Yale University
Department of Computer Science

Shuffle Permutations on Boolean Cubes

S. Lennart Johnsson and Ching-Tien Ho

YALEU/DCS/TR-653
October 1988

This work has in part been supported by the Office of Naval Research under contract N00014-86-K-0310. Approved for public release: distribution is unlimited.

Shuffle Permutations on Boolean Cubes

S. Lennart Johnsson[1] and Ching-Tien Ho
Department of Computer Science
Yale University
New Haven, CT 06520
Johnsson@think.com, Ho@cs.yale.edu

Abstract. In this paper we prove lower bounds and present algorithms optimal within a small constant factor for *generalized shuffle permutations* on Boolean cubes. A *generalized shuffle permutation* is a permutation where a global address $(a_{q-1}a_{q-2}\ldots a_0)$ receives its new content from a global address $(a_{\delta(q-1)}a_{\delta(q-2)}\ldots a_{\delta(0)})$, with $\delta(\alpha_0) = \alpha_1, \delta(\alpha_1) = \alpha_2, \ldots, \delta(\alpha_{\sigma-1}) = \alpha_0$ for $\alpha_i \in \{0, 1, \ldots, q-1\}, \sigma \leq q$. For packet switched communication restricted to one port at a time per processor, the minimum number of communications in sequence is equal to the number of address bits to which the permutation is applied. The data transfer time of the permutation is proportional to the size of the data set per processor *and* the number of address bits being part of the permutation. With concurrent communication on all ports of every processor the data transfer time is proportional to the size of the data set per processor. Depending on communication capability, message size, cube size, data transfer rate, and communication start-up time, different algorithms must be chosen for a communication time optimal within a small constant factor. The analysis is verified by experimental results on the Intel iPSC/1.

1 Introduction

A dimension permutation is a permutation defined on the bits of the address field, while an arbitrary permutation is a permutation on the address field. There are $(\log_2 M)!$ possible dimension permutations compared to $M!$ arbitrary permutations for an address space of size M. Examples of dimension permutations are k-shuffle/unshuffle permutations, matrix transposition [5], [8], bit-reversal [10], and conversion between various data structures, such as consecutive and cyclic storage [5], [8]. Shuffle operations can be used to reconfigure a two dimensional partitioning to a three dimensional partitioning of a matrix for multiplication with maximum concurrency [7]. They may also be used for data (re)alignment for certain Fast Fourier Transform algorithms [6], [10].

The main focus of this paper is on lower bounds for stable generalized shuffle permutations for communication restricted to one port at a time per processor, *one-port communication*, and concurrent communication on all ports, *n-port communication*, and optimal algorithms for *n-port communication*. We also present new optimal algorithms for *one-port communication* demonstrating that for some situations there is a slight reduction in the communication complexity compared to the previously known algorithms. A *stable* permutation is a permutation where

[1]Also with Dept. of Electrical Engineering, Yale Univ., and Thinking Machines Corp., Cambridge, MA 02142.

the same processors are used before and after the permutation. A generalized shuffle permutation corresponds to a cyclic shift on a properly permuted index set. Dimension permutations, a generalization of shuffle permutations are treated in [3]. *Unstable* dimension permutations are treated in [4]. Stable dimension permutations have also been studied by Flanders [1] on mesh-connected array processors, and by Swarztrauber [14] on Boolean cubes. Flanders and Swarztrauber give almost identical algorithms for communication restricted to one port at a time. The notation and definitions used throughout the paper are introduced in Section 2. In Section 3 we discuss lower bounds. Algorithms are described in Section 4, and implementations on the Intel iPSC are described in Section 5. We conclude with a few remarks in Section 6.

2 Preliminaries

2.1 Address spaces and Boolean cubes

The nodes in a Boolean n-cube can be given addresses such that adjacent nodes differ in precisely one bit. The number of nodes is $N = 2^n$.

Definition 1 The *Hamming distance* between two numbers a and a' with binary encodings $a = (a_{q-1}a_{q-2}\ldots a_0)$ and $a' = (a'_{q-1}a'_{q-2}\ldots a'_0)$ is $Hamming(a,a') = \sum_{i=0}^{q-1}(a_i \oplus a'_i)$.

The *distance* between two nodes x and y in a Boolean n-cube is $Hamming(x,y)$. The number of nodes at distance j from any node is $\binom{n}{j}$. The number of disjoint paths between any pair of nodes x and y is n. $Hamming(x,y)$ paths are of length $Hamming(x,y)$ and $n - Hamming(x,y)$ paths are of length $Hamming(x,y) + 2$ [11]. $||a||$ denotes the number of 1-bits in the binary representation of a, i.e., $||a|| = Hamming(a,0)$. $|S|$ is the cardinality of set S.

The *machine address space* is $\mathcal{A}$ and the *logic address space* is $\mathcal{L}$. The *machine address space* $\mathcal{A} = \{(a_{q-1}a_{q-2}\ldots a_0)|\ a_i = 0,1;\ 0 \le i < q\}$ is the Cartesian product of the *processor address space* and *local storage address space*. The processor address space requires n bits, or *dimensions*. The storage per node is 2^{q-n} elements. Of the machine address space, the n low-order dimensions are used for processor addresses, and the $q-n$ high-order dimensions are used for local storage addresses:

$$(\underbrace{a_{q-1}a_{q-2}\ldots a_n}_{s}\underbrace{a_{n-1}a_{n-2}\ldots a_0}_{p}).$$

The set of *machine dimensions* is $\mathcal{Q} = \{0,1,\ldots,q-1\}$, the set of *processor dimensions* is $\mathcal{Q}_p = \{0,1,\ldots,n-1\}$, and the set of *local storage dimensions* is $\mathcal{Q}_s = \{n,n+1,\ldots,q-1\}$.

The *logic address space* $\mathcal{L} = \{(w_{m-1}w_{m-2}\ldots w_0)|w_i = 0,1;\ 0 \le i < m\}$ encodes a set of $|\mathcal{L}| = 2^m$ elements. The set of *logic dimensions* is $\mathcal{M} = \{0,1,\ldots,m-1\}$. The relationship between the number of processor dimensions, the local storage dimensions, and the number of logic dimensions is arbitrary. For instance, if $m \le q-n$ and $m \le n$, then the entire data set can be allocated to the local storage of a single processor, or across processors with one element per processor using 2^m processors.

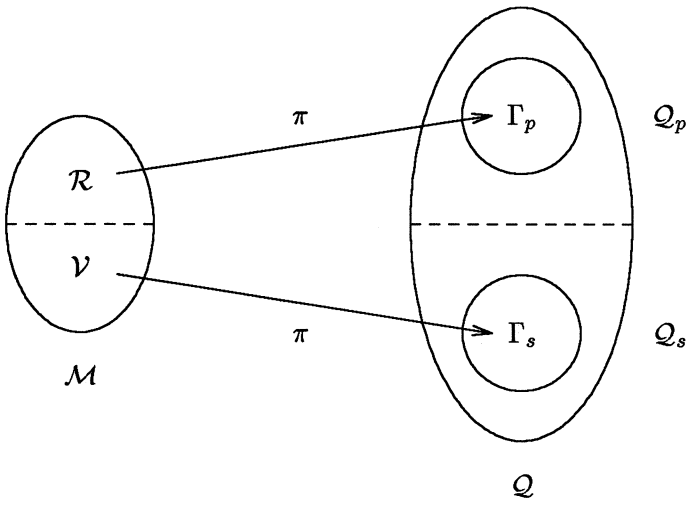

Figure 1: The relationship between the sets of address dimensions.

Definition 2 A *dimension allocation function*, π, is a one-to-one mapping from the set of logic dimensions, $\mathcal{M}$, to the set of machine dimensions, $\mathcal{Q}$; $\pi : \mathcal{M} \rightarrow \mathcal{Q}$.

Let $\mathcal{R} = \{r_{m_p-1}, r_{m_p-2}, \ldots, r_0\}$ be the set of logic dimensions mapped to *processor dimensions*, i.e., $\pi(i) \in \mathcal{Q}_p$, $\forall i \in \mathcal{R}$ and $\mathcal{V} = \{v_{m_s-1}, v_{m_s-2}, \ldots, v_0\}$ be the set of logic dimensions mapped to *local storage dimensions*, i.e., $\pi(i) \in \mathcal{Q}_s$, $\forall i \in \mathcal{V}$. Then, $|\mathcal{R}| = m_p \leq n$, $|\mathcal{V}| = m_s \leq q - n$, $\mathcal{R} \cup \mathcal{V} = \mathcal{M}$, $\mathcal{R} \cap \mathcal{V} = \phi$, $m_p + m_s = m$. $\Gamma_p = \{\pi(i) | \forall i \in \mathcal{R}\}$ and $\Gamma_s = \{\pi(i) | \forall i \in \mathcal{V}\}$, are the sets of processor and local storage dimensions used for the allocation of elements: $\Gamma = \Gamma_p \cup \Gamma_s$. The inverse of the dimension allocation function π^{-1} is a mapping: $\Gamma \rightarrow \mathcal{M}$, such that $\pi^{-1} \circ \pi = I$, where I is the identity function. Figure 1 illustrates the relationships between the different sets and the allocation function. We will refer to the dimensions in $\mathcal{Q}_p$ (processor dimensions) as *real dimensions* and the dimensions in $\mathcal{Q}_s$ (local storage dimensions) as *virtual dimensions.*

Definition 3 The *real distance* between two locations with addresses a and a', $a, a' \in \mathcal{A}$, is $Hamming_r(a, a') = \sum_{i=0}^{n-1}(a_i \oplus a'_i)$ and the *virtual distance* between a and a' is $Hamming_v(a, a') = \sum_{i=n}^{q-1}(a_i \oplus a'_i)$.

Lemma 1 $Hamming(a, a') = Hamming_r(a, a') + Hamming_v(a, a')$.

If the m_p lowest-order logic dimensions are mapped to processor dimensions, then the allocation is *cyclic*; if the m_p highest-order logic dimensions are mapped to processor dimensions, then the allocation is *consecutive* [5]. We use the notation $(v_{m_s-1} v_{m_s-2} \ldots v_0 | r_{m_p-1} r_{m_p-2} \ldots r_0)$ for w when we want to stress the separation of logic dimensions mapped to real and virtual dimensions. Element w is allocated to location a, where $a_i = w_{\pi^{-1}(i)}$, if $i \in \Gamma$, and $a_i = 0$, otherwise. We arbitrarily define the unassigned address fields to be 0.

