implements an N-element bitonic sorting network for arbitrary sequences as described in the recursive construction of the previous section. Note that if j=m in S14, then the remaining portion of the boolean expression controlling the conditional is ignored. As an alternative to computing the exclusive-or in S14, its "initial" value may be computed between S9 and S10, outside the scope of the S10 to S18 loop. Then inside the scope of the S10 to S18 loop the value of

```
     begin
(S1) if ADDRESS(0) = 0 then A ← DTR
(S2) if ADDRESS(0) = 1 then e
(S3) if (ADDRESS(m-1) ⊕ ... ⊕ ADDRESS(2) ⊕ ADDRESS(1))=0
(S4)      then if DTR < A and ADDRESS(0)=0 then A ↔ DTR
(S5)      else if DTR > A and ADDRESS(0)=0 then A ↔ DTR
(S6) if ADDRESS(0) = 0 then e
(S7) if ADDRESS(0) = 0 then DTR ← A
(S8) for j=2 until m do
          begin
(S9)      for i=1 until m-j do s
(S10)     for i=1 until j do
               begin
(S11)          s
(S12)          if ADDRESS(0) = 0 then A ← DTR
(S13)          if ADDRESS(0) = 1 then e
(S14)          if (j=m) or (ADDRESS(m-(j+1)+i) ⊕ ...
                    ⊕ ADDRESS(i+1) ⊕ ADDRESS(i)) = 0)
(S15)                   then
                        if DTR < A and ADDRESS(0) = 0
                             then A ↔ DTR
(S16)                   else
                        if DTR > A and ADDRESS(0) = 0
                             then A ↔ DTR
(S17)          if ADDRESS(0) = 0 then e
(S18)          if ADDRESS(0) = 0 then DTR ← A
               end
          end
     end
```

the exclusive-or can be "updated" using a constant number of operations. In particular, insert the instruction:

COMP ← ADDRESS(m-(j+1))⊕...ADDRESS(1) ⊕ ADDRESS(0)

between S9 and S10; and replace S14 with:

"COMP ← COMP ⊕ (ADDRESS(m-(j+1)+i) ⊕ ADDRESS(i-1))

if (j=m) or COMP = 0."

Another alternative would be for each PE to precompute the values computed in S14 and store them in a bit vector of length (1/2)m(m+1). The vector could be accessed using an index that is incremented inside the S10 to S18 loop.

If either of these alternatives is used, the entire algorithm would require a constant times $2m^2$ operations. □

The algorithm in the previous lemma can sort $2^{m'}$ elements, where m' < m, by changing m to m' in S8. Thus, the algorithm for 2 elements would require m(m'-1) shuffles and m'(m'+1) exchanges.

The following theorem summarizes the results of this section.

Theorem 4: The following table shows an upper bound on the number of interprocessor data transfers required by each interconnection network to sort an arbitrary sequence of N elements.

Network	# Data Transfers
Cube	$m(m+1)$
PM2I	$m(m+1)$
WPM2I	$m(m+1)$
PS	$2m^2$
Illiac	$2nm-6m+16n-16$

Proof: For the Cube, PM2I, WPM2I, and Perfect Shuffle networks this follows from the four previous lemmas. The result for the Illiac network is from Orcutt's thesis[21], page 35. □

VI. Conclusions

We discussed the universality of various types of SIMD machine interconnection networks that have been proposed in the literature. We showed how the simulation of arbitrary interconnection functions can be accomplished by the sorting of destination tags. Batcher's bitonic sorting method was then analyzed. Algorithms to implement that sorting method on SIMD machines using different types of interconnection networks were presented.

One can observe from the table in Theorem 4 that the Cube, PM2I, and WPM2I networks can implement the bitonic sorter for arbitrary sequences almost twice as quickly as can the Perfect Shuffle. The Perfect Shuffle network, on the other hand, requires only two interconnection functions, while the Cube requires m, the PM2I requires 2m-1, and the WPM2I requires 2m. To sort $2^{m'}$ elements, $m' < m$, the PM2I, WPM2I, and Cube networks required only $m'(m'+1)$ interprocessor data transfers, while the Perfect Shuffle required $m(m'-1) + m'(m'+1)$. Thus, there is a tradeoff between the speed of the sorting operation and the number od interconnection functions required to implement the network.

