In the Proceedings of the 12th International Parallel Processing Symposium, Orlando, FL, March 30 - April 3, 1998

Optimal All-to-Some Personalized Communication on Hypercubes

Y. Charlie Hu
Department of Computer Science
Rice University
6100 Main Street
Houston, TX 77005-1892
ychu@cs.rice.edu

Abstract

In a hypercube multiprocessor with distributed memory, each data element has a street address and an apartment number (i.e. a hypercube node address and a local memory address). We describe an optimal algorithm for performing the all-to-some personalized communication (ASPC) on Boolean n-cubes, defined as $(i|j) \to (i \pm 2^j|j)$, $i \in [0, 2^n - 1]$, $j \in [0, n-1]$, where $(i|j)$ denote the data element on node i and location j. The algorithm also gives an optimal schedule for emulating PM2I networks on hypercubes under the binary-reflected Gray code encoding.

We also study an important class of parallel algorithms, called $\pm 2^b$-descend, which perform $\log M$ iterations on an M-element input $a[0 : M-1]$. For $b = \log M - 1, ..., 0$, iteration b computes new values of each a[i] as a function of $a[i], a[i+2^b], a[i-2^b]$. For large applications, the problem size M is typically much larger than the number of nodes N. We show that on hypercubes, the optimal ASPC algorithm devised in this paper can be used in combination with pipelining communication and computation in $\pm 2^b$-descend computations to reduce the communication steps from $2 \cdot \log N \cdot \frac{M}{N}$ to $4(\log M + \frac{M}{N} - 1)$. At one communication step, a hypercube node can send n elements along its n links, one per link.

1. Introduction

Assume $n2^n$ elements are distributed among an n-dimensional hypercube, with n elements per node. We represent the address of the jth element on node i as $(i|j)$. We define the *all-to-some personalized communication* (ASPC) pattern to be $(i|j) \to (i \pm 2^j|j)$, $i \in [0, 2^n - 1]$, $j \in [0, n-1]$. This pattern reflects the inter-connection of PM2I networks [6]. It can also be viewed as performing n concurrent 2^b permutations and n concurrent -2^b permutations on an n-dimensional hypercube with $b \in [0, ..., n-1]$. A 2^b permutation of N elements moves the element at position i to position $i + 2^b$ mod N. The inverse is called a -2^b permutation.

If we restrict to always using the same links of the different nodes – links that change the same bit of the node addresses (called SIMD hypercubes in [4]) – at a time, it has been proved that at least $\log N - b$ communication steps are required for a 2^b permutation no matter what mapping mechanism is used to embed the data elements [4]. On SIMD hypercubes, a hypercube node can send one element along one of its n links at one communication step.

For a hypercube on which data can be sent along any links connecting a node with its hypercube neighbor nodes, a hypercube node can send n elements along its n links at one communication step, one per link. On such hypercubes, it has been shown that a 2^b permutation can be performed in $O(1)$ steps, using a Gray-code mapping of data elements to the hypercube nodes [3]. The algorithm involves concurrent communications on different nodes along different links, although one link at a time. Straightforward use of this algorithm for ASPC would require $O(n)$ communication steps.

The challenge of scheduling ASPC more efficiently comes from the fact that the two bits that addresses of the source and destination nodes i and $i \pm 2^j$ differ depend on both i and j. In this paper, we describe an optimal algorithm for performing the all-to-some personalized communication on Boolean n-cubes. The algorithm performs the communication in four steps, using all or almost all the links at each routing step. The algorithm effectively gives an optimal schedule for emulating PM2I networks on hypercubes under the binary-reflected Gray code encoding.

We also study an important class of parallel algorithms, called $\pm 2^b$-descend [4], which perform $\log M$ iterations on an M-element input $a[0 : M-1]$. For $b = \log M - 1, ..., 0$, iteration b computes new values of each a[i] as a function of $a[i], a[i+2^b]$, and $a[i-2^b]$. In [4], an algorithm requiring $O(\log N)$ communication steps on SIMD hypercubes was developed. However, the algorithm assumes that the number of data elements is the same as the number of hypercube nodes. Furthermore, the algorithm performs three times more redundant computations at each iteration.

