Optimal BPC Permutations on a Cube Connected SIMD Computer

DAVID NASSIMI AND SARTAJ SAHNI

Abstract—In this correspondence we develop an algorithm to perform BPC permutations on a cube connected SIMD computer. The class of BPC permutations includes many of the frequently occurring permutations such as matrix transpose, vector reversal, bit shuffle, and perfect shuffle. Our algorithm is shown to be optimal in the sense that it uses the fewest possible number of unit routes to accomplish any BPC permutation.

Index Terms—BPC permutation, cube connected SIMD computer, complexity.

I. INTRODUCTION

An SIMD (single instruction stream—single data stream) computer is a parallel computer consisting of a large number of identical processing elements. A block diagram of such a computer is given in Fig. 1. SIMD computers have the following characteristics.

1) They consist of \( N \) processing elements (PE’s). The PE’s are indexed \( 0, 1, \ldots, N - 1 \) and an individual PE may be referenced as \( \text{PE}(i) \). Each PE is capable of performing the standard arithmetic and logical operations. In addition, each PE knows its index.

2) Each PE has some local memory.

3) The PE’s are synchronized and operate under the control of a single instruction stream. This instruction stream is generated by the control unit which has access to the program that is to be run.

4) An enable/disable mask can be used to select a subset of the PE’s that are to perform an instruction. Only the enabled PE’s will perform the instruction. The remaining PE’s will be idle. All enabled PE’s execute the same instruction. The set of enabled PE’s can change from instruction to instruction.

The essential feature that distinguishes one SIMD computer family from another is the interconnection network. In this correspondence we are concerned only with two types of interconnection networks: the mesh and the cube.

1) Mesh Connected Computer (MCC): In this model the PE’s may be thought of as being logically arranged as in a two-dimensional array \( A(n_1, n_2) \), where \( N = n_1 \times n_2 \). The PE at location \( A(i, j) \) is directly connected to the PE’s at locations \( A(i \pm 1, j) \) and \( A(i, j \pm 1) \) (provided these PE’s exist). Fig. 2(a) shows the interconnections in a \( 4 \times 4 \) MCC.

2) Cube Connected Computer (CCC): Assume that the number of PE’s \( N \) is a power of 2. So \( N = 2^k \). Let \( b_{k-1} \cdots b_0 \) be the binary representation of \( i \), \( i \in [0, N - 1] \) and let \( i^{(0)} \) denote the number whose binary representation is \( b_{k-1} \cdots b_0 \cdots b_{k-2} b_{k-1} \cdots b_0 \), where \( b_0 \) is the complement of \( b_0 \) and \( 0 \leq b < k \). Hence, if \( i \) has the binary representation 10110, then \( i^{(2)} \) has the representation 10111. In a cube connected computer, PE(\( i \)) is connected to PE(\( i^{(0)} \)), \( 0 \leq b < k \). Fig. 2(b) shows the PE interconnections in an 8 PE CCC.

It is important to note that PE’s can communicate only via the interconnection network. Besides the mesh and cube connections, several other connection schemes are possible. The reader is referred to Siegel [6], [7] for a survey of interconnections networks for SIMD computers. The largest SIMD computer currently under construction is the massively parallel processor (MPP) being built by Goodyear

Manuscript received April 10, 1981. This work was supported in part by the Office of Naval Research under Contract N00014-80-C-0650.

D. Nassimi is with the Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL 60201.

S. Sahni is with the Department of Computer Science, University of Minnesota, Minneapolis, MN 55455.

0018-9340/82/0400-0338$00.75 © 1982 IEEE
Figure 1. Block diagram of an SIMD computer.

Figure 2. (a) 4 x 4 MCC. (b) 8 PE CCC.

Aerospace Research [1]. This machine uses the mesh interconnection scheme (together with some variations) and will have 16384 PE's.

An important problem that arises in SIMD computers is that of data routing: moving data from one PE to another. While there are several forms of the data routing problem [5], we shall deal only with the permutation form. In a permutation problem PE(i) wishes to send data to PE(A(i)), 0 ≤ i < N where [A(0), · · · , A(N − 1)] is a permutation of [0, 1, · · · , N − 1]. Arbitrary data permutations are generally accomplished by sorting. For certain classes of permutations, however, there exist algorithms that are more efficient than the best sorting algorithms [4]. One such class is the BPC (bit permute complement) class of permutations introduced in [3]. A permutation A is a BPC permutation if it can be described by a vector

B = [Bq−1, Bq−2, · · · , B0] (where N = 2q is the number of PE's), such that

1) Bt ∈ {±0, ±1, · · · , ± (q − 1)}, 0 ≤ t < q, and
2) [[Bq−1], [Bq−2], · · · , [B0]] is a permutation of [0, 1, · · · , q − 1].