The algorithms presented can be directly implemented on any SIMD machine satisfying our few assumptions. To actually simulate an interconnection, the number of interprocessor data transfers required would be twice the number necessary for sorting, since both the data and the destination tags must be moved. Thus, these results provide the system architect with an additional means of comparing the capabilities of different interconnection networks, as well as providing algorithms that can be used to simulate arbitrary interconnections if one of these networks is implemented.

VII. Acknowledgements

I would like to thank J. D. Ullman for his guidance and suggestions. I would also like to thank L. J. Siegel for her comments and help.

References

1. G. H. Barnes, et. al., "The ILLIAC IV computer," IEEE Trans. Comput., Vol. C-17 (Aug., 1968), pp. 746-757.
2. K. E. Batcher, "Sorting networks and their applications," 1968 Spring Joint Computer Conf., AFIPS Proc., Vol. 32, pp. 307-314, 1968.
3. K. E. Batcher, "STARAN/RADCAP hardware architecture," Proceedings of the 1973 Sagamore Computer Conference on Parallel Processing, pp. 147-152.
4. K. E. Batcher, "The Multi-dimensional access memory in STARAN," submitted to the IEEE Trans. Comput. Special Issue of Parallel Processing; summary in the Proceedings of the 1975 Sagamore Conference on Parallel Processing, p. 167.
5. L. H. Bauer, "Implementation of data manipulating functions on the STARAN associative processor," Proceedings of the 1974 Sagamore Computer Conference on Parallel Processing, pp. 209-227.
6. W. J. Bouknight, et. al., "The ILLIAC IV system," Proceedings of the IEEE, Vol. 60, No. 4 (Apr. 1972) pp. 369-388.
7. G. R. Couranz, M. S. Gerhardt, and C. J. Young, "Programmable RADAR signal processing using the RAP," Proceedings of the 1974 Sagamore Computer Conference on Parallel Processing, pp. 37-52.
8. B. A. Crane, et. al., "PEPE computer architecture," Sixth Annual IEEE Computer Society International Conference (1972), pp. 57-60.
9. T. Feng. Parallel Processing Characteristics and Implementation of Data Manipulating Functions, Dept. of Electrical and Computer Engineering, Syracuse University, RADC-TR-73-189 (July, 1973).
10. T. Feng, "Data manipulating functions in parallel processors and their implementations," IEEE Trans. Comput., Vol. C-23 (Mar., 1974), pp. 309-318.
11. M. J. Flynn, "Very high-speed computing systems," Proceedings of the IEEE, Vol. 54, No. 12 (Dec., 1966), pp. 1901-1909.
12. S. W. Golomb, "Permutations by cutting and shuffling," SIAM Review, Vol. 3, No. 4 (Oct., 1961) pp. 293-297.
13. L. C. Higbie, "The Omen computer: associative array processor," Compcon 72, IEEE Computer Society Conference Proceedings, (Sept., 1972), pp. 287-290.
14. P. B. Johnson, "Congruences and card shuffling," American Mathematical Monthly, Vol. 63 (Dec., 1956), pp. 718-719.
15. D. E. Knuth, The Art of Computer Programming: Vol. 3 Sorting and Searching, Additon-Wesley, section 5.3.4, 1973.
16. T. Lang, "Interconnections between processors and memory modules using the shuffle-exchange network," IEEE Trans. Comput., Vol. C-25 (May, 1976), pp. 496-503.
17. T. Lang and H. S. Stone, "A shuffle-exchange network with simplified control," IEEE Trans. Comput., Vol. C-25 (Jan., 1976), pp. 55-56.
18. D. Lawrie, Memory-Processor Connection Networks, Dept. of Computer Science, University of Illinois, Rep. 557. (Feb., 1973).
19. D. Lawrie, "Access and alignment of data in an array processor," IEEE Trans. Comput., Vol. C-24 (Dec., 1975), pp. 1145-1155.
20. D. H. Lawrie, T. Layman, D. Baer and J. M. Randall, "Glypnir - A programming language for Illiac IV," CACM, Vol. 18, No. 3 (Mar., 1975), pp. 157-164.
21. S. E. Orcutt, Computer Organization and Algorithms for Very-High Speed Computation, Dept. of Computer Science, Stanford University, Ph.D. Thesis, (Sept., 1974).
22. D. Rahmlow, "Parasim," Princeton University, unpublished paper, (1974).

23. H. J. Siegel, "Analysis techniques for SIMD machine interconnection networks and the effects of processor address masks," IEEE Trans. Comput., to appear Feb., 1977.