For large applications, the problem size M is typically much larger than the number of nodes N. Directly applying the method in [4] thus will result in $2 \cdot \log N \cdot \frac{M}{N}$ computation steps, plus the redundant computations. We show that on hypercubes with all-port communications, the optimal ASPC algorithm devised in this paper can be used in combination with pipelining communication and computation to reduce the communication steps to $4(\log M + \frac{M}{N} - 1)$ and without redundant computation.

In the following, we first review some properties of hypercubes and the binary-reflected Gray code. The optimal ASPC algorithm is then presented in Section 3, together with its correctness proof. Section 4 describes how to use the optimal ASPC algorithm to reduce the communication in $\pm 2^b$-descend type of parallel computations.

2. Hypercubes and Gray-Code Embedding

2.1. Hypercubes

An n-dimensional hypercube consists of 2^n nodes. Nodes j and k are nearest neighbors if the n-bit binary representations of them differ by exactly one bit. Let $j \oplus k$ be the bitwise exclusive or of the binary representations of j and k.

We assume communication edges are directed and there are $n2^n$ directed edges all together in an n-dimensional cube. Each node has n outgoing links and n incoming links. Let link k be the link that changes the kth bit of the end node addresses.

2.2. Binary-Reflected Gray Code

The binary-reflected gray code sequence [5] derives its name from the fact the second half of the code is identical to the first half in reverse order. The transition sequence is defined recursively as follows:

$$\begin{aligned} T_1 &= 0 \\ T_n &= T_{n-1} \circ n-1 \circ T_{n-1} \\ \text{where } \circ &= \text{string concatenation} \end{aligned}$$

Given this transition sequence, the binary-reflected gray code sequence G_n is easily derived: $G_n(0) = 0^n$, and for $0 \leq i < 2^n$, $G_n(i+1)$ equals $G_n(i)$ with bit $T_n(i)$ complemented. To make it circular, we can define

$$H_n = T_n \circ n-1$$

Then $G_n(2^n - 1)$ will go back to $G_n(0)$ if the $H_n(2^n - 1)$ bit is complemented.

Let $i = (b_{n-1}b_{n-2}...b_0)$ be the binary representation of i and $G_n(i) = (g_{n-1}g_{n-2}...g_0)$ be the binary-reflected gray code of i. The conversions between the two are defined by $g_i = b_{i+1} \oplus b_i$ and $b_i = (\sum_{j=i}^{n-1} g_j) \bmod 2$.

Lemma 1 *Let* $i = (b_{n-1}b_{n-2}...b_0)$, $H_n(i) = \sum_{l=0}^{n-1} \Pi_{m=0}^{l} b_m$. $H_n(i)$ *equals the number of ones before the first zero from right to left in the binary representation of* i.

Lemma 2

$$H_n(m2^{j+1} + 2^j - 1) = H_n(2^j - 1) = j, \forall j \in [0, n-1], m \in [0, 2^{n-j-2}],$$

and thus the periodicity of j *in* H_n *is* 2^{j+1}.

Lemma 3 $|T_j| = 2^j - 1$, *where* $|s|$ *denotes the length of the sequence* s.

Lemma 4

$$H_n = T_j \circ k_0 \circ T_j \circ k_1 \circ ... \circ k_{2^{n-j}-2} \circ T_j \circ k_{2^{n-j}-1},$$

where $k_p \in [j, ..., n-1], \forall p \in [0, 2^{n-j} - 1]$. *Furthermore,* $k_p \neq k_{p+1}$.

Corollary 1 $H_n(m \cdot 2^j - 1) \geq j$.

Let $\frac{a}{b}$ denote the integer division of a over b.

Theorem 1

$$G_n(i) \oplus G_n(i + 2^j) = \begin{cases} 2^{H_n(\frac{i}{2^{j-1}}2^{j-1} + 2^j - 1)} + \\ \quad 2^{H_n(\frac{i}{2^{j-1}}2^{j-1} + 2^{j-1} - 1)}, & \text{if } j > 0, \\ 2^{H_n(i)}, & \text{if } j = 0. \end{cases}$$

$$G_n(i) \oplus G_n(i - 2^j) = \begin{cases} 2^{H_n(\frac{i}{2^{j-1}}2^{j-1} - 1)} + \\ \quad 2^{H_n(\frac{i}{2^{j-1}}2^{j-1} - 2^{j-1} - 1)}, & \text{if } j > 0, \\ 2^{H_n(i-1)}, & \text{if } j = 0. \end{cases}$$