2.2 Classification of dimension permutations

A dimension permutation implies a change in allocation from $\pi^b(\mathcal{M})$, before the permutation, to $\pi^a(\mathcal{M})$, after it. Let $\mathcal{R}^b$ be the set of logic dimensions mapped to processor dimensions before the permutation. Let $\mathcal{R}^a$ be the set of logic dimensions mapped to processor dimensions after it. $\mathcal{V}^b$ and $\mathcal{V}^a$ are defined similarly. The sets of machine dimensions used before and after the permutation are denoted $\Gamma^b = \Gamma_p^b \cup \Gamma_s^b$ and $\Gamma^a = \Gamma_p^a \cup \Gamma_s^a$, where $\Gamma_p^b, \Gamma_p^a \subseteq \mathcal{Q}_p$ and $\Gamma_s^b, \Gamma_s^a \subseteq \mathcal{Q}_s$. Clearly $|\Gamma^b| = |\Gamma^a|$, since the number of elements is conserved. If $\Gamma^b = \Gamma^a$ (i.e., $\Gamma_p^b = \Gamma_p^a$ and $\Gamma_s^b = \Gamma_s^a$) then the permutation is *stable*. Otherwise, it is *unstable*. Note that we classify the case where $\Gamma_p^b = \Gamma_p^a$ and $\Gamma_s^b \neq \Gamma_s^a$ as an unstable permutation. The restriction of *stable* permutations to use the same local address space before and after the permutation is made for notational convenience. The algorithms for stable permutations presented in this paper also work for the case $\Gamma_p^b = \Gamma_p^a$ and $\Gamma_s^b \neq \Gamma_s^a$ with the same complexity as in the stable case, if the time for local data rearrangement is ignored. For convenience, let $\Gamma = \Gamma^b = \Gamma^a$, $\Gamma_p = \Gamma_p^b = \Gamma_p^a$ and $\Gamma_s = \Gamma_s^b = \Gamma_s^a$.

Definition 4 A *stable generalized shuffle permutation* (GSH), *gsh*, on a subset of the machine address space Γ is a one-to-one mapping $\Gamma \rightarrow \Gamma$ with $\delta = \pi^a \circ \pi^{-b}$ (π^{-b} denotes $(\pi^b)^{-1}$) restricted such that $\delta_\sigma(\alpha_0) = \alpha_1, \delta_\sigma(\alpha_1) = \alpha_2, \ldots, \delta_\sigma(\alpha_{\sigma-1}) = \alpha_0$, $\alpha_i \neq \alpha_j$, $i \neq j$, $\alpha_i, \alpha_j \in \mathcal{J}$, $0 \leq i, j < \sigma$, and $\delta_\sigma(i) = i$, $\forall i \in \Gamma - \mathcal{J}$. The *index set* $\mathcal{J}$ of the permutation is the subset of Γ such that $\{i | \delta(i) \neq i\} = \mathcal{J}$. The *order* of the permutation is $\sigma = |\mathcal{J}|$. Alternatively, one can define a *generalized shuffle permutation*, *gsh'*, on the set of logic dimensions, i.e., $gsh' : \mathcal{M} \rightarrow \mathcal{M}$ (for the stable case) with $gsh' = \pi^{-a} \circ \pi^b$.

For convenience, we let $\mathcal{J} = \{\alpha_0, \alpha_1, \ldots, \alpha_{\sigma-1}\}$ be an ordered index set with the order implied. The permutation function δ, or *gsh*, applies to the subset of machine dimensions. For convenience, we use $\delta(a)$, or $gsh(a)$, $a \in \mathcal{A}$, to denote $(a_{\delta(q-1)} a_{\delta(q-2)} \ldots a_{\delta(0)})$, where δ is extended to a function of $\mathcal{Q} \rightarrow \mathcal{Q}$ with $\delta(i) = i$, $i \in \mathcal{Q} - \mathcal{J}$. We require that the GSH forms a single cycle over the machine dimensions in the index set $\mathcal{J}$, i.e., it corresponds to a right cyclic shift of the ordered set $\mathcal{J}$. In a GSH a logic dimension k assigned to machine dimension $i = \pi^b(k)$ is reassigned to machine dimension $\delta(i) = \pi^a(k)$. Following the definition, we have the corollary below.

Corollary 1 *In a GSH, a global address $a = (a_{q-1} a_{q-2} \ldots a_0)$ receives its content from the global address $\delta(a) = (a_{\delta(q-1)} a_{\delta(q-2)} \ldots a_{\delta(0)})$ and sends its contents to the global address $\delta^{-1}(a) = (a_{\delta^{-1}(q-1)} a_{\delta^{-1}(q-2)} \ldots a_{\delta^{-1}(0)})$, if it contains an element originally.*

Proof: Assume $q = m$ first. Before a GSH, element $w = (w_{m-1} w_{m-2} \ldots w_0)$ is allocated to global address $a = (a_{m-1} a_{m-2} \ldots a_0)$. Then, $w_i = a_{\pi^b(i)}$ or $a_i = w_{\pi^{-b}(i)}$. After a GSH, element w is relocated to global address $a' = (a'_{m-1} a'_{m-2} \ldots a'_0)$. Then, $w_i = a'_{\pi^a(i)}$ or $a'_i = w_{\pi^{-a}(i)}$. Since,

$$a'_i = w_{\pi^{-a}(i)} = w_{\pi^{-b} \circ \pi^b \circ \pi^{-a}(i)} = a_{\pi^b \circ \pi^{-a}(i)} = a_{\delta^{-1}(i)},$$

global address $(a_{m-1} a_{m-2} \ldots a_0)$ sends its element to global address $(a_{\delta^{-1}(m-1)} a_{\delta^{-1}(m-2)} \ldots a_{\delta^{-1}(0)})$. For $q > m$, we consider the subset of global address with $a_i = 0$ for all $i \in \mathcal{Q} - \Gamma$. ∎

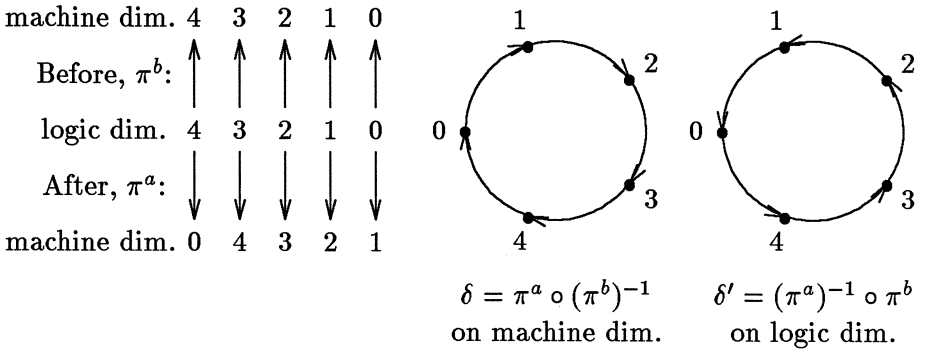

Figure 2: The definition of δ and δ', and cycles formed by traversing δ and δ'.

Definition 5 The *identity permutation* I is defined by $\delta_0(i) = i, \forall i \in \Gamma$ or $\mathcal{J} = \phi$.

The order of the identity permutation is 0. A subscript σ on δ, δ_σ, is used to denote the order of the GSH being σ.

Figure 2 shows a shuffle permutation in which machine dimension i becomes $\delta(i) = (i + 1) \bmod 5$. Global address $(a_4a_3a_2a_1a_0)$ sends its contents to global address $(a_{\delta^{-1}(4)}a_{\delta^{-1}(3)}a_{\delta^{-1}(2)}a_{\delta^{-1}(1)}a_{\delta^{-1}(0)}) = (a_3a_2a_1a_0a_4)$. Note that $\delta = (\delta')^{-1}$, if $\pi^b(i) = i, \forall i \in \mathcal{M}$, i.e., if $\pi^b = I$ is the identity function.

Definition 6 A *full-cube permutation* (FCP) is a GSH for which the data set is allocated to all real processors (but not necessarily the entire memory): $\Gamma_p = \mathcal{Q}_p$. An *extended-cube permutation* (ECP) is a GSH for which the data set only occupies a fraction of the cube: $\Gamma_p \subset \mathcal{Q}_p$.

Definition 7 A *real* GSH is a GSH such that $\mathcal{J} \subseteq \Gamma_p$ and $\mathcal{J} \neq \phi$. A *virtual* GSH is a GSH such that $\mathcal{J} \subseteq \Gamma_s$ and $\mathcal{J} \neq \phi$. A GSH that is neither real nor virtual and $\mathcal{J} \neq \phi$ is a *mixed* GSH.

Definition 8 A *shuffle permutation* sh is a GSH such that $\mathcal{J} = \{0, 1, \ldots, \sigma - 1\}$ (i.e., $\delta(j) = (j + 1) \bmod \sigma$, $0 \leq j < \sigma$), and an *unshuffle permutation* sh^{-1} is a GSH such that $\mathcal{J} = \{\sigma - 1, \sigma - 2, \ldots, 0\}$ (i.e., $\delta(j) = (j - 1) \bmod \sigma$). A *k-shuffle* is the permutation $sh^k = sh \circ sh^{k-1}$.

A *real* GSH preserves the local address map. Data is moved between processors in the subcube defined by the set of dimensions Γ_p. All processors in this set have identical address maps. A *real* GSH is a generalization of the *collinear planar exchanges* described in [1]. A *virtual* GSH (called *vertical exchanges* in [1]) only involves local data movement, or a change of local address map in the set of processors defined by the set Γ_p. The same change is made for every processor in this set. A *mixed* GSH (*planar-vertical exchanges* [1]) involves both processor and local storage dimensions in the index set $\mathcal{J}$.

The reason for distinguishing between the real and virtual permutations is that access times for local storage references are usually considerably faster than communication between processors. The former can often, with good approximation, be ignored. In standard random access

memories (RAM) the access time is independent of the address, however, the interprocessor access time is often a function of the distance. In a packet switched communication system the minimum time for a communication is proportional to the number of links that need to be traversed. The actual time may be higher due to contention for communication links. In a bit-serial pipelined system the time for a communication is independent of the distance for a given machine, in the absence of contention. The time increases in proportion to the logarithm of the machine size. The contention is often a function of the number of dimensions.

Definition 9 The *real order* σ_p of a GSH on a machine address space is $|\{j|j \neq \delta(j), j \in \Gamma_p\}| = |\mathcal{J} \cap \Gamma_p|$, and its *virtual order* σ_s is $|\{j|j \neq \delta(j), j \in \Gamma_s\}| = |\mathcal{J} \cap \Gamma_s|$.

2.3 Base algorithms

All Boolean cube algorithms we present are based on sequences of exchange operations between pairs of processors through communication in the same two dimensions for all pairs.

Definition 10 A *dimension exchange* function $E(i,j)$ is a GSH with $\mathcal{J} = \{i,j\}$. Data is exchanged between pairs of locations (global addresses) as defined by

$$(a|a_i = 0, a_j = 1) \Longleftrightarrow (a|a_i = 1, a_j = 0).$$

A shuffle permutation on Γ can be performed as a sequence of $\sigma - 1$ dimension exchanges on adjacent machine dimensions. Similarly, a GSH can be performed as a sequence of $\sigma - 1$ dimension exchanges on adjacent dimensions in the index set $\mathcal{J}$.

Lemma 2 *A generalized shuffle permutation $gsh(a)$ of order σ can be realized by a sequence of dimension exchanges on a properly ordered index set $\mathcal{J}$.*

$$\begin{aligned} gsh(a) &= E(\alpha_{(i+2) \bmod \sigma}, \alpha_{(i+1) \bmod \sigma}) \circ \cdots \circ E(\alpha_{(i-1) \bmod \sigma}, \alpha_{(i-2) \bmod \sigma}) \circ E(\alpha_i, \alpha_{(i-1) \bmod \sigma}) \\ &= \coprod_{j=i}^{i+2} E(\alpha_j, \alpha_{(j-1) \bmod \sigma}) \end{aligned}$$

Lemma 2 is easily proved by induction. The dimension exchange operations are performed in a right to left order.