The destination d of the data in PE(i) can be computed from this vector B as follows. Let i1, i2, · · · , il be the binary representation of i and let d1, d2, · · · , dl be the binary representation of d. For j = 0, 1, · · · , q − 1, we have

\[ d_{|B_j|} = \begin{cases} i_j & \text{if } B_j \geq 0 \\ i_j & \text{if } B_j < 0 \end{cases} \]

Note that we distinguish between +0 and −0 and that −0 < +0 = 0. Also note that the total number of permutations that can be specified in this way is 2q! = N!(log N)!

Intuitively, for each BPC permutation A specified by B, the destination PE for PE(i) is obtained by permuting the bits in the binary representation of i and complementing certain bits. The vector B specifies how the bits are to be complemented and also which bits are to be complemented. [Bt] tells us where bit i is to go, and the sign of Bt tells us if the jth bit of i is to be complemented.

As an example, consider the case N = 16, q = 4 and B = [−1, 0, −1, 2]. The data from PE(0), i.e., i = j3j2j1j0, are to be routed to PE(1), where j = j3j2j1j0 = i3i2i1i0, 0 ≤ i < N. The BPC class of permutations includes most of the permutations that commonly arise. Table I gives the B vectors corresponding to several popular permutations.

Nassimi and Sahni [3] present an optimal algorithm for routing BPC permutations on mesh-connected computers. In [4] they show how BPC permutations may be performed efficiently on a CCC. Their algorithm is, however, suboptimal in the sense that for some BPC permutations more data movement may take place than necessary. In this correspondence we develop an optimal algorithm for routing BPC permutations on a CCC.

II. OPTIMAL BPC ALGORITHM

Let b ∈ [0, q − 1]. In a unit-route (on a CCC) data can be moved from PE(i) to PE((i)b), 0 ≤ i < N. Let B = [Bq−1, · · · , B0] be the vector representation of the BPC permutation A = [A(0), · · · , A(N − 1)]. We first obtain a lower bound \( \beta(B) \) on the number of unit-routes needed to perform A on an N PE CCC.

Theorem I: Let B = [Bq−1, · · · , B0] define the BPC permutation A = [A(0), · · · , A(N − 1)]. \( \beta(B) \) as given below is a lower bound on the number of unit-routes needed to perform A on a CCC.

\[ \beta(B) = |b| : B_0 \neq b | \]

Proof: For each b for which \( B_0 \neq b \), there exists at least one A(i) with the property that \( b_0 \neq A(i)_0 = (A(i))_0 \), denotes bit b of A(i). Thus, at least one unit-route along bit b is needed. So at least \( \beta(B) \) unit-routes are needed to perform A.

By making minor modifications to the routing algorithm presented in [4], we can arrive at an algorithm that performs each BPC permutation B using the \( 2\beta(B) \) unit-routes. The algorithm we are about to present will use exactly \( \beta(b) \) unit-routes and is therefore optimal.

Our algorithm follows the cycles present in the bit permutation B. If (k1, k2, · · · , kp) are the bits in a cycle of A, then our algorithm first routes all data to PE's having the correct final value for bit k1 (i.e., following this route the destination D, for the data in PE(i) is such that (D)k1 = (i)k1). Next, we route along k2, then k3, etc. Having finished with this cycle, the next permutation cycle is followed and so on.

Let us first consider an example. Consider performing a perfect shuffle on a CCC with 8 PE's (Fig. 3). B = [0, 2, 1] and the destination A(i) for the data in PE(i) has the binary representation i1i0i2 (note that the binary representation of i is i3i2i1i0). The elements to be permuted are assumed to be in register R of each PE. B has only one cycle (1, 2, 0) = (B0, B1, B2). The first route is along bit 1, then along bit 2, and finally along bit 0. The route along bit j is done only for those PE's containing data with destination A(i) such that \( i_j \neq (A(i))_j \). In our example, when routing along bit 1 we need to route only data from PE(i) with \( i_1 \neq i_0 \). This is so because \( A(i)_1 = i_0 \) and if \( i_1 = i_0 \) then the data in PE(i) are already in a PE with the right bit \( i_1 \). Data to be routed are moved to a routing register S and the route performed.

We shall use the following notation and assumptions in specifying our permutation algorithm.