Proof: We can rewrite Lemma 4 as:

$$H_n = T_{j-1} \circ k_0 \circ T_{j-1} \circ k_1 \circ ... \circ k_{2^{n-j+1}-2} \circ T_{j-1} \circ k_{2^{n-j+1}-1},$$

where $k_p \in [j-1, ..., n-1], \forall p \in [0, 2^{n-j+1} - 1]$. If we complement the bits $H_n(i), H_n(i+1), ..., H_n(i+2^j-1)$ of $G_n(i)$ one at a time, we will end up with $G_n(i + 2^j)$. Since the sequence from $H_n(i)$ to $H_n(i + 2^j - 1)$ has length 2^j, it will cross exactly two adjacent k_p; they are the two bits that $G_n(i)$ and $G_n(i + 2^j)$ differ. □

Lemma 5 *Fix* i, $H_n(\frac{i}{2^{j-1}}2^{j-1} + 2^j - 1), j \in [0, n-1]$ *are distinct. So are* $H_n(\frac{i}{2^{j-1}}2^{j-1} + 2^{j-1} - 1)$, $j \in [0, n-1]$.

Proof: From Corollary 1, $H_n(\frac{i}{2^{j-1}}2^{j-1} + 2^j - 1) \geq j - 1$, and from Lemma 2, its periodicity is at least 2^j, thus it will not appear in $H_n(i), ..., H_n(\frac{i}{2^{j-1}}2^{j-1} + 2^j - 2)$. □

Lemma 6 *The smaller of the two bits that $G_n(i)$ and $G_n(i+2^j)$ differ is $j-1$, same with $G_n(i)$ and $G_n(i-2^j)$.*

Proof: The binary encoding of i and that of $i+2^j$ start to differ at the jth bit. From $g_i = b_{i+1} \oplus b_i$, their binary-reflected gray codes start to differ at $(j-1)$th bit. $\square$

Lemma 7 *Let $i = m2^j + k$,*

- if $0 \leq k < 2^{j-1}$,
  $H_n(\frac{i}{2^{j-1}}2^{j-1} + 2^{j-1} - 1) = H_n(\frac{i}{2^{j-1}}2^{j-1} - 2^{j-1} - 1)$
  $= j-1$,
- if $2^{j-1} \leq k < 2^j$,
  $H_n(\frac{i}{2^{j-1}}2^{j-1} + 2^j - 1) = H_n(\frac{i}{2^{j-1}}2^{j-1} - 1) = j-1$.

3. An Optimal Algorithm for ASPC

We now describe an optimal algorithm for all-to-some personalized communication described in the introduction.

Our optimal algorithm uses the Gray-code embedding of nodes. From Theorem 1 and Lemma 5, under the Gray-code embedding, every element only needs to travel two links to reach its destination node for either half ASPC. Furthermore, all n elements from the same node can traverse their first links concurrently without conflict. We will show in this section (Theorem 5) that all the elements from all the nodes can also traverse their second links conflict-free.

Assume we embed 2^n logical nodes into an n-dimensional hypercube. Processor i will be on physical node $G_n(i)$. Each node has n data elements. The address of the jth element on node i is denoted as $(i|j)$. The all-to-some personalized communication pattern is defined as

$$(i|j) \rightarrow (i \pm 2^j|j),\ \forall i \in [0, 2^n-1], j \in [0, n-1].$$

Theorem 2 *The all-to-some personalized communication requires at least 4 communication steps on a Boolean n-cube for $n > 2$.*

Proof: From Theorem 1, every data element of ASPC needs to travel one ($j=0$) or two links ($j>0$). A lower bound on the total number of communication steps is thus given by:

$$\lceil \frac{\text{total number of links that must be crossed}}{\text{total number of links available}} \rceil$$

$$= \lceil \frac{2(1+2(n-1))2^n}{n2^n} \rceil = \lceil \frac{2(1+2(n-1))}{n} \rceil = 4 \quad \square$$

In the following, we show an optimal algorithm which finishes half of the communication, $(i|j) \rightarrow (i+2^j|j)$, in two steps. The other half of the communication $(i|j) \rightarrow (i-2^j|j)$ are performed similarly.