Remark: The starting dimension for the exchange sequence is arbitrary, but the direction (increasing or decreasing dimension) is important.

A generalized shuffle permutation can also be realized by a sequence of exchange operations between a fixed dimension and cyclically and monotonically increasing dimensions.

Lemma 3 *A generalized shuffle permutation $gsh(a)$ can be realized by a sequence of dimension exchanges between an arbitrary, fixed dimension and a sequence of dimensions of a properly ordered index set $\mathcal{J}$.*

$$gsh(a) = E(\alpha_i, \alpha_{(i-1) \bmod \sigma}) \circ \cdots \circ E(\alpha_i, \alpha_{(i+2) \bmod \sigma}) \circ E(\alpha_i, \alpha_{(i+1) \bmod \sigma}) = \prod_{j=i+1}^{i-1} E(\alpha_i, \alpha_j)$$

Lemma 3 can be proved by induction. Both the "$\prod$" "$\coprod$" signs denote the composition of a sequence of functions in which the index of the functions are substituted by $i \bmod \sigma, (i+1) \bmod \sigma, \ldots, i' \bmod \sigma$ for $\prod_{j=i}^{i'}$, and $i \bmod \sigma, (i-1) \bmod \sigma, \ldots, i' \bmod \sigma$ for $\coprod_{j=i}^{i'}$.

A dimension exchange $E(i,j), i \neq j$, requires $Hamming_r(a', a'')$ routing cycles, where $a' = a''$ except $a'_i \neq a''_i$ and $a'_j \neq a''_j$. It is 2 if $i,j \in \Gamma_p$; 0 if $i,j \in \Gamma_s$; and 1, otherwise. If $i,j \in \Gamma_p$, then the set of processors is partitioned into four groups with respect to the values of a_i and a_j. The groups are labeled {00, 11, 01, 10}. Processors in the 10-group exchange data with processors in the 01-group. The exchange operation can be realized by *two* nearest-neighbor communications, with processors in the 00- and 11-groups as intermediate nodes. The minimum temporary storage required at the intermediate processor is equal to the size of a message. With *one-port communication* the time complexity is the same whether or not a message is forwarded in the second dimension before the next message in the first dimension is sent. For simplicity in the description below we assume that all K elements per processor are communicated in one dimension before communication in the next dimension takes place. Our description also corresponds to the case where the entire algorithm is repeated $\frac{K}{B}$ times with a packet size B. The temporary storage requirement in intermediate nodes is equal in size to the storage per processor in the initial and final stages. If one of the dimensions is a virtual dimension, then one nearest-neighbor communication will suffice. If both dimensions are virtual dimensions, then no communication is required.

In the case of a *mixed* GSH, one should pick the fixed dimension $\alpha_i \in \mathcal{J} \cap \Gamma_s$. Each exchange operation then becomes *one* nearest-neighbor communication step. For a *real* GSH $\mathcal{J} \cap \Gamma_s = \phi$, one can extend the index set $\mathcal{J}$ by a virtual dimension $v \in \Gamma_s$ and use the virtual dimension as the fixed dimension. Terminating the exchange sequence by repeating the first dimension exchange (between dimension v and a dimension in the set $\mathcal{J}$) yields the desired GSH.

Lemma 4 $gsh(a) = E(v,\alpha_i) \circ E(v, \alpha_{(i-1) \bmod \sigma}) \circ \cdots \circ E(v, \alpha_{(i+1) \bmod \sigma}) \circ E(v, \alpha_i) = \prod_{j=i}^{i} E(v, \alpha_j)$.

Proof: By Lemma 3, the σ rightmost exchange operations (i.e., excluding the leftmost one) realizes a GSH on the ordered set $\mathcal{J}' = \{v, \alpha_i, \alpha_{(i+1) \bmod \sigma}, \ldots, \alpha_{(i-1) \bmod \sigma}\}$. This means that $\delta(\alpha_j) = \alpha_{(j+1) \bmod \sigma}$ for all $0 \leq j < \sigma, j \neq (i-1) \bmod \sigma$, $\delta(\alpha_{(i-1) \bmod \sigma}) = v$, and $\delta(v) = \alpha_i$. But, the exchange $E(v, \alpha_i)$ implies $\delta(v) = v$ and $\delta(\alpha_{(i-1) \bmod \sigma}) = \alpha_i$. ∎

If $v \in \mathcal{Q}_s$ is a virtual dimension, then one can partition the local memory of each processor into two equal parts with respect to dimension v, i.e., the one of a global address with $a_v = 0$ (first half) and $a_v = 1$ (the second half). In Lemma 4, if $v \in \Gamma_s$, then the original data set is partitioned into two parts with respect to a_v and only one part needs to be moved for an exchange operation. If $v \in \mathcal{Q}_s - \Gamma_s$, then all the original data set is in the first half with respect to virtual dimension v (i.e., $a_v = 0$). By locally moving the data of the processor with $a_{\alpha_{i-1}} = 1$ into the second part, we have $a_v = a_{\alpha_{i-1}}$ for all processors. This means that the last exchange step of Lemma 4, $E(v, \alpha_i)$, which exchanges $a_{\alpha_{(i-1) \bmod \sigma}}$ and a_v (as described in the proof) can be omitted, Corollary 2. After the GSH, the valid data is in the first half if $a_{\alpha_i} = 0$; and in the second half, otherwise. The data volume is twice that of Lemma 4.

Corollary 2 $gsh(a) = E(v, \alpha_{(i-1) \bmod \sigma}) \circ \cdots \circ E(v, \alpha_{(i+1) \bmod \sigma}) \circ E(v, \alpha_i) = \prod_{j=i}^{i-1} E(v, \alpha_j)$, *if* $a_v = a_{\alpha_{i-1}}$.

Note that Lemmas 2 and 3 both require $\sigma - 1$ dimension exchanges for a GSH of order σ. However, Lemma 4 requires $\sigma + 1$ exchanges. Corollary 2 needs σ exchanges. Dimension permutations are also realized by dimension exchanges in [1].

Dimension permutation algorithms can also be obtained by using a matrix transposition algorithm recursively [8], [13], [12], or by using *all-to-all personalized communication* twice [13], [9].

3 The complexity of stable generalized shuffle permutations

For each internode communication, there is an associated transmission time t_c for each element, and a start-up time, or overhead, τ for each communication of a packet of B elements. The packet size that minimizes the communication complexity is B_{opt}. We consider both *one-port communication* and *n-port communication*. In the first case communication is restricted to one port at a time for each processor. In the second case communication can take place on all ports concurrently. The links are assumed to be bidirectional.

The time complexity for the different GSH's are denoted $T^*(ports, \sigma_p, m_p, K)$. The superscript $*$ is either lb for a lower bound, or an algorithm identifier for an upper bound. The first argument for T is the number of ports per processor used concurrently, the second argument the real order of the GSH, the third argument the number of processor dimensions being used, and the last argument the data volume per "allocated" processor, i.e., $K = 2^{m_s}$.

3.1 Some properties of stable generalized shuffle permutations

The following lemma establishes the fact that GSH's only involving a part of the logic address space can be viewed as block permutations.

Lemma 5 *For a GSH of order* σ, $2^{m-\sigma}$ *elements are subject to the same permutation.*

Proof: $|\Gamma - \mathcal{J}| = m - \sigma$. The GSH function δ_σ is defined in the same way for each of the $2^{m-\sigma}$ blocks identified by the corresponding $m - \sigma$ bits of the machine dimensions. ∎

Corollary 3 *For a GSH of real order* σ_p, *the permutation consists of* $2^{m_p - \sigma_p}$ *permutations in subcubes of size* 2^{σ_p}.

If a permutation only involves a subset of the processor dimensions to which data have been allocated, and if a lower bound algorithm is used for each subcube permutation, then performance cannot be gained by using the processor dimensions used for data allocation, but not participating in the permutation.

Lemma 6 *A GSH of real order $\sigma_p < m_p$ cannot be improved by communication in the $m_p - \sigma_p$ processor dimensions that are not included in the GSH, if the original algorithm fully utilizes the bandwidth.*

Proof: Let P and P' be two problems such that $\Gamma_s = \Gamma'_s$ and subject to the same GSH, δ_{σ_p}, on a σ_p-cube and an m_p-cube, respectively. Let T be the communication complexity of a lower bound algorithm A for P, i.e., $T = \frac{B}{L}$, where B is the total bandwidth required and L the available bandwidth per unit time. Suppose there exists an algorithm A' for problem P' with communication complexity $T' < T$. Consider a new problem P'': a GSH δ_{σ_p} on a σ_p-cube with the bandwidth of each cube link, and the data set per node expanded by a factor of $2^{m_p - \sigma_p}$ of the ones in P. Now, map the nodes in P' to nodes in P'' such that the corresponding bits of the σ_p dimensions for problem P' are used to identify processors in P''. Every algorithm for problem P' can be converted to an algorithm for problem P'' with a communication complexity that is at most the same. So, $T = \frac{B}{L} > T' \geq T''$, where T'' is the communication complexity of the corresponding algorithm for P''. However, $T'' \geq \frac{2^{m_p - \sigma_p} B}{2^{m_p - \sigma_p} L} = \frac{B}{L}$, which in turn is equal to T, and we have a contradiction. ∎

Note that Lemma 6 only addresses the communication within the subcube used for the logic address space. It does not address the extended-cube permutation case.

3.2 Lower bounds

A full-cube, real GSH of order σ_p on an n-cube consists of $2^{n-\sigma_p}$ independent GSH's, each of order 2^{σ_p}. If the communication channels in each such subcube are fully utilized, then no additional reduction in the data transfer time is possible by using the remaining $n - \sigma_p$ ports in case of *n-port* communication by Lemma 6.

Lemma 7 *The lower bound for a full-cube, real GSH of order σ_p, $\sigma_p > 0$, on an n-cube is*

$$T^{lb}_{gsh}(1, \sigma_p, n, K) = \begin{cases} \max(\frac{\sigma_p K}{2} t_c, \sigma_p \tau), & \sigma_p \text{ is even,} \\ \max(\frac{\sigma_p K}{2} t_c, (\sigma_p - 1)\tau), & \sigma_p \text{ is odd,} \end{cases}$$

for one-port communication, and

$$T^{lb}_{gsh}(n, \sigma_p, n, K) = \begin{cases} \max(\frac{K}{2} t_c, \sigma_p \tau), & \sigma_p \text{ is even,} \\ \max(\frac{K}{2} t_c, (\sigma_p - 1)\tau), & \sigma_p \text{ is odd,} \end{cases}$$

for n-port communication.

Proof: Consider the minimum number of start-ups first. Let a be such that $a_{\alpha_i} = 0$, if $i \bmod 2 = 0$, and $a_{\alpha_i} = 1$ otherwise, $0 \leq i < \sigma_p$. It follows that $Hamming_r(a, gsh(a)) = \sigma_p$, if σ_p is even; and $\sigma_p - 1$, if σ_p is odd. To show that $\sigma_p - 1$ is the maximum Hamming distance if σ_p is odd, we show that a Hamming distance of σ_p is impossible. This is easily seen since $a_{\alpha_i} \neq a_{\alpha_{(i+1) \bmod \sigma_p}}$, $\forall 0 \leq i < \sigma_p$, is impossible if σ_p is odd.