1) Each PE has two registers R and S. Both these registers are large enough to hold the data being routed. \( R(i) \) and \( S(i) \) refer to the corresponding registers in PE(i).
TABLE I

<table>
<thead>
<tr>
<th>Permutation</th>
<th>Vector Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix Transpose</td>
<td>([q/2 - 1, \ldots, 0, q - 1, \ldots, q/2])</td>
</tr>
<tr>
<td>Bit Reversal</td>
<td>([0,1,2,\ldots,q-1])</td>
</tr>
<tr>
<td>Vector Reversal</td>
<td>([-q - 1, -q - 2, \ldots, 0])</td>
</tr>
<tr>
<td>Perfect Shuffle</td>
<td>([0, q - 1, 2, \ldots, q])</td>
</tr>
<tr>
<td>Unshuffle</td>
<td>([q - 2, q - 3, \ldots, 0, q - 1])</td>
</tr>
<tr>
<td>Shuffled Row Major</td>
<td>([q - 1, q/2 - 2, q - 2, \ldots, q, q/2])</td>
</tr>
<tr>
<td>Bit Shuffle</td>
<td>([q - 1, q - 3, \ldots, 1, q - 4, q - 4, \ldots, 0])</td>
</tr>
</tbody>
</table>

Fig. 3. Perfect shuffle on a cube.

2) Three types of assignments will be used.

a) := will be used for assignments requiring no routing. For example, \(R(i) := S(i)\) (both \(R\) and \(S\) are in the same PE).

b) := will be used for exchanges requiring no routing. \(R(i) := S(i)\) results in the \(R\) and \(S\) registers of \(PE(i)\) interchanging data.

c) := will denote an assignment requiring a route. We shall require that the PE’s denoted by the left- and right-hand sides be connected by a direct link in the PE interconnection pattern. For example, \(R(i)b := S(i)\) is valid for a CCC (recall that \(b\) is obtained from \(i\) by complementing bit \(b\) in the binary representation of \(i\)). Each assignment of this type is a unit-route.

3) \(i_b\) will denote bit \(b\) in the binary representation of \(i\).

4) PE selectivity can be done using a mask. The mask is specified in parenthesis following the statement. Some examples of masks are as follows:

i) \((i_b = 1)\): this enables all PE’s for which the binary representation of the PE index has bit \(b\) equal to 1.

ii) \((i_b \neq i_k)\): this enables all PE’s for which the \(j\)th bit in the binary representation of the PE index is different from the \(k\)th bit.

When no mask is specified, all PE’s are enabled. Instructions are executed only on enabled PE’s.

Procedure BPC–CUBE below is a formal statement of our BPC permutation algorithm for CCC’s.

```
procedure BPC–CUBE \((B, q)\)
//Permute \(R(0:2^q - 1)\) according to the BPC permutation \(B(0:q - 1)\)
1 for \(b = 0\) to \(q - 1\) do
2 case
3 :\(B_b = b\): do nothing
4 :\(B_b = -b\): \(R(i)b := R(i)\)
5 :\(|B_b| \neq b\):
6 \(j := b\); \(z := B_b\)
7 repeat
8 \(k := |B_z|\) //Next route is along dimension \(k\) //Put outgoing elements in \(S\) //if \(B_j \geq 0\) then \(S(i) := R(i), (i_j \neq i_k)\)
9 \(\text{else } S(i) := R(i), (i_j = i_k)\)
10 \(S(i)b := S(i)\)
11 \(B_j := j; j := k\)
12 until \(j = b\)
13 \(k := |x|\) // is the initial \(B_b\) //if \(x \geq 0\) then \(R(i) := S(i), (i_b \neq i_k)\)
14 \(\text{else } R(i) := S(i), (i_b = i_k)\)
15 end
16 end
```

The loop of lines 1–16 searches for the beginning of a bit cycle. If \(|B_b| = b\), we have a cycle of length 1. When \(B_b = -b\), no work needs to be done. When \(B_b = b\), it is necessary to complement along bit \(b\) (line 4). If \(|B_b| \neq b\), then we are at the start of a cycle of length more than 1. Lines 7–12 follow this cycle. \(j\) is used to move along \(b\), \(B_b\), \(|B_b|\), \(R(i)\), etc. Line 9 puts into \(S(i)\) the data to be routed out of \(PE(i)\). Line 10 carries out the route along bit \(k\), and then in line 11 \(B_j\) is set to \(j\) to signify that the cycle containing \(j\) has been taken care of by the time we exit from the case statement. \(j\) is moved to the next point on the cycle. Line 14 moves all valid data to the \(R\) registers.