Definition 1 *A 2^n-by-n outgoing schedule table ϕ for pattern $(i|j) \rightarrow (i+2^j|j)$ is defined as:*

$$\phi(i)(j) = H_n(\frac{i}{2^{j-1}}2^{j-1} + 2^j - 1),\ \forall i \in [0, 2^n-1], j \in [0, n-1].$$

From Theorem 1, $\phi(i)(j)$ contains one of the two links that $(i|j)$ should go through in order to reach $(i+2^j|j)$. Lemma 5 tells us that for fixed i, links $\phi(i)(j), j \in [0, n-1]$ are disjoint. We can send all n local elements on node i along links $\phi(i)(j), j \in [0, n-1]$, without conflict.

Similarly, we can define an outgoing table for communication $(i|j) \rightarrow (i-2^j|j)$.

Definition 2 *A 2^n-by-n outgoing schedule table φ for pattern $(i|j) \rightarrow (i-2^j|j)$ is defined as: $\varphi(i)(j) = H_n(\frac{i}{2^{j-1}}2^{j-1} - 2^{j-1} - 1),\ \forall i \in [0, 2^n-1], j \in [0, n-1]$.*

At each communication step, our schedule always puts the jth element of the source node in the jth location of destination node.

Definition 3 *A 2^n-by-n incoming schedule table ϕ' is defined relative to the outgoing schedule table ϕ. If element $(i|j)$ goes out through link $\phi(i)(j)$, the element coming from link $\phi'(i)(j)$ of node i will be put in location j.*

For simplicity, we will use $\phi_i(j), \phi'_i(j)$, and $\varphi_i(j)$ to denote $\phi(i)(j), \phi'(i)(j)$ and $\varphi(i)(j)$, respectively.

If an element on node i is sent through link k, the address of the destination node will be *gray-to-binary(binary-to-gray*$(i) \oplus k$*)*, denoted as i^k.

Lemma 8 $\quad i^k = \frac{i}{2^{k+1}}2^{k+1} + 2^{k+1} - 1 - (i \text{ mod } 2^{k+1})$.

Lemma 9 $\quad \varphi_i(j) = \phi_{2^n-1-i}(j)$.

Lemmas 8 and 9 follows from the reflection property of binary-reflected Gray code.

Theorem 3 $\quad \phi'_i(j) = \varphi_i(j),\ \forall i \in [0, 2^n-1], j \in [0, n-1]$.

At the first communication step, element $(i|j)$ goes through $i^{\phi_i(j)}$. The other link that $(i|j)$ has to go through in order to reach its destination happens to be the same as $\phi_{i^{\phi_i(j)}}(j)$. So at second step, if node $i^{\phi_i(j)}$ sends its jth element (from $(i|j)$) along link $\phi_{i^{\phi_i(j)}}(j)$, element $(i|j)$ will reach its destination.

Theorem 4 $\quad H_n(\frac{i}{2^{j-1}}2^{j-1} + 2^{j-1} - 1) = \phi_{i^{\phi_i(j)}}(j)$.

The proofs for Theorems 3 and 4 use Definition 1, Lemmas 8 and 9, and consider two cases (i mod 2^j) $< 2^{j-1}$ or (i mod 2^j) $> 2^{j-1}$ separately.

Theorem 5 *The optimal schedule for all-to-some personalized communication $(i|j) \rightarrow (i+2^j|j)$, $\forall j \in [0, n-1]$ is given by $\phi_i(j)$ and $\varphi_i(j)$,$i \in [0, 2^n-1], j \in [0, n-1]$.*

Given such a schedule, the optimal algorithm is:

- *Step 1: Every node i sends element $j \in [0, n-1]$ through link $\phi_i(j)$; and put element coming from link $\varphi_i(j), j \in [0, n-1]$ into location j,*