The minimum data transfer time is bounded from below by the required bandwidth divided by the available bandwidth. By Lemma 6, we can consider the bandwidths for each σ_p-cube. For each $j \in \mathcal{J} \cap \Gamma_p = \mathcal{J}$ with $\delta^{-1}(j) = i$, $i \neq j$, only half of the nodes ($a_i \neq a_j$) need to send elements across cube dimension j. Therefore, the bandwidth requirement for each permutation in subcubes of dimension σ_p is $\sigma_p 2^{\sigma_p - 1} K$. The available bandwidth per routing cycle of a σ_p-cube is 2^{σ_p} for *one-port* and $\sigma_p 2^{\sigma_p}$ for *n-port communication.* ∎

Corollary 4 *The data transfer time of any fixed packet size algorithm of c routing cycles for a full-cube, real GSH is at least* $\frac{cK}{2(c-1)} t_c$ *with n-port communication, and at least* $\frac{c\sigma_p K}{2(c-1)} t_c$ *with one-port communication where all processors are using the same dimension during the same routing cycle.*

Proof: In order to realize the minimum data transfer time described in Lemma 7, all links should be used "effectively" in the sense that all messages are routed through a shortest path, and "evenly" during every routing cycles. However, during the first and last routing cycles, at least half of the cube links can not support effective communication for the GSH. For *one-port communication*, the same argument applies to the first and last routing cycles for any cube dimension used by all processors. ∎

Lemma 8 *The lower bound for a full-cube, mixed GSH of real order* σ_p, $0 < \sigma_p < \sigma$, *on an n-cube is*

$$T_{gsh}^{lb}(1, \sigma_p, n, K) = \max\left(\frac{\sigma_p K}{2} t_c, \sigma_p \tau\right)$$

for one-port communication, and

$$T_{gsh}^{lb}(n, \sigma_p, n, K) = \max\left(\frac{K}{2} t_c, \sigma_p \tau\right)$$

for n-port communication.

Proof: The minimum number of start-ups required is $\max\{Hamming_r(a, gsh(a))\}$, $\forall a \in \mathcal{A}$. Choose any α_j such that $\alpha_j \in \Gamma_p$ and $\alpha_{(j-1) \bmod \sigma} \in \Gamma_s$. Define a such that $a_{\alpha_i} = 0$, if i is even, $0 \leq i < \sigma$; and $a_{\alpha_i} = 1$, otherwise. Moreover, change $a_{\alpha_{(j-1) \bmod \sigma}}$ to be different from a_{α_j} (when σ is odd). Clearly, $Hamming_r(a, gsh(a)) = \sigma_p$ for the a so defined. It is easily seen that $\max_a\{Hamming(a, gsh(a))\} \leq \sigma_p$, since $a_i = a_{\delta(i)}$, $\forall i \in \Gamma - \mathcal{J}$.

The minimum data transfer time can be proved as in Lemma 7. ∎

An extended cube permutation of real order σ_p can be improved by communication in the $n - m_p$ processor dimensions not used for the allocation of the data array. A possible algorithm is a composition of a subcube expansion, full cube permutation, and subcube compression algorithms. The permutation is then performed on a data set reduced by a factor of 2^{n-m_p}. The subcube expansion permutation is of type *one-to-all personalized communication* [2].

Corollary 5 *The lower bound for the communication complexity for an extended-cube GSH ($m_p < n$) of real order σ_p on an n-cube is*

$$T_{gsh}^{lb}(1,\sigma_p,m_p,K) = \begin{cases} \max\left((K+\sigma_p-1)t_c, \frac{\sigma_p \cdot K}{2^{n-m_p+1}}t_c, \sigma_p\tau\right), & \sigma_p \text{ is even or mixed GSH,} \\ \max\left((K+\sigma_p-2)t_c, \frac{\sigma_p \cdot K}{2^{n-m_p+1}}t_c, (\sigma_p-1)\tau\right), & \text{otherwise,} \end{cases}$$

for one-port communication, and

$$T_{gsh}^{lb}(n,\sigma_p,m_p,K) = \begin{cases} \max\left(\left(\frac{K}{n-m_p+\sigma_p}+\sigma_p-1\right)t_c, \frac{K}{2^{n-m_p+1}}t_c, \sigma_p\tau\right), & \sigma_p \text{ is even or mixed GSH,} \\ \max\left(\left(\frac{K}{n-m_p+\sigma_p}+\sigma_p-2\right)t_c, \frac{K}{2^{n-m_p+1}}t_c, (\sigma_p-1)\tau\right), & \text{otherwise,} \end{cases}$$

for n-port communication.

Proof: By Lemma 6, the lower bound for a GSH of real order σ_p on a data set of $2^{m_s+m_p}$ elements on an n-cube is the same as the lower bound of the same GSH of real order σ_p on a data set of $2^{m_s+\sigma_p}$ elements on an $(n-m_p+\sigma_p)$-cube. We now prove the lower bound for the latter problem. The first argument of the *max* function is derived by considering the minimum time required to send out the K elements for any processor that needs to send data, and the propagation delay for the last element sent out. From the proof of Lemma 7, the bandwidth required is $\sigma_p 2^{\sigma_p-1}K$. The "effective" bandwidth available is $\sigma_p 2^{n-m_p+\sigma_p}$ for *n-port communication* and $2^{n-m_p+\sigma_p}$ for *one-port communication*. The former can be shown by collapsing the $(n-m_p+\sigma_p)$-cube into a σ_p-cube identified by the σ_p dimensions in the set $\mathcal{J}$, and the bandwidth of each link increased by a factor of 2^{n-m_p}. ∎

4 Algorithms

4.1 Overview

In this section we present algorithms for both *real* and *mixed* generalized shuffle permutations (GSH's). We consider *full-cube permutations* (FCP) and *extended-cube permutations* (ECP). Algorithms of the ECP will utilize algorithms for the FCP as primitives.

Lemma 2 defines an algorithm for GSH's that we refer to as Algorithm A0. Every dimension, except the first and last, is traversed twice in successive exchanges. By combining the successive communications through a look-ahead scheme Algorithm A1 is obtained. The number of start-ups is reduced to σ instead of $2(\sigma-1)$ for a real GSH. The modified algorithm can be improved further by dividing the data set for each communication that takes place into two packets that are sent during consecutive cycles, Algorithm A1′. The data transfer time is reduced approximately by a factor of two compared to Algorithm A1.

Lemma 3 also defines a dimension permutation algorithm, that is of interest for mixed GSH's, since the dimension exchanges can be performed as nearest-neighbor communications. Lemma 4 shows how a real GSH can be performed as a sequence of nearest-neighbor communications by the inclusion of a virtual dimension. If the virtual dimension $v \in \Gamma_s$, then Algorithm A2 [1],

[14] is obtained. If instead $v \in \mathcal{Q}_s - \Gamma_s$, then each processor contains K *valid* elements and K *dummy* elements, and σ steps suffice by Corollary 2, Algorithm A2'.

For *n-port communication* the algorithms based on Lemmas 2, 3, and 4 are generalized by dividing the data set K into σ parts. Each such part is subject to a dimension exchange sequence with a different starting dimension. The starting dimension can be chosen arbitrarily. Note that at most $\lceil \log_2 \sigma \rceil$ virtual dimensions suffice to realize the *n-port* versions of the *one-port* algorithms.

We first present algorithms for real shuffle permutations, then algorithms for mixed shuffle permutations. The complexity estimates for the different algorithms are summarized in Table 3. For notational convenience the algorithms are described for real shuffle operations of order $\sigma_p = n$.

4.2 Full-cube, real shuffle algorithms

4.2.1 One-port communication

Algorithm A0. This algorithm is a direct application of Lemma 2. The algorithm can be expressed as:

```
/* bit(i, x) = the ith bit of x. */
/* pid = the processor id. */
/* nbr[i] = the neighbor processor id along dimension i. */
do i = n − 1, 1, −1
    if (bit(i, pid) = bit(i − 1, pid)) then
        /* The intermediate node is passive. */
        recv (nbr[i], tmp)
        send (nbr[i − 1], tmp)
    else
        /* The node is active, exchange needed. */
        send (nbr[i], buf)
        recv (nbr[i − 1], buf)
    endif
enddo
```

The path from a node $a = (a_{n-1}a_{n-2}\ldots a_0)$ to the destination node $(a_{n-2}a_{n-3}\ldots a_0 a_{n-1})$ is:

$$(\underline{a_{n-1}a_{n-2}}\ldots a_1 a_0) \rightarrow (a_{n-2}\underline{a_{n-1}a_{n-3}}\ldots a_1 a_0) \rightarrow \cdots \rightarrow (a_{n-2}a_{n-3}\ldots\underline{a_{n-1}a_0})$$
$$\rightarrow (a_{n-2}a_{n-3}\ldots a_0 a_{n-1}),$$

where the underlined dimensions are subject to exchange during the next step, if $a_i \neq a_{i-1}$. After the first exchange step, the shuffle permutation on an n-cube is reduced to two independent shuffle permutations on two $(n-1)$-dimensional subcubes that are performed recursively and

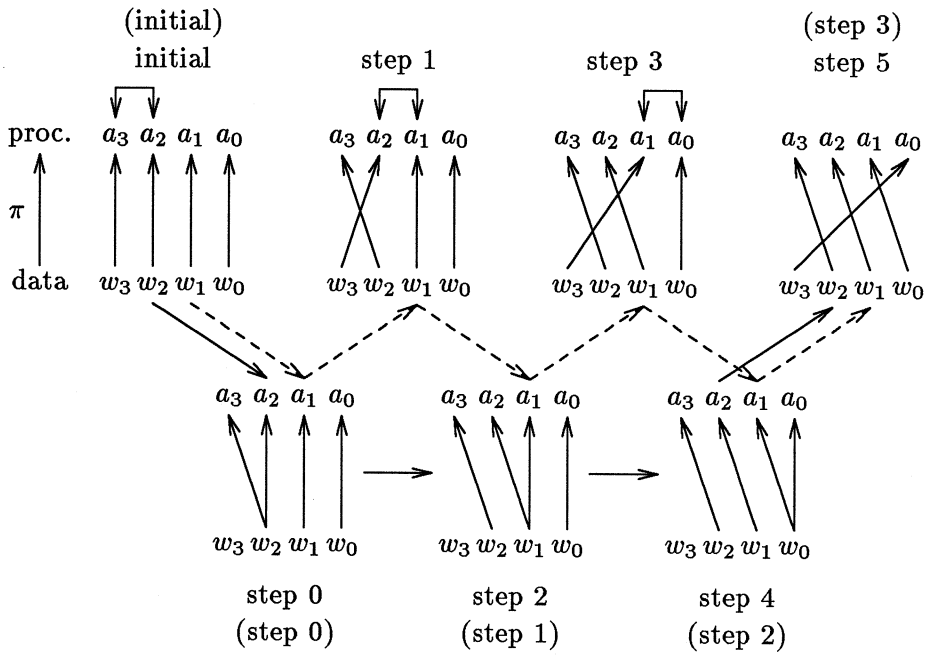

Figure 3: The data allocation as a function of exchange step for Algorithms A0 and A1.

concurrently. After step $k, 0 \leq k < n-1$ there are 2^{k+1} independent shuffle operations of order $n-k-1$. Each exchange operation takes place on 2^{n-2} independent 2-cubes.

