# Generalized Shuffle Permutations on Boolean Cubes

# Generalized Shuffle Permutations on Boolean Cubes

S. Lennart Johnsson
Ching-Tien Ho

TR-04-91

February 1991


Parallel Computing Research Group

Center for Research in Computing Technology
Harvard University
Cambridge, Massachusetts

Air Force Office of Scientific Research grant: 44-752-9000-2.
National Science Foundation grant: 44-752-9705-2.

# Generalized Shuffle Permutations on Boolean Cubes

S. Lennart Johnsson
Division of Applied Sciences
Harvard University
Cambridge, MA 02138
and
Thinking Machines Corp.
Johnsson@harvard.edu

Ching-Tien Ho
IBM Almaden Research Center
650 Harry Road
San Jose, CA 95120-6099
Ho@ibm.com

**Abstract.** In a *generalized shuffle permutation* an address $(a_{q-1}a_{q-2}\ldots a_0)$ receives its content from an address obtained through a cyclic shift on a subset of the $q$ dimensions used for the encoding of the addresses. Bit-complementation may be combined with the shift. We give an algorithm that requires $\frac{K}{2}+2$ exchanges for $K$ elements per processor, when storage dimensions are part of the permutation, and concurrent communication on all ports of every processor is possible. The number of element exchanges in sequence is independent of the number of processor dimensions $\sigma_r$ in the permutation. With no storage dimensions in the permutation our best algorithm requires $(\sigma_r+1)\lceil\frac{K}{2\sigma_r}\rceil$ element exchanges. We also give an algorithm for $\sigma_r=2$, or the real shuffle consists of a number of cycles of length two, that requires $\frac{K}{2}+1$ element exchanges in sequence when there is no bit complement. The lower bound is $\frac{K}{2}$ for both real and mixed shuffles with no bit complementation. The minimum number of communication start-ups is $\sigma_r$ for both cases, which is also the lower bound. The data transfer time for communication restricted to one port per processor is $\sigma_r\frac{K}{2}$, and the minimum number of start-ups is $\sigma_r$. The analysis is verified by experimental results on the Intel iPSC/1, and for one case also on the Connection Machine.

## 1 Introduction

The main contributions of this paper are optimal algorithms for dimension permutations on Boolean cube configured distributed memory multi-processors, and lower bounds for such permutations with concurrent communication on all channels. Communication systems (packet or circuit switched) only allowing communication on one channel at a time per processor are treated briefly. The Connection Machine is an example of a computer allowing concurrent communication on all channels. Some computers allow concurrent communication on several, but not all channels of every processor. The techniques for concurrent communication on all channels may be adapted to such architectures.

A dimension permutation is defined by permuting and/or complementing the bits of the logic address field. There are $M(\log_2 M)!$ possible dimension permutations where $M$ is the number of elements in the address field. We consider *stable* permutations [2], i.e., permutations for which the data occupy the same machine address space before and after the data rearrangement. The machine address space consists of two parts: a processor address field and a local storage

address field. *Real* dimension permutations are restricted to the processor address field, whereas *mixed* dimension permutations include parts of both the local storage and processor address fields. *Virtual* dimension permutations that only include local storage addresses require no communication, and are not considered here.

The reason for distinguishing between processor and local storage dimensions is that access times to local storage usually is considerably faster than communication between processors. The width of the data paths to local memory are often equal to the word width of the architecture (32 or 64 bits), while the width of network channels typically are in the range 1 – 16 bits. Furthermore, the contention for network channels is often a more serious issue than the contention for local memory, and the techniques for reducing network contention are quite different from the techniques for handling contention for local memory.

The difference in the width of the data paths to local memory and inter-processor communication channels, and the different protocols used for interprocessor communication and local memory references often allow many local memory references to be performed in the time required for an inter-processor communication. For instance, in a Connection Machine model CM-2 each processor has a single 32-bit wide data path to memory, while inter-processor communication channels are 1-bit wide. The width of the communication channels is determined by a trade off between many demands for off-chip communication. Pins on chips is a highly critical resource in many technologies. A memory read of 32 bits is a one-cycle operation, while a store is slightly longer. An exchange of 32-bits between a pair of processors requires about 185 cycles. On a fully configured Connection Machine model CM-2 22 exchanges can be performed concurrently. In the $n$-port communication model local memory references on a Connection Machine account for at most about 25% of the total time.