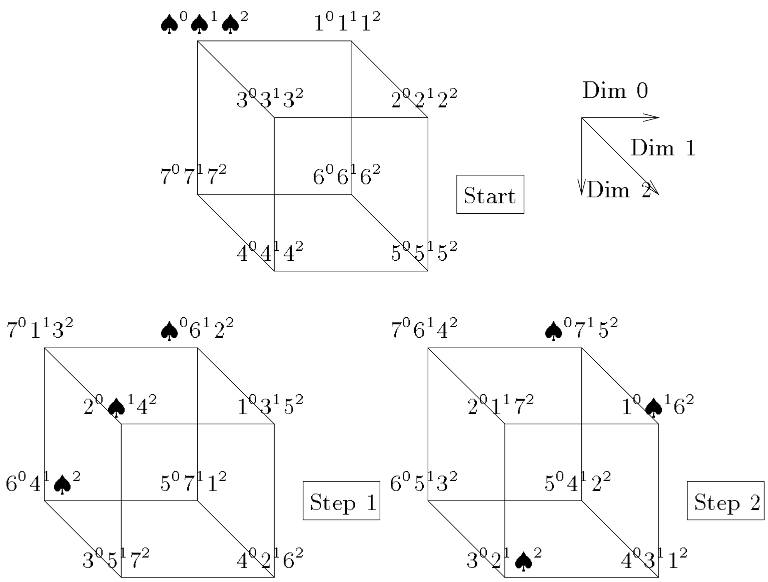

Figure 1. **Data movement for all-to-some personalized communication.**

	Processor i							
Element j	0	1	2	3	4	5	6	7
0	0	1	0	2	0	1	0	2
1	1	0	2	0	1	0	2	0
2	2	2	1	1	2	2	1	1

Table 1. **Schedule $\phi_i(j)$ for $n = 3$.**

- *Step 2: Repeat Step 1, except $j \in [1, n-1]$.*

The route that element $(i|j)$ takes is $(i|j) \rightarrow (i^{\phi_i(j)}|j) \rightarrow ((i^{\phi_i(j)})^{\phi_{i^{\phi_i(j)}}(j)}|j) = (i + 2^j|j)$.

Figure 1 shows the data movement for $n = 3$. The data starting at node i at location j are indicated by i^j. The data starting at node 0 are denoted with a $\spadesuit$. Tables 1 and 2 illustrate the schedule ϕ for $n = 3$ and 5, respectively. From Lemma 9, $\varphi_i(j) = \phi_{2^n-1-i}(j)$ and is not shown.

Note that the schedule can be precomputed and each node only needs to store its n entries.

4. Application of ASPC to $\pm 2^b$-Descend Computations

In this section, we consider an important class of parallel algorithms called $\pm 2^b$-descend [4], and show how to pipeline communication and computation to expose the ASPC communication pattern. We then apply our optimal ASPC algorithm to reduce the communication complexity of these algorithms.

The class of $\pm 2^b$-descend computations is defined in [4] in the following generic form.

Algorithm PM2B-DESEND
1) begin
2) for b = $\log N - 1$ downto 0 do
3) $\quad a[i] \leftarrow f_{b,i}(a[i], a[i + 2^b \bmod N],$
$\qquad a[i - 2^b \bmod N]), 0 \leq i \leq N-1$
4) end

Examples of $\pm 2^b$-descend include those studied in [4], such as odd-even merge, cyclic-shift permutation, and parallel-prefix computation. Another interesting example is Parallel Cyclic Reduction (PCR) [2] which is a well-known method for solving tridiagonal systems.

In [4], a general algorithm for the class of $\pm 2^b$-descend computations requiring $O(\log N)$ communication steps on SIMD hypercubes was developed. However, the algorithm assumes that the number of data elements is the same as the number of hypercube nodes, and furthermore, the algorithm performs three times as much redundant computations of the original computation at each iteration. For computations such as PCR, which has non trivial computation on each data element at each iteration, the three-fold redundant computation is not acceptable.

For large applications on modern massively parallel processors, the problem size M is typically much larger than the number of nodes N. Directly applying the method in [4] thus will result in $2 \cdot \log N \cdot \frac{M}{N}$ communication steps, in addition to the redundant computations. In the following, we describe how to pipeline the communication and computation for the elements of a node, and make use of the ASPC algorithm developed in the previous section.

Assuming that there are M data elements and N nodes in

	Processor i																														
Element j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0	4
1	1	0	2	0	1	0	3	0	1	0	2	0	1	0	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0	4	0
2	2	2	1	1	3	3	1	1	2	2	1	1	4	4	1	1	2	2	1	1	3	3	1	1	2	2	1	1	4	4	1	1
3	3	3	3	3	2	2	2	2	4	4	4	4	2	2	2	2	3	3	3	3	2	2	2	2	4	4	4	4	2	2	2	2
4	4	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	4	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3

Table 2. **Schedule $\phi_i(j)$ for $n = 5$.**

the hypercube. Each node will thus have $\frac{M}{N}$ data elements. The successive $\log M$ iterations in the $\pm 2^b$-descend computations can therefore be split into two stages. The inter-node stage contains $\log N$ iterations, and the communication are uniformly inter-node. The remaining $\log \frac{M}{N}$ iterations form the local stage, containing mixed local data movement and inter-node communication.