Figure 3 illustrates the data allocations as a function of the exchange step on a 4-cube. The state for each half exchange step is shown below the sequence of states after each complete exchange step. The dashed arrows show the sequence of communications. The solid arrows apply to Algorithm A1. Figure 4 shows the interprocessor communication as a function of the exchange step for a shuffle operation on a 4-cube. The data to be shuffled is identified by the initial processor address (the sender), and is given in parentheses above the processor addresses.

Algorithms A1 (A1′). Algorithm A0 needs $2(n-1)$ routing cycles for a real shuffle of order n. All dimensions, except dimensions $n-1$ and 0 are subject to two exchange operations. For example, the path originating at processor (01101) traverses the same edge in dimension 3 twice (11101) $\rightarrow$ (10101) $\rightarrow$ (11101), Figure 5. The number of routing cycles can be reduced to n by combining successive communications along the same dimension, and removing redundant communications, like the one in the example. In Algorithm A1 routing cycles $2i-1$ and $2i$ of Algorithm A0 are combined, except for $i=0$ and $i=n-1$. The data transfer time remains the same. In Algorithm A0 all edges in a dimension are used in one direction ($0 \rightarrow 1$ or $1 \rightarrow 0$) in every step. By eliminating redundant communications half of the edges in a dimension become idle. By combining non-redundant communications the other half of the edges carry twice the load ($2K$).

By including the temporary storage in the intermediate nodes required by Algorithm A0 in the description of the dimension exchange sequence, Algorithms A0, A1, and A2 can be

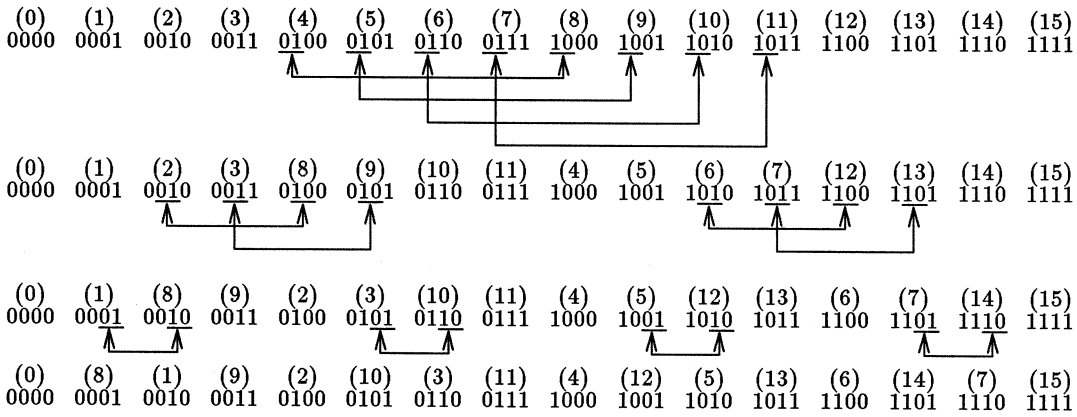

Figure 4: The processor communication as a function of exchange step for a shuffle on a 4-cube.

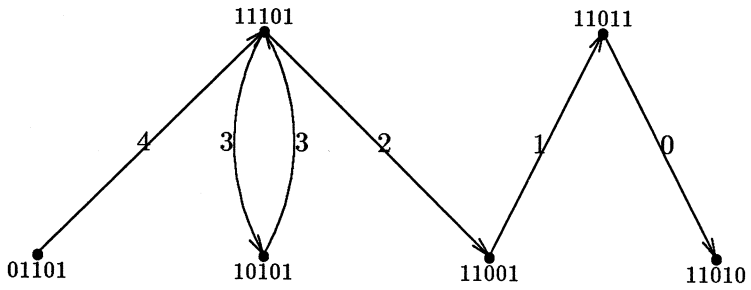

Figure 5: The path of node (01101) in Algorithm A1.

$(r_{j+1}r_jr_{j-1})$	$E(j+1,j)$	$E(j,j-1)$	combined j
000	$(\text{recv}_{j+1}, \text{send}_j)$	$(\text{recv}_j, \text{send}_{j-1})$	ϕ
001	$(\text{recv}_{j+1}, \text{send}_j)$	$(\text{send}_j, \text{recv}_{j-1})$	send_j
010	$(\text{send}_{j+1}, \text{recv}_j)$	$(\text{send}_j, \text{recv}_{j-1})$	ϕ
011	$(\text{send}_{j+1}, \text{recv}_j)$	$(\text{recv}_j, \text{send}_{j-1})$	recv_j
100	$(\text{send}_{j+1}, \text{recv}_j)$	$(\text{recv}_j, \text{send}_{j-1})$	recv_j
101	$(\text{send}_{j+1}, \text{recv}_j)$	$(\text{send}_j, \text{recv}_{j-1})$	ϕ
110	$(\text{recv}_{j+1}, \text{send}_j)$	$(\text{send}_j, \text{recv}_{j-1})$	send_j
111	$(\text{recv}_{j+1}, \text{send}_j)$	$(\text{recv}_j, \text{send}_{j-1})$	ϕ

Table 1: The combined communication along dimension i for Algorithm A1.

treated uniformly. The temporary storage is modeled by a virtual dimension $v \in \mathcal{Q}_s - \Gamma_s$. The initial and final data sets are located in memory locations for which $a_v = 0$. Temporary storage corresponds to $a_v = 1$. The first *two* dimension exchanges in Algorithm A0 consist of the following *four* communication cycles:

$$(0v_{k-1}v_{k-2}\ldots v_0|\underline{r_{n-1}}r_{n-2}\ldots r_0) \rightarrow (1v_{k-1}v_{k-2}\ldots v_0|\bar{r}_{n-1}r_{n-2}\ldots r_0)|r_{n-1} \neq r_{n-2},$$

$$(1v_{k-1}v_{k-2}\ldots v_0|r_{n-1}\underline{r_{n-2}}\ldots r_0) \rightarrow (0v_{k-1}v_{k-2}\ldots v_0|r_{n-1}\bar{r}_{n-2}\ldots r_0)|r_{n-1} = r_{n-2},$$

$$(0v_{k-1}v_{k-2}\ldots v_0|r_{n-1}\underline{r_{n-2}}r_{n-3}\ldots r_0) \rightarrow (1v_{k-1}v_{k-2}\ldots v_0|r_{n-1}\bar{r}_{n-2}r_{n-3}\ldots r_0)|r_{n-2} \neq r_{n-3},$$

$$(1v_{k-1}v_{k-2}\ldots v_0|r_{n-1}r_{n-2}\underline{r_{n-3}}\ldots r_0) \rightarrow (0v_{k-1}v_{k-2}\ldots v_0|r_{n-1}r_{n-2}\bar{r}_{n-3}\ldots r_0)|r_{n-2} = r_{n-3}.$$

The first cycle in each pair of cycles implies a sending of all data from one half of the nodes to the other half. The nodes defined by $r_{n-1-i} \neq r_{n-2-i}$ are *empty* after communication cycle $2i$, where $i = \{0, 1\ldots, n-2\}$ is the dimension exchange step. Cycle $2i+1$ communicates in dimension $n-2-i$, the same dimension as cycle $2(i+1)$. In cycle $2i+1$ nodes for which $r_{n-1-i} = r_{n-2-i}$ send the content of the second half of their storage to the first half of the storage of nodes $(r_{n-1}r_{n-2}\ldots r_{n-1-i}\bar{r}_{n-2-i}r_{n-3-i}\ldots r_0)$. In cycle $2(i+1)$ the nodes for which $r_{n-2-i} \neq r_{n-3-i}$ send the content of the first half of their storage across dimension $n-2-i$. Hence, half of the nodes that send data in dimension $n-2-i$ during cycle $2i+1$ also send data in the same dimension during cycle $2(i+1)$. The other half receives during cycle $2(i+1)$ the data set it sent during cycle $2i+1$. Since both communications are in the same dimension, $n-2-i$, they are redundant. Half of the processors (i.e., those in the empty state) have no data at any given intermediate step. The processors in the other half have $2K$ elements each. Figure 6 shows the sequence of communication steps on a 4-cube. The numbers in parentheses are the data identified by the processor addresses initially.

Table 1 shows the 8 possible patterns of $(r_{j+1}r_jr_{j-1})$ and its corresponding combined communication along cube dimension j. The second and third columns are the two communication steps for $E(j+1,j)$ and $E(j,j-1)$, respectively, in which send_j (recv_j) denotes "send (receive) along dimension j". The last column shows the combined communication along dimension j with ϕ denoting the cancellation of communication.

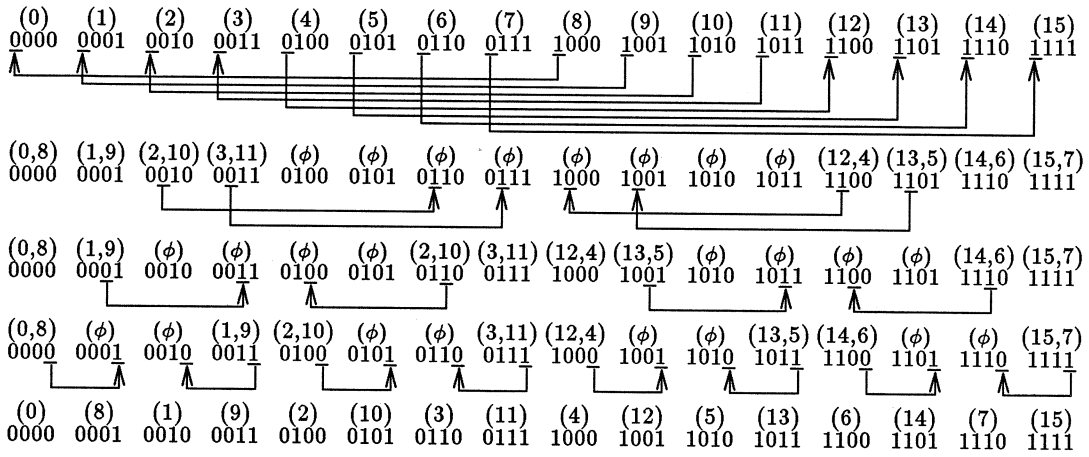

Figure 6: The sequence of communication steps for Algorithm A1.

The algorithm can be expressed as follows:

```
/* bit(i, x), pid and nbr[i] are defined as before. */
/* buf[1] = the local data to be shuffled. */
/* buf[2] = the temporary buffer. */
if (n ≤ 1) stop
if (bit(n − 1, pid) ≠ bit(n − 2, pid)) then
    send (nbr[n − 1], buf[1])
else
    recv (nbr[n − 1], buf[2])
endif
do j = n − 2, 1, −1
    if (bit(j + 1, pid) = bit(j, pid)) then
        /* was in a holding state. */
        if (bit(j, pid) ≠ bit(j − 1, pid)) then
            /* need to change to an empty state. */
            send (nbr[j], buf[1 : 2])
        endif
    else
        if (bit(j, pid) = bit(j − 1, pid) then
            recv (nbr[j], buf[1 : 2])
        endif
    endif
enddo
if (bit(1, pid) = bit(0, pid)) then
    if (bit(0, pid) = bit(n − 1, pid)) then
        send (nbr[0], buf[2])
    else
        send (nbr[0], buf[1])
    endif
```

else
recv $(nbr[0], buf[1])$
endif

The complexity of Algorithm A1 is always less than, or at most equal to that of Algorithm A0. Note that Algorithm A1 can be improved further by approximately a factor of two for the data transfer time, if one send *and* one receive operation can be performed concurrently on different ports. In Algorithm A1, if a node issues a send (receive) during routing cycle i, then it does not issue a send (receive) during cycles $i-1$ and $i+1$ (if these cycles exist). Therefore, the communication during cycle i can be split into two communications, each communicating half of the data set that needs to be communicated. The data volume for each intermediate step is reduced to K, instead of $2K$, for each active link. This algorithm is labeled A1′. Note that Algorithms A1 (A1′) only uses a communication link in one direction $0 \rightarrow 1$ *or* $1 \rightarrow 0$.