We need to elaborate upon two of the statements just made concerning the algorithm. First, we need to show that line 9 moves into the \(S\) registers all records that are in a PE whose \(k\)th bit does not agree with the \(k\)th bit of its destination PE. Second, we need to show that line 14 correctly leaves all records in the \(R\) registers.

Let \(TR(i)\) and \(TS(i)\), respectively, denote the source or originating PE for the records currently in \(R(i)\) and \(S(i)\). Let \(AR(i)\) and \(AS(i)\) denote the destination PE’s for the records in \(R(i)\) and \(S(i)\), respectively. At the start of each bit cycle (line 6) all records are in \(R(i)\). Let \(TS(i) = \phi\) and \(AS(i) = \phi\). So initially at line 8 the following loop invariant holds: \((TR(i))_j = i_j\) and \((TS(i))_j \neq i_j, 0 \leq j < N\). (We shall assume that all relations involving \(\phi\) are true. So \((\phi) = i_j\) is true and \((\phi) = i_j\) is also always true.) The relation holds since no routing on bit \(j\) could have been performed on any previous iteration of the for loop of lines 1 to 16. If \(B_j \geq 0\) then for each PE \(PE(i)\) with \(i_j \neq i_k\), we have

\[(AR(i))_k = (TR(i))_j = i_j = i_k \text{ and } (AS(i))_k = (TS(i))_j = i_k.\]

Hence, \(R(i)\) needs to be routed along bit \(k\) and \(S(i)\) does not. If \(i_j = i_k\), then we have

\[(AR(i))_k = (TR(i))_j = i_k \text{ and } (AS(i))_k = (TS(i))_j = i_k.\]

Hence, \(S(i)\) needs to be routed along bit \(k\) and \(R(i)\) does not. So, line 9 correctly places into the \(S\) registers the records that need to be routed along bit \(k\) when \(B_j \geq 0\). Using a similar argument one can show the correctness of line 9 when \(B_j < 0\). So following line 11 we have

\[(AR(i))_k = (AS(i))_k = i_k, \quad 0 \leq i < N.\]

Also note that preceding the execution of line 10

\[(TR(i))_k = (TS(i))_k = i_k\]

as no routes along bit \(k\) have yet been performed. Line 10 routes only \(S\) values, so after line 10 we shall have

\[(TR(i))_k = i_k \text{ and } (TS(i))_k \neq i_k, \quad 0 \leq i < N.\]

As a result, on all subsequent iterations of the loop of lines 7–12, the loop invariant \([(TR(i))_k = i_k \text{ and } (TS(i))_k \neq i_k, 0 \leq i < N\) still holds before line 8. So line 9 will correctly set \(S(i)\) and line 10 will route correctly.
The preceding argument shows that when the loop of lines 7-12 is completed for any $b$, then

$$(AR(i))_q = (AS(i))_q = i_q \quad \text{for all } q \in \{|B_k|, |B_{|B_k|}, \cdots, b\}.$$ 

It remains to move all records back into the $R$ registers (line 14). When a cycle is finished, half the records will be in the $R$ registers and half in the $S$ registers. The records in the $S$ registers need to be moved to the $R$ registers. The first time line 9 is executed for any cycle, $j = b$ and $k = |s|$. When line 10 is executed, records leave half the PE's and the remaining half contain two records each. The empty PE's are those with $i_b = i_{|s|}$ if $B_b > 0$ and $i_b = i_{|s|}$ if $B_b < 0$. Since bits $b$ and $B_b$ do not get used in line 10 again until the last iteration of the repeat-until loop, these PE's remain empty. They get a record only after the last execution of line 10 for this cycle. At this time $k = b$. Thus, the PE's containing records in their $S$ registers are those with index $i$ such that $i_b \neq i_{|s|}$ if $s \geq 0$ and $i_b = i_{|s|}$ if $s < 0$ (note that $s = B_b$). Hence, lines 6-14 correctly handle cycles of length more than 1 and leave all records in the $R$ registers. From this and lines 3 and 4, it follows that all cycles are handled correctly and BPC_CUBE performs every BPC permutation. The time complexity of BPC_CUBE is $O(\log N)$ and the number of unit-routes (lines 4 and 10) is $\beta(B)$. Hence, BPC_CUBE is optimal.

III. CONCLUSIONS

We have presented an optimal BPC routing algorithm for cube connected computers. An optimal BPC routing algorithm for mesh connected computers is given in [3]. Several open problems remain. Is there a similarly optimal algorithm to perform BPC permutations on perfect shuffle computers (see [5] for a description of the interconnection network used here)? Can we develop optimal algorithms for other classes of permutations such as omega and inverse omega permutations [2], etc.?

REFERENCES