We can pipeline the communication and computation for the data elements on each node to expose the ASPC communication pattern. The key observation is that during the first iteration of the inter-node stage, the first elements of each node can perform the communication, i.e. fetch their $\pm 2^{\log M - 1} = \pm \frac{M}{2}$ partners and then perform the computation. Afterwards, all these "first" elements can move on to the next iteration and communicate with their $\pm 2^{\log M - 2} = \pm \frac{M}{4}$ partners, which are also "first" elements on their respective nodes and thus have finished the first iteration. In the meantime, the second element on every node can perform the required communication for the first iteration, i.e. with their $\pm 2^{\log M - 1} = \pm \frac{M}{2}$ partners. The point is that this communication can be overlapped with that for the second iteration of the first element using our ASPC algorithm, since they have different offsets.

Now it is straightforward how to repeat this process. At the next step, the communication for the third, second, and first iterations for the first, second, and third elements on each node can be overlapped. After $\log N$ steps, the available communication that can be scheduled together effectively form a full all-to-some personalized communication.

After $\frac{M}{N}$ steps, the last element on each processor just finishes communicating with its $\pm \frac{M}{2}$ partner, and will have the largest offset during the rest of the iterations. Since the offset is halved at each iteration, there can be at most $\log M - 1$ iterations left. Therefore, the total number of ASPC thus scheduled is $4(\log M + \frac{M}{N} - 1)$. It is clear that such pipelining communication and computation induce no redundant computation.

The above algorithm can be effectively derived via loop transformations on the original $\pm 2^b$-descend algorithm after first splitting the node address index loop and the local memory address index loop, and can therefore be automated by a compiler, as we have previously demonstrated in [1].

5. Conclusions

We have described an optimal algorithm for performing the all-to-some personalized communication (ASPC) on Boolean n-cubes with all-port communication. The algorithm effectively gives an optimal schedule for emulating PM2I networks on hypercubes under the binary-reflected Gray code encoding. To the best of our knowledge, this is also the first optimal algorithm for ASPC.

We have also studied the class of $\pm 2^b$-descend parallel computations for large number of data elements M relative to the number of hypercube nodes N. We have shown that the communication and computation for the data elements on each node can be pipelined to expose the ASPC pattern, and thus make use of our optimal ASPC algorithm to reduce the communication steps from $2 \cdot \log N \cdot \frac{M}{N}$ to $4(\log M + \frac{M}{N} - 1)$.

In [4], the class of $\pm 2^b$-ascend computations was also defined to be simply reversing the iteration order. It was claimed to be harder than $\pm 2^b$-descend computations, and an $O(\log^2 N / \log \log N)$ algorithm was developed. Our technique of pipelining communication and computation combined with the use of our optimal ASPC algorithm can maintain the constant communication complexity achieved for $\pm 2^b$-descend in a straightforward manner.

References

[1] M. Chen and Y. Hu. Optimizations for compiling iterative spatial loops to massively parallel machines. In *Languages and Compilers for Parallel Computing, Lecture Notes in Computer Science 757*. Springer–Verlag, 1993.

[2] R. W. Hockney and C. Jesshope. *Parallel Computers*. Adam Hilger, 1981.

[3] S. L. Johnsson. Communication efficient basic linear algebra computations on hypercube architectures. *J. Parallel Distributed Computing*, 4(2):133–172, April 1987.

[4] D. Nassimi. Parallel algorithms for the classes of $\pm 2^b$ DESEND and ASEND computations on SIMD hypercube. *IEEE Trans. Para. and Dist. Sys.*, 4(12):1372 – 1381, December 1993.

[5] E. Reingold, J. Nievergelt, and N. Deo. *Combinatorial Algorithms*. Prentice-Hall, Englewood Cliffs. NJ, 1977.

[6] H. J. Siegel. *Interconnection Networks for Large Scale Parallel Processing*. Lexington Books, 1985.