Examples of dimension permutations are $k$-shuffle/unshuffle permutations, matrix transposition, bit-reversal, vector-reversal, and conversion between various data allocation schemes, such as consecutive and cyclic storage [3, 4], reshaping of arrays [7], and multi-sectioning. A vector reversal expressed in binary code is the operation $(b_{n-1}b_{n-2}\ldots b_0) \rightarrow (\overline{b_{n-1}}\ \overline{b_{n-2}}\ldots\overline{b_0})$ where $\overline{b_i}$ is the complement of $b_i$, i.e., $V(i) \leftarrow V(2^n - 1 - i)$ with elements numbered consecutively from zero. Bit-reversal is the operation $(b_{n-1}b_{n-2}\ldots b_0) \rightarrow (b_0b_1\ldots b_{n-1})$, and shuffle is the operation $(b_{n-1}b_{n-2}\ldots b_0) \rightarrow (b_{n-2}b_{n-3}\ldots b_0b_{n-1})$. The transposition of a matrix with axis lengths being powers of two is the operation $(r_{p-1}r_{p-2}\ldots r_0c_{q-1}c_{q-2}\ldots c_0) \rightarrow (c_{q-1}c_{q-2}\ldots c_0r_{p-1}r_{p-2}\ldots r_0)$, which is a shuffle repeated $p$ times, or an unshuffle performed $q$ times. In the consecutive allocation scheme successive elements are allocated to the same processor, whereas in cyclic allocation successive elements are assigned to successive processors in a wraparound fashion. From the illustration below, it is clear that conversion between the two is a dimension permutation, which can be defined as an $n$-shuffle, or $n$-unshuffle.

$$\underset{Consecutive}{(\underbrace{b_{p-1}b_{p-2}\ldots b_{p-n}}_{paddr}|\underbrace{b_{p-n-1}b_{p-n-2}\ldots b_0}_{maddr})} \qquad \underset{Cyclic}{(\underbrace{b_{p-1}b_{p-2}\ldots b_n}_{maddr}|\underbrace{b_{n-1}b_{n-2}\ldots b_0}_{paddr})}$$

Nassimi and Sahni [10, 11] consider stable, *real* dimension permutations on multi-processors configured as meshes and hypercubes, while Flanders [1] focuses on stable, both *real* and *mixed*, dimension permutations on two-dimensional mesh-connected multi-processors. Swarztrauber [15] considers dimension permutations on Boolean cubes. Swarztrauber does not consider bit

complementation, which is not required for the FFT, the main focus of [15]. In all of the previous work only one communication channel per processor is used in any step of the algorithms, or for lower bounds. We consider concurrent communication on all channels of all processors. Such communication is possible on the Connection Machine. The emphasis is on mixed, stable permutations. Real, stable permutations are treated briefly. Algorithms for real, stable dimension permutations with one-port communication can be found in [10, 11, 15].

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 results from implementations on the Intel iPSC/1 and the Connection Machine are described in Section 5. We conclude with a few remarks in Section 6.

## 2 Preliminaries

In *one-port communication* the communication is restricted to *one* exchange operation per processor. In *n-port communication* exchange operations can be performed concurrently on *all* ports of every processor. With the processors configured as a Boolean $n$-cube each processor has $n$ channels (edges) to $n$ distinct processors (nodes). With a binary encoding there is an adjacent processor for every bit of the address, or *dimension*. The number of node-disjoint minimum length paths between any pair of nodes at real distance $d$ is $d$. There are $n-d$ node-disjoint paths of length $d+2$ between any pair of nodes at real distance $d$ [12]. The *machine address space* is $\mathcal{A} = \{(\overbrace{a_{q-1}a_{q-2}\ldots a_{q-n}}^{paddr} | \overbrace{a_{q-n-1}a_{q-n-2}\ldots a_0}^{maddr})\}$, where $a_i \in \{0,1\}$. The symbol "|" denotes concatenation. The rightmost dimension is dimension zero, the least significant one. We arbitrarily let the *processor address field* form the high order part of the global address, and the *local storage address field* form the low order part. The *real distance* between two locations in the address space is $Hamming_r(a,a') = \sum_{i=q-n}^{q-1}(a_i \oplus a'_i)$ and the *virtual distance* is $Hamming_v(a,a') = \sum_{i=0}^{q-n-1}(a_i \oplus a'_i)$. $Hamming(a,a') = Hamming_r(a,a') + Hamming_v(a,a')$. The set of machine dimensions is $\mathcal{D}^{\mathcal{A}} = \{q-1, q-2, \ldots, 0\}$. The processor dimensions are $\mathcal{D}_p^{\mathcal{A}} = \{q-1, q-2, \ldots, q-n\}$, and the local storage dimensions are $\mathcal{D}_s^{\mathcal{A}} = \{q-n-1, q-n-2, \ldots, 0\}$. The *logic address space* $\mathcal{L} = \{(b_{m-1}b_{m-2}\ldots b_0)\}$ has the dimensions $\mathcal{D}^{\mathcal{L}} = \{m-1, m-2, \ldots, 0\}$, $m \leq q$. A *dimension allocation* is a mapping: $\mathcal{D}^{\mathcal{L}} \rightarrow \mathcal{D}^{\mathcal{A}}$. The set of machine dimensions used for the data allocation is $\mathcal{D}^{\mathcal{U}} \subseteq \mathcal{D}^{\mathcal{A}}$. $\mathcal{D}_p^{\mathcal{U}} = \mathcal{D}^{\mathcal{U}} \cap \mathcal{D}_p^{\mathcal{A}}$ and $\mathcal{D}_s^{\mathcal{U}} = \mathcal{D}^{\mathcal{U}} \cap \mathcal{D}_s^{\mathcal{A}}$. $K = 2^k$ is the number of elements per processor, where $k$ is the cardinality of the set $\mathcal{D}_s^{\mathcal{U}}$, or $k = |\mathcal{D}_s^{\mathcal{U}}|$.