Algorithm A2 (A2′). In Algorithms A0 and A1 the required storage is twice the size of the data set, if the temporary storage is accounted for. By employing Lemma 4 and using dimension v_{k-1} as the "fixed" dimension, the storage need is reduced to the size of the data set. For a real shuffle permutation dimension v_{k-1} is not included in the index set $\mathcal{J}$ and $n+1$ exchange steps are required for a shuffle of real order n. By using a dimension used for local storage addresses as the virtual dimension, the communication in each cycle becomes *bidirectional*, i.e., exchange operations. During each step, all communications occur in the same dimension of the cube. Processors in subcube 0 exchange the second half of the data with the first half of the data of the processors in subcube 1. The sequence of exchange steps for a shuffle permutation can be illustrated as follows:

$$(\underline{v_{k-1}}v_{k-2}\ldots v_1v_0|r_{n-1}r_{n-2}\ldots r_1\underline{r_0}) \rightarrow (\underline{r_0}v_{k-2}\ldots v_1v_0|r_{n-1}r_{n-2}\ldots r_2\underline{r_1}v_{k-1})$$

$$\rightarrow (\underline{r_1}v_{k-2}\ldots v_1v_0|r_{n-1}r_{n-2}\ldots \underline{r_2}r_0v_{k-1}) \rightarrow \cdots \rightarrow (\underline{r_{n-2}}v_{k-2}\ldots v_1v_0|\underline{r_{n-1}}r_{n-3}\ldots r_1r_0v_{k-1})$$

$$\rightarrow (\underline{r_{n-1}}v_{k-2}\ldots v_1v_0|r_{n-2}r_{n-3}\ldots r_1r_0\underline{v_{k-1}}) \rightarrow (v_{k-1}v_{k-2}\ldots v_1v_0|r_{n-2}r_{n-3}\ldots r_1r_0r_{n-1}).$$

Figure 7 shows the 4 exchange steps in a 3-cube that realizes the shuffle permutation. Figure 8 shows the data allocation as a function of the exchange step in a 4-cube.

Note that after n exchange steps half of the data (for each processor) have been permuted to the right processor. The other half of the data need one more exchange step. Hence, if the data that need the final exchange step were *dummy* data, then n exchange steps would suffice. If instead of choosing $v \in \Gamma_s$, v is selected such that $v \in \mathcal{Q}_s - \Gamma_s$, then n steps suffice. Consider subcubes 00, 01, 10 and 11 with respect to dimensions $n-1$ and 0. During the first step, data in subcubes 01 and 10 are sent to subcubes 00 and 11, respectively. During the next $n-2$ steps, data are exchanged within subcubes 00 and 11 while subcubes 01 and 10 are idle. During the last step, half of the data (for each processor) in subcubes 00 and 11 are sent to subcubes 10 and 01, respectively. The amount of data communicated during every step is K, instead of $\frac{K}{2}$.

Note that all data are sent through some shortest path in A2′, since each dimension is only routed once (unlike A2). The total bandwidth required is the same as the lower bound. With

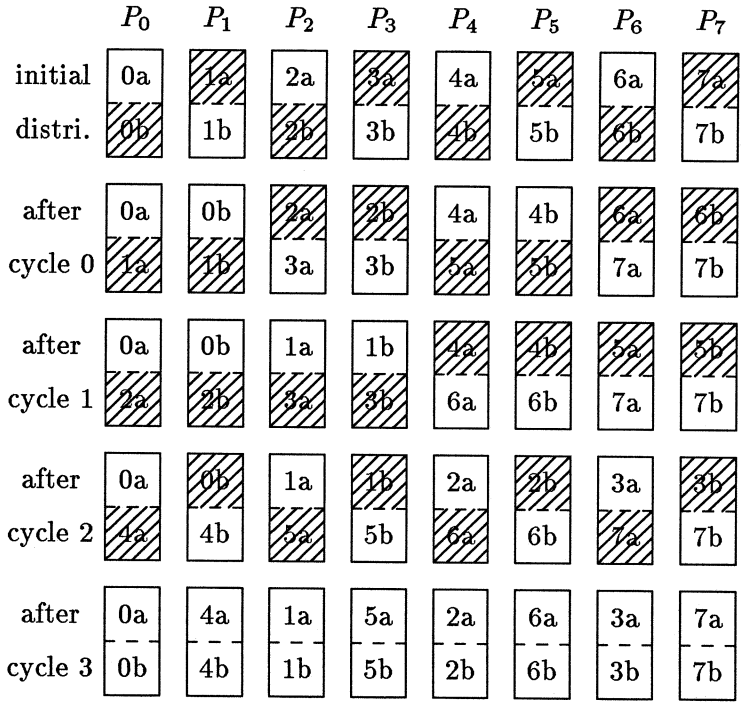

Figure 7: The 4 exchange steps in a 3-cube. The shaded areas are the parts of data subject to exchange during the next step.

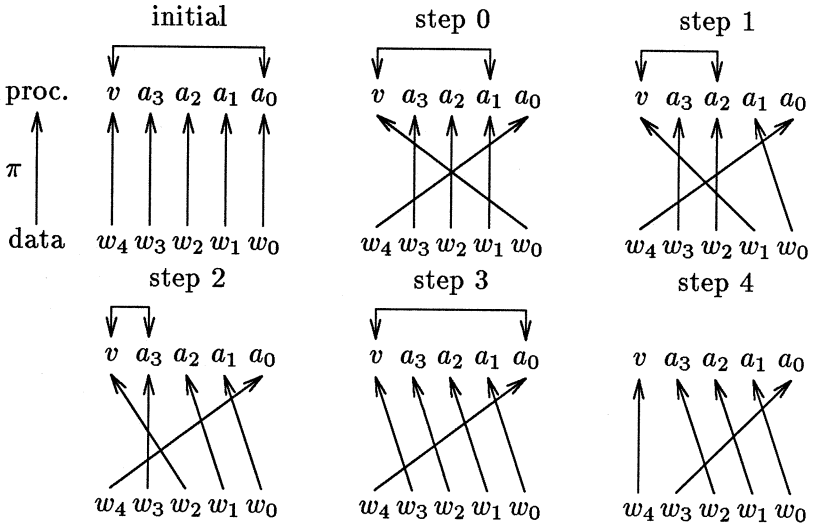

Figure 8: The changes of data allocations along time.

one-port communication, the data transfer time nKt_c is exactly twice the lower bound. Note that Algorithm A1 (A1′) traverses the cube dimensions in decreasing order while Algorithm A2 (A2′) traverses the cube dimensions in increasing order, cyclically. In Algorithm A1 the active half of processors are changing, while in Algorithm A2′ the active half stay the same.

4.2.2 n-port communication

The *n-port* versions of Algorithms A0, A1 and A1′ are obtained by using the same exchange sequences, cyclically, with different starting indices. Since their complexities are always higher than, or at most, the same as, that of Algorithm A2′, we omit them in the following discussion.

Algorithm A2. For a shuffle permutation of order n the data is partitioned into n equal sized subsets. Assume $\log_2 n = \eta$ is an integer, then exchange sequences i, $0 \leq i < n$, can be represented as

$$
\begin{aligned}
&(v_{k-1} \ldots v_{k-\eta} \underline{v_{k-\eta-1}} v_{k-\eta-2} \ldots v_0 | r_{n-1} r_{n-2} \ldots r_{i+2} r_{i+1} \underline{r_i} r_{i-1} \ldots r_0) \\
&\rightarrow (v_{k-1} \ldots v_{k-\eta} \underline{r_i} v_{k-\eta-2} \ldots v_0 | r_{n-1} r_{n-2} \ldots r_{i+2} \underline{r_{i+1}} v_{k-\eta-1} r_{i-1} \ldots r_0) \\
&\rightarrow (v_{k-1} \ldots v_{k-\eta} \underline{r_{i+1}} v_{k-\eta-2} \ldots v_0 | r_{n-1} r_{n-2} \ldots \underline{r_{i+2}} r_i v_{k-\eta-1} r_{i-1} \ldots r_0) \rightarrow \cdots \\
&\rightarrow (v_{k-1} \ldots v_{k-\eta} \underline{r_{i-2}} v_{k-\eta-2} \ldots v_0 | r_{n-2} r_{n-3} \ldots r_i v_{k-\eta-1} \underline{r_{i-1}} r_{i-3} \ldots r_0 r_{n-1}) \\
&\rightarrow (v_{k-1} \ldots v_{k-\eta} \underline{r_{i-1}} v_{k-\eta-2} \ldots v_0 | r_{n-2} r_{n-3} \ldots r_i \underline{v_{k-\eta-1}} r_{i-2} \ldots r_0 r_{n-1}) \\
&\rightarrow (v_{k-1} \ldots v_{k-\eta} v_{k-\eta-1} v_{k-\eta-2} \ldots v_0 | r_{n-2} r_{n-3} \ldots r_i r_{i-1} r_{i-2} \ldots r_0 r_{n-1}),
\end{aligned}
$$

where $(v_{k-1} v_{k-2} \ldots v_{k-\eta}) = i$. Note that the assumption of $\log_2 n$ being an integer is only required for notational convenience.

Formally, let $\hat{\mathcal{J}} = \alpha_0, \alpha_1, \ldots, \alpha_{\sigma-1}$ be a sequence obtained from the order set $\mathcal{J}$. Let L be the left rotation operator, i.e., $L(\hat{\mathcal{J}}) = \alpha_1, \ldots, \alpha_{\sigma-1}, \alpha_0$, and $L^i = L^{i-1} \circ L$. The σ exchange sequences are defined by Seq_i, $0 \leq i < \sigma$.

$$\text{Seq}_i = L^i(\hat{\mathcal{J}}), \alpha_i.$$

Note that α_i is also the first dimension of Seq_i. For $\sigma = 3$,

$$\text{Seq}_0 = \alpha_0, \alpha_1, \alpha_2, \alpha_0.$$

$$\text{Seq}_1 = \alpha_1, \alpha_2, \alpha_0, \alpha_1.$$

$$\text{Seq}_2 = \alpha_2, \alpha_0, \alpha_1, \alpha_2.$$

During any routing cycle, different sequences use edges in different dimensions.

Algorithm A2′. The *one-port* version of Algorithm A2′ uses a single dimension during each exchange step. An *n-port* version of the algorithm can be created by defining the exchange sequences Seq_i, $0 \leq i < n$,

$$\text{Seq}_i = L^i(\hat{\mathcal{J}}).$$

Node a is active (during the intermediate steps) for Seq_i if $a_i = a_{(i-1) \bmod n}$. During routing cycle j, $1 \leq j \leq n-2$, node a exchanges data along dimension i if $a_{(i-j) \bmod n} = a_{(i-j-1) \bmod n}$.