24. H. J. Siegel, "Single instruction stream - multiple data stream machine interconnection network design," Proceedings of the 1976 International Conference on Parallel Processing, (Aug., 1976), pp. 273-282.

25. D. L. Slotnick, et. al., "The SOLOMON computer," 1962 Fall Joint Computer Conf., AFIPS Proc., Vol. 22 (1962), pp. 97-107.

26. K. G. Stevens, Jr., "CFD - A FORTRAN-like language for the Illiac IV," Proceedings of a Conference on Programming Languages and Compilers for Parallel and Vector Machines, sponsored by ACM (Mar., 1975), pp. 72-76.

27. H. S. Stone, "Parallel processing with the perfect shuffle," IEEE Trans. Comput., Vol. C-20 (Feb., 1971), pp. 153-161.

28. A. H. Webster, "Special features in SIMDA," Proceedings of the 1972 Sagamore Computer Conference, pp. 29-40.

29. D. E. Wilson, "The PEPE support software system," Sixth Annual IEEE Computer Society International Conference (1972), pp. 61-64.

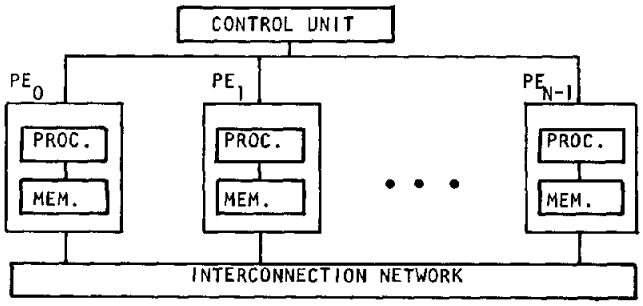

Figure 1: Processing element-to-processing element configuration.

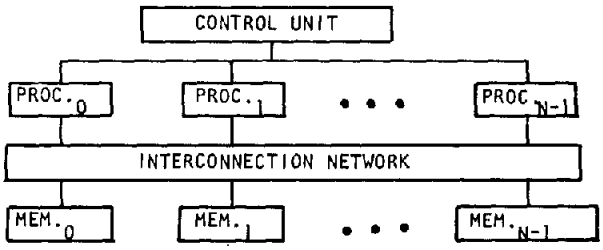

Figure 2: Processor-to-memory configuration.

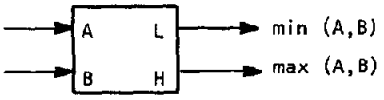

Figure 3: Comparator module.

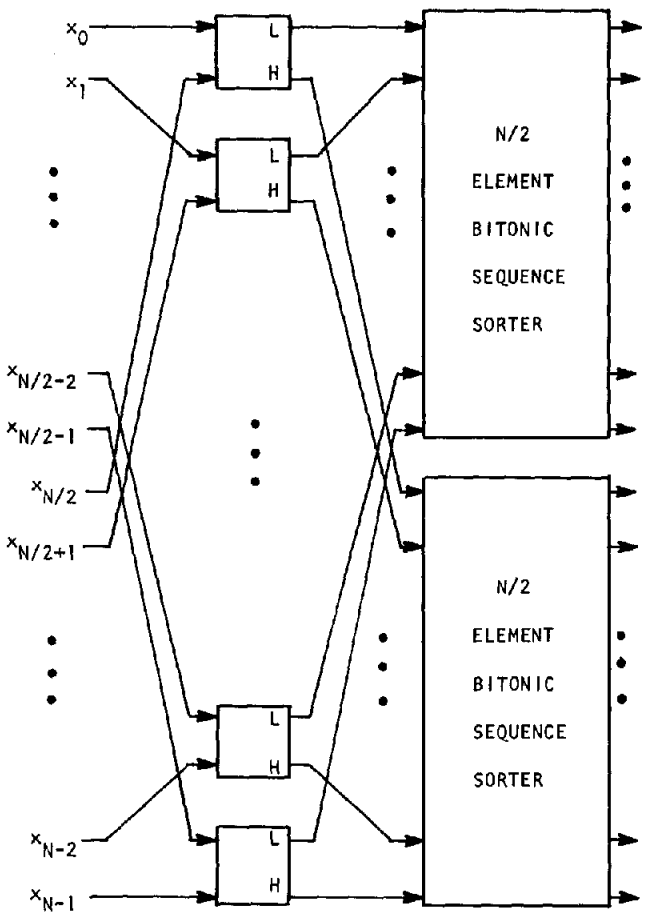

Figure 4: Iterative technique for constructing bitonic sequence sorter.