**Definition 1** A *Stable Dimension Permutation* (SDP), $\delta$, is a one-to-one mapping $\mathcal{D}^{\mathcal{U}} \rightarrow \mathcal{D}^{\mathcal{U}}$ such that $\beta_i \leftarrow \left\{ \begin{array}{c} \alpha_j \\ \overline{\alpha_j} \end{array} \right.$, where $\alpha_i, \beta_i \in \mathcal{D}^{\mathcal{U}}$, $i \in \{q-1, q-2, \ldots, 0\}$, and $\alpha_i \neq \alpha_j$, $\beta_i \neq \beta_j$, for any $i \neq j$. The *index set* $\mathcal{J}$ of the permutation is the set $\{\alpha_i | \alpha_i \neq \beta_i\}$. The SDP is *real* if $\mathcal{J} \subseteq \mathcal{D}_p^{\mathcal{U}}$, *virtual* if $\mathcal{J} \subseteq \mathcal{D}_s^{\mathcal{U}}$, and *mixed* otherwise. The *order* $\sigma$ of the permutation is $|\mathcal{J}|$, the *real order* $\sigma_r$ is $|\mathcal{J} \cap \mathcal{D}_p^{\mathcal{U}}|$, and the *virtual order* $\sigma_s$ is $|\mathcal{J} \cap \mathcal{D}_s^{\mathcal{U}}|$.

**Definition 2** Let $\mathcal{J} = \{\alpha_{\sigma-1}, \alpha_{\sigma-2}, \ldots, \alpha_0\} \subseteq \mathcal{D}^{\mathcal{A}}$, where $\sigma > 1$ and $\alpha_i \neq \alpha_j$, $i \neq j$. A *stable generalized shuffle permutation* (GSH) is an SDP such that $\alpha_i \leftarrow \left\{ \begin{array}{c} \alpha_{(i-1) \bmod \sigma} \\ \overline{\alpha_{(i-1) \bmod \sigma}} \end{array} \right.$, $i \in$

| paddr | | | |
|---|---|---|---|
| 00 | 01 | 10 | 11 |
| (00\|00) | (01\|00) | (10\|00) | (11\|00) |
| (00\|01) | (01\|01) | (10\|01) | (11\|01) |
| (00\|10) | (01\|10) | (10\|10) | (11\|10) |
| (00\|11) | (01\|11) | (10\|11) | (11\|11) |

$\Longrightarrow$

| paddr | | | |
|---|---|---|---|
| 00 | 01 | 10 | 11 |
| (00\|00) | (01\|00) | (00\|10) | (01\|10) |
| (00\|01) | (01\|01) | (00\|11) | (01\|11) |
| (10\|00) | (11\|00) | (10\|10) | (11\|10) |
| (10\|01) | (11\|01) | (10\|11) | (11\|11) |

Figure 1: Shuffle permutation of order two.

$\{0, 1, \ldots, \sigma - 1\}$.

In a dimension permutation the content of address $a$ is assigned to address $\delta(a)$ (i.e., $\delta(a) \leftarrow a$). A GSH corresponds to a single cycle on the bits of the address field. We let the GSH correspond to a left cyclic shift on the address field. For example, the shuffle $(a_4a_3a_2a_1a_0) \leftarrow (a_3a_2a_1a_0a_4)$ has the index set $\{4, 3, 2, 1, 0\}$. For a 2-shuffle on the same dimensions $\mathcal{J} = \{4, 2, 0, 3, 1\}$. In general, an SDP defines several cycles on the set $\mathcal{J}$. There are $2^{m-\sigma}$ identical SDP of order $\sigma$ in an address space of size $2^m$.