Another *n-port* version of Algorithm A2′ is to pipeline the communications along one path per node, such as a path derived from Seq_0. In the *one-port* algorithm, data originated from different nodes use different edges in dimension i during routing cycle i (if they need to be routed through dimension i). The paths from all N nodes are edge-disjoint. The complexity of the pipelined algorithm with optimal packet size, $(\sqrt{Kt_c} + \sqrt{(n-1)\tau})^2$, is higher than the version of Algorithm A2′ using n sequences, in general.

4.3 Full-cube, real, generalized shuffle algorithms

For a *real* GSH of order n on an n-cube, one can modify any algorithm for a *real* shuffle permutation on an n-cube by considering cube dimensions $\alpha_0, \alpha_1, \ldots, \alpha_{n-1}$, instead of cube dimensions $0, 1, \ldots, n-1$. Since all cube dimensions are assumed to have the same communication characteristics the complexity is unaffected by the change. For a *real* GSH of order $\sigma_p < n$, the permutation consists of $2^{n-\sigma_p}$ permutations in independent subcubes. These permutations are performed concurrently. By Lemma 6 no advantage can be taken of the fact that $\sigma_p < n$, if the permutation algorithm is optimal. We ignore the improvement possible over our algorithms since they are of optimal order.

4.4 Full-cube, mixed, generalized shuffle algorithms

By Lemma 3 a *mixed* GSH can be realized through the following exchange sequence:

$$gsh(a) = E(\alpha_i, \alpha_{(i-1) \bmod \sigma}) \circ \cdots \circ E(\alpha_i, \alpha_{(i+2) \bmod \sigma}) \circ E(\alpha_i, \alpha_{(i+1) \bmod \sigma}).$$

No communication is necessary for exchange operations for which both dimensions are in Γ_s. The *one-port communication* complexity is $\frac{\sigma_p K}{2} t_c + \sigma_p \tau$, which is optimal with respect to the data transfer time, and the start-up time. Figure 9 shows a GSH on the index set $\mathcal{J} = \{3, 0, 2, 4, 1\}$. This index set is derived from the initial index map $\pi^b(0) = 3, \pi^b(1) = 4, \pi^b(2) = 0, \pi^b(3) = 1$ and $\pi^b(4) = 2$, and $\pi^a(i) = i, 0 \leq i \leq 4$. The cycle on the left is derived from the dimension permutation function δ. The cycle on the right is derived from δ', which describes the mapping for the set of logic dimensions. Table 2 shows the sequence of exchange operations of the permutation for the case where $0, 1, 2 \in \mathcal{Q}_p$ (real dimensions) and $3, 4 \in \mathcal{Q}_s$ (virtual dimensions). The permutation is the conversion from consecutive storage to cyclic storage [5], [8].

For *n-port communication* it is possible to extend the index set $\mathcal{J}$ with virtual dimensions and apply Algorithm A2 as for a real GSH. If there are at least $\lceil \log_2 \sigma_p \rceil$ virtual dimensions in $\Gamma_s - \mathcal{J}$ then no extra storage is needed. Alternatively, one can run m'_s GSH's concurrently, where $m'_s = |\{\alpha_i | \alpha_i \in \mathcal{J} \cap \Gamma_s, \alpha_{(i+1) \bmod \sigma} \in \mathcal{J} \cap \Gamma_p\}|$. No two generalized shuffle permutations need to utilize the same processor dimension during the same routing cycle. The complexity is $\frac{\sigma_p K}{2m'_s} t_c + \sigma_p \tau$. The data transfer time is bounded from above by $\frac{\sigma_p K}{2} t_c$ and from below by $\frac{K}{2} t_c$. The latter algorithm, though it may have a higher data transfer time, has one less start-up. The choice of algorithm depends on K, τ, t_c and m'_s.

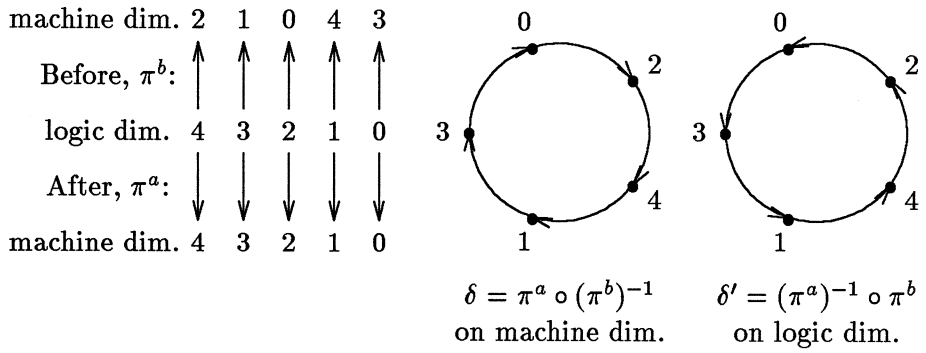

Figure 9: The cycles formed by traversing δ function (on the left) and δ' function (on the right).

Encoding	P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7
$(\underline{43}\|2\underline{10})$	0	4	8	12	16	20	24	28
	1	5	9	13	17	21	25	29
$(w_1w_0w_4w_3w_2)$	2	6	10	14	18	22	26	30
	3	7	11	15	19	23	27	31
$(\underline{43}\|\underline{2}10)$	0	1	8	9	16	17	24	25
	4	5	12	13	20	21	28	29
$(w_1w_2w_4w_3w_0)$	2	3	10	11	18	19	26	27
	6	7	14	15	22	23	30	31
$(\underline{43}\|210)$	0	1	8	9	4	5	12	13
	16	17	24	25	20	21	28	29
$(w_1w_4w_2w_3w_0)$	2	3	10	11	6	7	14	15
	18	19	26	27	22	23	30	31
$(\underline{43}\|2\underline{10})$	0	1	8	9	4	5	12	13
	2	3	10	11	6	7	14	15
$(w_4w_1w_2w_3w_0)$	16	17	24	25	20	21	28	29
	18	19	26	27	22	23	30	31
$(43\|210)$	0	1	2	3	4	5	6	7
	8	9	10	11	12	13	14	15
$(w_4w_3w_2w_1w_0)$	16	17	18	19	20	21	22	23
	24	25	26	27	28	29	30	31

Table 2: Conversion from $(w_1w_0w_4w_3w_2)$ encoding to $(w_4w_3w_2w_1w_0)$ encoding. The underlined dimensions are the pair of dimensions going to be exchanged during the next routing cycle. Machine dimensions 0, 1 and 2 are real dimensions, and 3, 4 are virtual dimensions.

4.5 Extended-cube permutation algorithms

For an extended-cube permutation, we adopt a three phase scheme: subcube expansion, full-cube permutation, and subcube compression. In the first phase, each processor partitions its data into 2^{n-m_p} pieces and all processors concurrently perform a *one-to-all personalized communication* to each of the 2^{m_p} distinct subcubes of dimension $n - m_p$. In the second phase, an algorithm for a full-cube permutation is used concurrently in the 2^{n-m_p} subcubes, with the data volume reduced by a factor of 2^{n-m_p}. The third phase is the reverse of the first phase, i.e., data are gathered (compressed) into the original active subcube. The complexities of the first phase and the third phase are the same, and for the best known algorithm [2,9] the complexity of each is

$$K\left(1 - \frac{1}{2^{n-m_p}}\right) t_c + (n - m_p)\tau$$

for *one-port communication*, and

$$\frac{K}{n - m_p}\left(1 - \frac{1}{2^{n-m_p}}\right) t_c + (n - m_p)\tau$$

for *n-port communication*. With *n-port communication*, if the algorithm used in the second phase is optimal, then the total data transferred is $\approx \frac{K}{2^{n-m_p+1}} + \frac{2K}{n-m_p}$ compared to $\frac{K}{2}$ for an optimal algorithm using links of the active subcube only. The speed-up of the data transfer time is about a factor of $\frac{n-m_p}{4}$, but the start-ups compare as $2(n - m_p) + \sigma_p$ to σ_p. For *one-port communication*, the data transferred is $\approx \frac{\sigma_p K}{2^{n-m_p+1}} + 2K$ compared to $\frac{\sigma_p K}{2}$. The speed-up of the data transfer time is about a factor of $\frac{\sigma_p}{4}$.

4.6 Algorithm comparison and summary

All presented algorithms have a communication complexity of the same order as the lower bound. The difference in the communication complexities of the algorithms and the lower bound is generally a small constant factor. The control is distributed for all algorithms. The communication complexities of a real GSH of order σ_p are summarized in Table 3. The second last column contains the ratio of data transfer times and the lower bound. The last column contains the ratio of the start-up times and the lower bound.[2]

For *one-port communication* the number of start-ups of the algorithms is at most twice the lower bound. Algorithms A1, A1′ and A2′ have a number of start-ups equal to the lower bound. For *n-port communication*, Algorithm A2′ has a number of start-ups equal to the lower bound. The data transfer time for Algorithm A2 with *one-port communication* is at most 50% higher than the lower bound, and that of Algorithm A1′ and A2′ is twice the lower bound. For *n-port communication* all algorithms have a data transfer time that is at most a factor of two, higher than the lower bound.

We conclude that the data transfer time of Algorithm A2 is a factor of $\frac{\sigma_p+1}{2\sigma_p}$ lower than that of Algorithm A2′ for both *one-port* and *n-port communications*. The factor ranges from

[2]For convenience, we use σ_p as the lower bound for comparison. (For a real GSH of odd σ_p, the lower bound is $\sigma_p - 1$.)

Comm.	Alg.	B_{opt}	Communication complexity	t_c factor/lb	τ factor/lb
one-port comm.	A0	K	$2(\sigma_p-1)Kt_c+2(\sigma_p-1)\lceil\frac{K}{B}\rceil\tau$	[2, 4)	2
	A1	$2K$	$2(\sigma_p-1)Kt_c+((\sigma_p-2)\lceil\frac{2K}{B}\rceil+2\lceil\frac{K}{B}\rceil)\tau$	[2, 4)	1
	A1′	K	$\sigma_pKt_c+\sigma_p\lceil\frac{K}{B}\rceil\tau$	2	1
	A2	$\frac{K}{2}$	$\frac{(\sigma_p+1)}{2}Kt_c+(\sigma_p+1)\lceil\frac{K}{2B}\rceil\tau$	(1, 1.5]	(1, 1.5]
	A2′	K	$\sigma_pKt_c+\sigma_p\lceil\frac{K}{B}\rceil\tau$	2	1
n-port comm.	A2	$\frac{K}{2\sigma_p}$	$\frac{(\sigma_p+1)}{2\sigma_p}Kt_c+(\sigma_p+1)\lceil\frac{K}{2\sigma_pB}\rceil\tau$	(1, 1.5]	(1, 1.5]
	A2′	$\frac{K}{\sigma_p}$	$Kt_c+\sigma_p\lceil\frac{K}{\sigma_pB}\rceil\tau$	2	1

Table 3: Summary of the communication complexities of various algorithms for a real GSH of order σ_p.

Comm.	*one-port*					*n-port*	
Algorithm	A0	A1	A1′	A2	A2′	A2	A2′
Memory	$2K$	$2K$	$2K$	K	$2K$	K	$2K$
$\min\{K\}$	1	1	1	2	1	$2\sigma_p$	σ_p

Table 4: The memory required and the minimum size of data volume required for maximum concurrency for different algorithms with *one-port communication*.