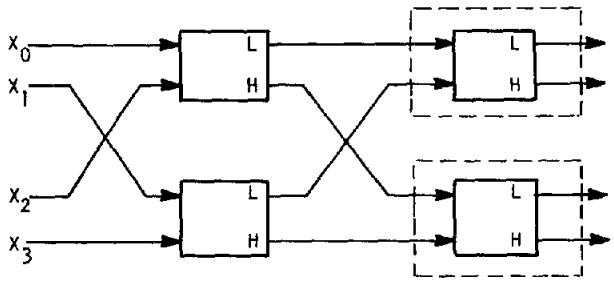

Figure 5: A 4-element bitonic sequence sorter.

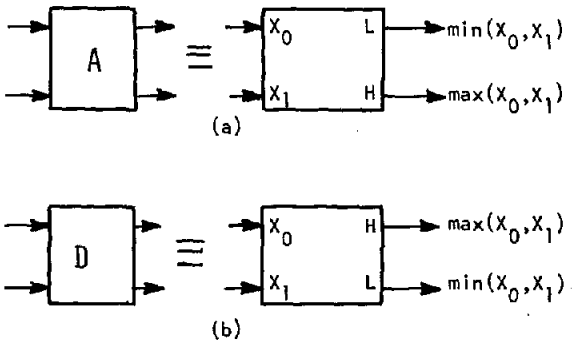

Figure 6: (a) Ascending comparator.
(b) Descending comparator.

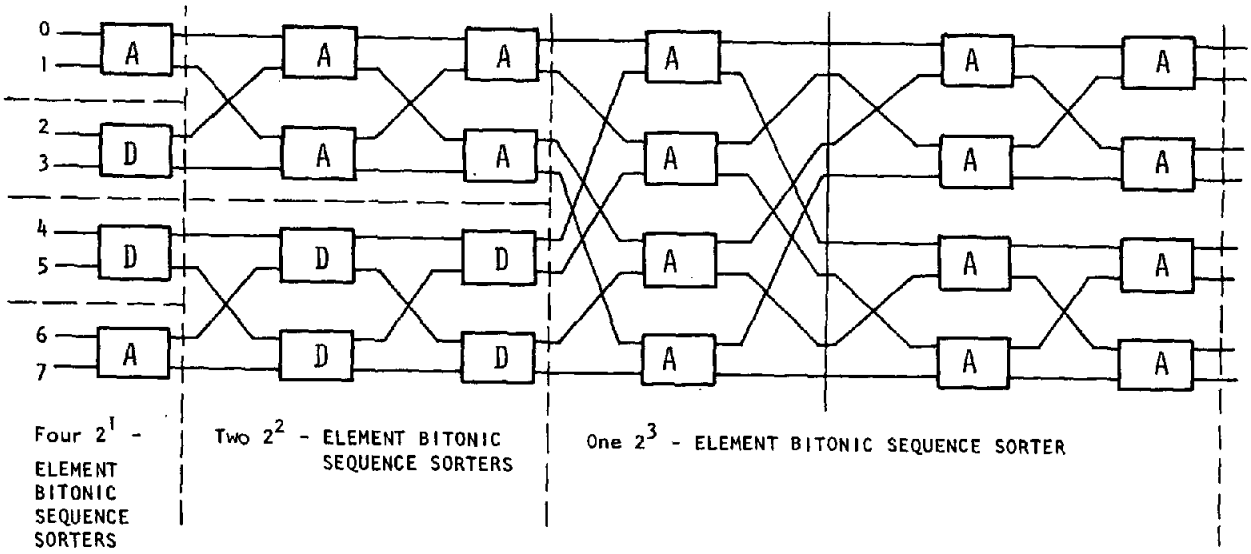

Figure 7: Bitonic sorter for an arbitrary sequence of 8 elements.

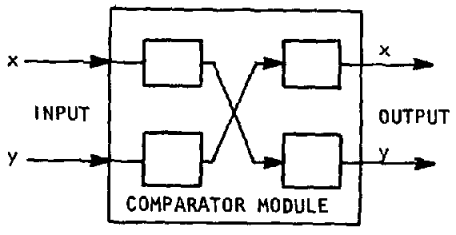

Figure 8: Input/output labels and interior of comparator module.