A *real* GSH preserves the local address map. Data is moved between processors in subcubes defined by the set of dimensions $\mathcal{J}$. A *real* GSH is a generalization of the *collinear planar exchanges* described in [1]. A *virtual* GSH (called *vertical exchanges* in [1]) implies the same local data movement in all processors. A *mixed* GSH (called *planar-vertical exchanges* in [1]) implies interprocessor communication and a different placement in local memory of communicated data. For instance, the GSH of order two defined by $(a_3a_2|a_1a_0) \leftarrow (a_1a_2|a_3a_0)$ results in the data motion in Figure 1. An *i–cycle* as defined in [15] does not include local memory reordering, and is further restricted to include only the most significant storage dimension.

**Definition 3** A *sub-cube permutation* (SCP) is an algorithm for dimension permutation in which the routing is confined to the set of processors to which the data set is allocated. An *extended-cube permutation* (ECP) is an algorithm for dimension permutation in which the routing is extended to an $n_e$-cube in which the $n$-cube holding data is embedded.

# 3 Time complexity

The transmission time for each element is $t_c$, and the start-up time, or overhead, for each packet of $B$ elements is $\tau$. In a circuit-switched system $\tau$ corresponds to the time to set one switch in a path. In the time complexity $T(ports, \sigma_r, n, K)$ for a lower bound, or an algorithm, the first argument 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 used for data allocation, and the last argument the size of the local data set.

**Lemma 1** *The time complexity of an SDP of real order $\sigma_r < n$ cannot be improved by communication in the $n - \sigma_r$ processor dimensions not included in the index set, if a dimension*

*permutation is required within all $\sigma_r$-cubes, and the SDP algorithm uses full bandwidth within each $\sigma_r$-cube.*

**Proof:** We prove the theorem by contradiction. Let the lower bound for the SDP of real order $\sigma_r < n$ on an $n$-cube be $T = \frac{W}{L}$, where $W$ is the total bandwidth required and $L$ the available bandwidth per unit time. Now, map $2^{n-\sigma_r}$ nodes of the $n$-cube to a single node of a $\sigma_r$-cube by identifying all nodes with the same values of the address bits for dimensions in the set $\mathcal{J}$ of the SDP. Furthermore, increase the communications bandwidth of each channel in the $\sigma_r$-cube by a factor of $2^{n-\sigma_r}$. Then, every algorithm for the SDP performed on the $n$-cube can be converted to an algorithm on the $\sigma_r$-cube with a running time $T''$ that is at most the same. Hence, $T \geq T'' = \frac{2^{n-\sigma_r}W}{2^{n-\sigma_r}L} = T$.

**Lemma 2** *Lower bounds for a sub-cube GSH of real order $\sigma_r > 0$ are*

$$T_{gsh}^{lb}(1, \sigma_r, n, K) = \begin{cases} \max(\sigma_r \frac{K}{2} t_c, (\sigma_r - 1)\tau), & \textit{if (number of dimensions that are moved} \\ & \textit{and not complemented) is odd} \\ \max(\sigma_r \frac{K}{2} t_c, \sigma_r \tau), & \textit{otherwise} \end{cases}$$

$$T_{gsh}^{lb}(\sigma_r, \sigma_r, n, K) = \begin{cases} \max(\frac{K}{2} t_c, (\sigma_r - 1)\tau), & \textit{if (number of dimensions that are moved} \\ & \textit{and not complemented) is odd} \\ \max(\frac{K}{2} t_c, \sigma_r \tau), & \textit{otherwise,} \end{cases}$$

**Proof:** The minimum number of start-ups, or switch settings in a path, is equal to the maximum number of channels that must be traversed, which is $\max_{a \in \mathcal{A}} Hamming_r(a, \delta(a))$.