0.5 - 0.75. With optimal packet size the number of start-ups of Algorithm A2 is a factor of $1+\frac{1}{\sigma_p}$ higher than that of Algorithm A2′ for both *one-port* and *n-port communications*. The factor ranges from 1 to 1.5. For a small packet size relative to the data set, the number of start-ups compares as the data transfer times. Algorithm A2′ is relatively more competitive for *n-port communication* than for *one-port communication* with optimal packet size. The break-even point between Algorithms A2 and A2′ is $\tau=(\sigma_p-1)\frac{K}{2}t_c$ for *one-port communication*, and $\tau=(1-\frac{1}{\sigma_p})\frac{K}{2}t_c$ for *n-port communication*.

Table 4 shows the memory required for different algorithms, and the minimum size of the data set K for which the complexity estimates are true. All algorithms require a memory which is at most twice the original data volume.

In addition to the algorithms above it is also possible to perform a generalized shuffle permutation through *all-to-all personalized communication* [9,13], but it can be shown always to be inferior to the algorithms presented here. Likewise, performing the permutation by recursively applying an optimal matrix transposition algorithm [8] yields a complexity higher than that of the algorithms presented here.

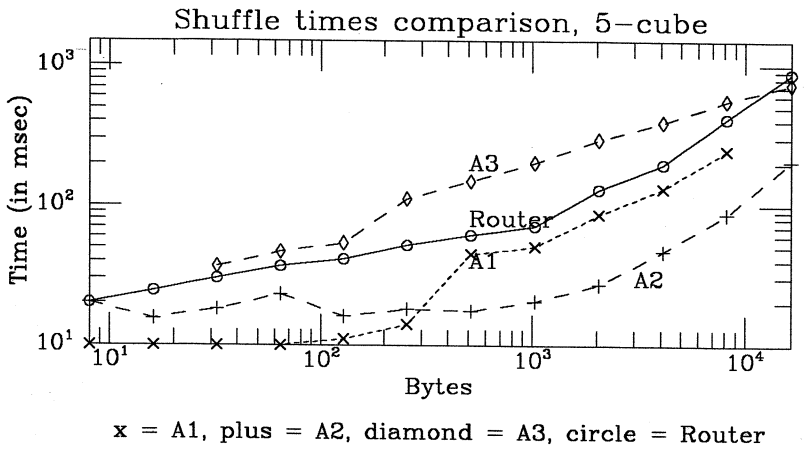

Figure 10: The measured shuffle times as a function of message lengths on an iPSC 5-cube.

5 Implementation issues

We have implemented the *one-port* versions of Algorithms A1 and A2 on the Intel iPSC/1. For comparison we have also performed shuffle permutations by using the routing logic. For a real, full-cube, shuffle permutation with a message size less than a few hundred bytes, Algorithm A1 is of lowest complexity, while for a large message size Algorithm A2 is preferable, Figure 10. We have also included measurements of shuffle permutations by *all-to-all personalized communication*, Algorithm A3, which as expected requires more time than Algorithms A1 and A2. Algorithm A3 has the disadvantage that the data to be exchanged between pairs of neighbors may not be in contiguous memory locations. Therefore, it may require more start-ups, or local data movement determined by the total data volume to be transmitted. Local data movement requires a substantial amount of time on the Intel iPSC [8]. The best time of either Algorithm A1 or A2 is 5 - 10 times less than that of the router. All *one-port* algorithms have a complexity that is linear in the number of dimensions for shuffle permutations with a real order equal to the number of cube dimensions. The deviation from the linear dependence exhibited in Figures 11 and 12 is due to a hybrid implementation which optimizes the sum of start-up time and the time for local data movement [8].

6 Summary and conclusions

We have proved lower bounds and devised a few algorithms optimal within small constant factors for stable generalized shuffle permutations on Boolean cubes. With concurrent communication on all ports of each node our algorithms are optimal within a factor of two. No lower bounds, or optimal algorithms were previously known for this case, except for shuffle permutations through *all-to-all personalized communication* [13], which requires a data volume of an order $O(2^{\sigma_P})$ to be optimal. With communication restricted to one processor port at a time, Algorithms A1′ and A2′ have a time complexity comparable to that of Algorithm A2 [14]. Depending on the

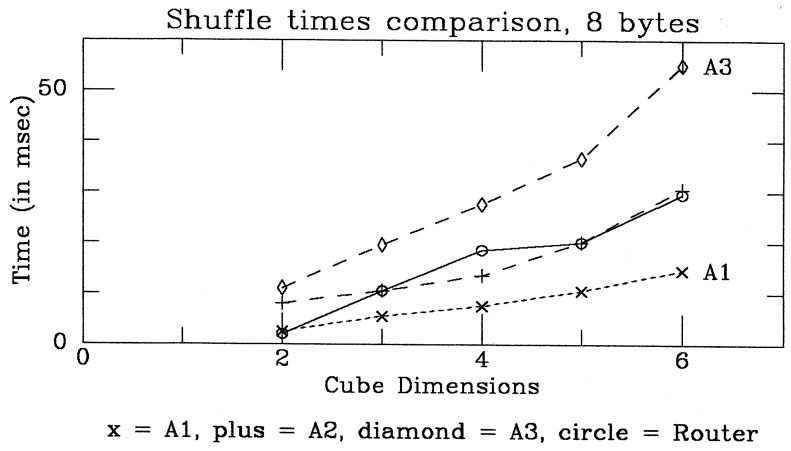

Figure 11: The measured shuffle times as a function of cube dimensions on an iPSC 6-cube with $K = 8$ bytes.

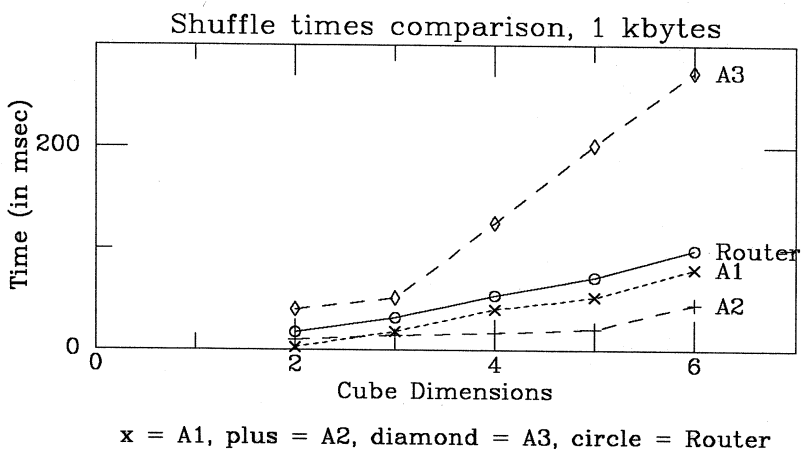

Figure 12: The measured shuffle times as a function of cube dimensions on an iPSC 6-cube with $K = 1$ *k*bytes.

machine parameters and data volume one or the other may have the lowest complexity with the ratio of communication times $\frac{\text{Algorithm } A2'}{\text{Algorithm A2}}$ varying in the range $[\frac{2}{3}, 2]$. Implementations on the Intel iPSC/1 show that for shuffle permutations with a message size of up to a few hundred bytes the measured communication time of Algorithm A1 is the lowest of Algorithms A1, A2, *all-to-all personalized communication*, and the router. For a larger message size Algorithm A2 has the lowest time complexity.

Acknowledgement

We would like to thank Eileen Connolly for her editorial effort. The generous support by the Office of Naval Research under Contract No. N00014-86-K-0310 is gratefully acknowledged.

References

[1] Peter M. Flanders. A unified approach to a class of data movements on an array processor. *IEEE Trans. Computers*, 31(9):809–819, September 1982.

[2] Ching-Tien Ho and S. Lennart Johnsson. Distributed routing algorithms for broadcasting and personalized communication in hypercubes. In *1986 International Conf. on Parallel Processing*, pages 640–648, IEEE Computer Society, 1986. Tech. report YALEU/DCS/RR-483, May 1986.

[3] Ching-Tien Ho and S. Lennart Johnsson. *Stable Dimension Permutations on Boolean Cubes.* Technical Report YALEU/DCS/RR-617, Department of Computer Science, Yale University, October 1988.

[4] Ching-Tien Ho and S. Lennart Johnsson. *Unstable Dimension Permutations on Boolean Cubes.* Technical Report YALEU/DCS/RR-, Department of Computer Science, Yale University, 1988. in preparation.

[5] S. Lennart Johnsson. Communication efficient basic linear algebra computations on hypercube architectures. *J. Parallel Distributed Comput.*, 4(2):133–172, April 1987. (Tech. Rep. YALEU/DCS/RR-361, Yale Univ., New Haven, CT, January 1985).

[6] S. Lennart Johnsson. The FFT and fast Poisson solvers on parallel architectures. In *Fast Fourier Transforms for Vector and Parallel Computers*, The Mathematical Sciences Institute, Cornell Univ., 1987. YALEU/DCS/RR-582, March 1987.

[7] S. Lennart Johnsson and Ching-Tien Ho. *Algorithms for Multiplying Matrices of Arbitrary Shapes Using Shared Memory Primitives on a Boolean Cube.* Technical Report YALEU/DCS/RR-569, Dept. of Computer Science, Yale Univ., New Haven, CT, October 1987. Revision of YALE/DCS/RR-530. Presented at the ARMY Workshop on Medium Scale Parallel Processors, Stanford Univ., January 1986.

[8] S. Lennart Johnsson and Ching-Tien Ho. Matrix transposition on Boolean n-cube configured ensemble architectures. *SIAM J. Matrix Anal. Appl.*, 9(3):419–454, July 1988.

YALE/DCS/RR-572, September 1987. (Revised edition of YALEU/DCS/RR-494 November 1986.).

[9] S. Lennart Johnsson and Ching-Tien Ho. *Spanning graphs for optimum broadcasting and personalized communication in hypercubes.* Technical Report YALEU/DCS/RR-500, Dept. of Computer Science, Yale Univ., New Haven, CT, November 1986. Revised November 1987, YALEU/DCS/RR-610. To appear in IEEE Trans. Computers.

[10] S. Lennart Johnsson, Ching-Tien Ho, Michel Jacquemin, and Alan Ruttenberg. Computing fast Fourier transforms on Boolean cubes and related networks. In *Advanced Algorithms and Architectures for Signal Processing II*, pages 223–231, Society of Photo-Optical Instrumentation Engineers, 1987. Report YALEU/DCS/RR-598, October 1987.

[11] Yousef Saad and Martin H. Schultz. *Topological properties of hypercubes.* Technical Report YALEU/DCS/RR-389, Dept. of Computer Science, Yale Univ., New Haven, CT, June 1985.

[12] Quentin F. Stout and Bruce Wager. *Intensive hypercube communication I: prearranged communication in link-bound machines.* Technical Report CRL-TR-9-87, Computing Research Lab., Univ. of Michigan, Ann Arbor, MI, 1987.

[13] Quentin F. Stout and Bruce Wager. Passing messages in link-bound hypercubes. In Michael T. Heath, editor, *Hypercube Multiprocessors 1987*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1987.

[14] Paul N. Swarztrauber. Multiprocessor FFTs. *Parallel Computing*, 5:197–210, 1987.