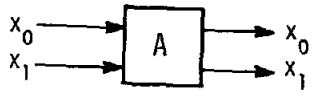

Figure 9: Input/output lables of a descending comparator.

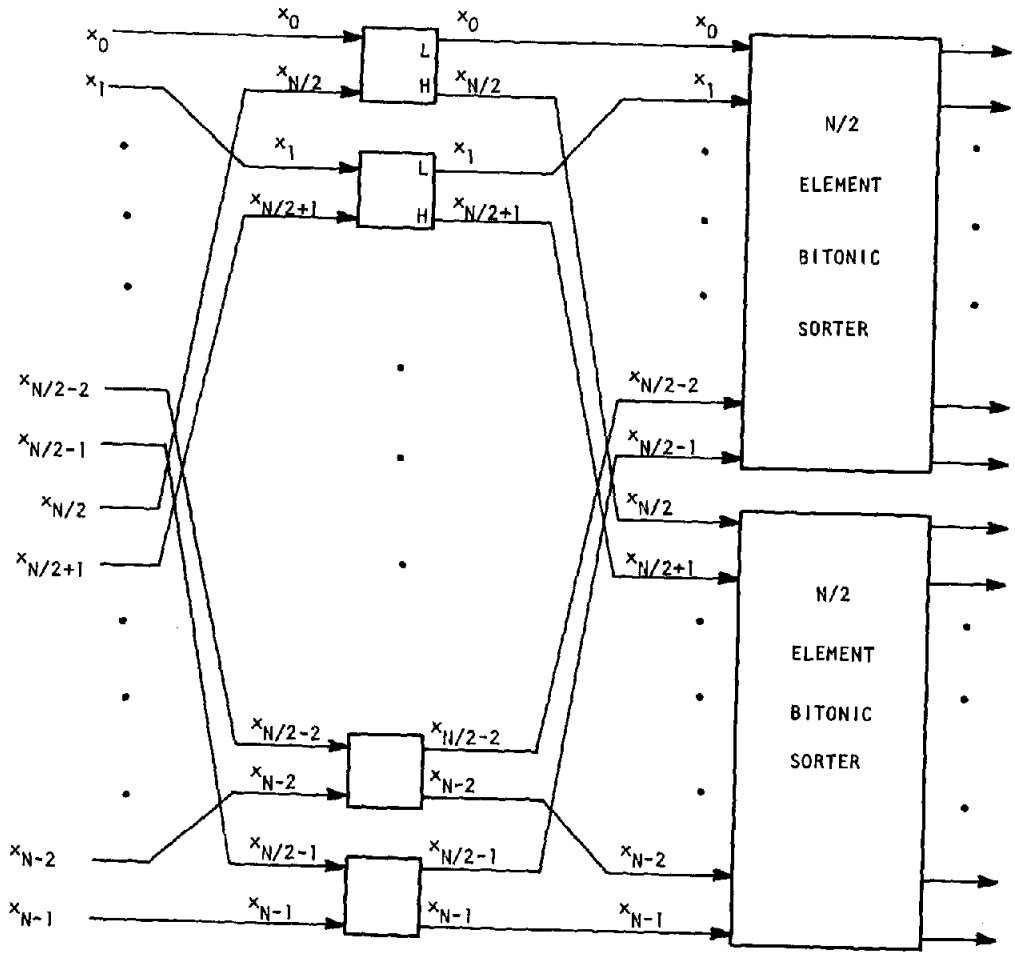

Figure 10: Iterative technique for constructing bitonic sequence sorter, with input/output labels.

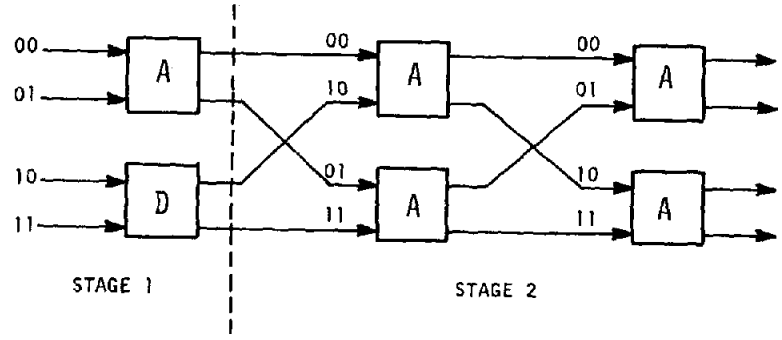

Figure 11: A 4-element sorter.