The minimum data transfer time is bounded from below by the required bandwidth divided by the available bandwidth. By Lemma 1 it suffices to consider a $\sigma_r$-cube. For each dimension $j \in \mathcal{J} \cap \mathcal{D}_p^{\mathcal{U}}$ such that $j \leftarrow i$, where $i \in \mathcal{J} \cap \mathcal{D}_p^{\mathcal{U}}$, only nodes for which $(a_i \neq a_j)$ need to send elements across dimensions $j$. If instead $i \in \mathcal{J} \cap \mathcal{D}_s^{\mathcal{U}}$ then all nodes send half of their data. Therefore, the bandwidth requirement for each $\sigma_r$-cube is $\sigma_r 2^{\sigma_r} \frac{K}{2}$. The available bandwidth per routing cycle is $2^{\sigma_r}$ for *one-port* and $\sigma_r 2^{\sigma_r}$ for *n-port communication*. ∎

The lower bounds are not tight when some dimensions in the permutation are complemented. For instance, consider the case $(\alpha_0 \alpha_1) \leftarrow (\overline{\alpha_1} \alpha_0)$. In this case the permutation is a cyclic shift in a loop of length four. All non-minimum length paths are of length three. There is only one minimum length path between any pair of processors, and only one non-minimum length path as well. For a routing time of $\frac{K}{2}$ element transfers in sequence at most $\frac{K}{2}$ elements can be routed along minimum length paths. The total routing requirement for elements routed along non-minimum length paths is at least $4 \times 3 \times \frac{K}{2}$. With four available links a total routing time of at most $\frac{K}{2}$ is impossible.

**Corollary 1** *The data transfer time for a sub-cube, real GSH, performed by any fixed packet size algorithm requiring c routing cycles per packet is at least $\frac{cK}{2(c-1)} t_c$ for n-port communication,*

*and at least* $\frac{c\sigma_r K}{2(c-1)}t_c$ *for one-port communication in which all processors use the same dimension during the same routing cycle.*

**Proof:** For the lower bound data transfer time in Lemma 2, all channels are used evenly in every routing cycle, and all elements are routed through a shortest path. But, during the first and last routing cycles, at least half of the channels are not used for a real GSH. For *one-port communication*, the same argument applies to the first and last routing cycles for any cube dimension used by all processors. ∎

A dimension permutation on a data set allocated to an $n$-dimensional subcube of an $n_e$-cube can be realized by subcube expansion from the $n$-cube to the $n_e$-cube, full cube permutation, and compression to the original $n$-cube. The permutation is performed with local data sets reduced by a factor of $2^{n_e-n}$. The subcube expansion (compression) is of type *one-to-all* (*all-to-one*) *personalized communication* [6].

**Corollary 2** *Lower bounds for a GSH of order* $\sigma$ *extended from an* $n$*-cube to an* $n_e$*-cube are*

$$T_{gsh}^{lb}(1,\sigma_r,n,K) = \begin{cases} \max\left((2(K-\frac{K}{2^{n_e-n}})+\sigma_r\frac{K}{2^{n_e-n+1}})t_c, (2(n_e-n)+\sigma_r-1)\tau\right), \\ \quad \textit{if (number of dimensions that are moved and not complemented) is odd,} \\ \max\left((2(K-\frac{K}{2^{n_e-n}})+\sigma_r\frac{K}{2^{n_e-n+1}})t_c, (2(n_e-n)+\sigma_r)\tau\right), \\ \quad \textit{otherwise,} \end{cases}$$

$$T_{gsh}^{lb}(n_e,\sigma_r,n,K) = \begin{cases} \max\left((2\frac{K}{n_e-n}+\frac{K}{2^{n_e-n+1}})t_c, (2(n_e-n)+\sigma_r-1)\tau\right), \\ \quad \textit{if (number of dimensions that are moved and not complemented) is odd,} \\ \max\left((2\frac{K}{n_e-n}+\frac{K}{2^{n_e-n+1}})t_c, (2(n_e-n)+\sigma_r)\tau\right), \\ \quad \textit{otherwise.} \end{cases}$$

The proof follows from Lemma 2 and the lower bounds for one-to-all personalized communication [6].

# 4 Algorithms

A left cyclic shift of the set $\mathcal{J} = \{\alpha_{\sigma-1}, \alpha_{\sigma-2}, \ldots, \alpha_0\}$ can be achieved in $\sigma - 1$ exchanges by: 1) exchanging adjacent pairs of dimensions starting with any pair and progressing to the right cyclicly, terminating with the pair immediately to the left of the starting pair, 2) exchanging an arbitrary fixed dimension with successively higher dimensions modulo $\sigma$. The left cyclic shift can also be accomplished in $\sigma + 1$ exchanges by augmenting the index set with one dimension $v$, and using it as a fixed exchange dimension: $v \leftrightarrow \alpha_i, v \leftrightarrow \alpha_{(i+1) \bmod \sigma}, \ldots, v \leftrightarrow \alpha_{(i+\sigma-1) \bmod \sigma}, v \leftrightarrow \alpha_{(i+\sigma) \bmod \sigma}$. The first sequence uses all dimensions twice except for the first and last dimension in the sequence. The second sequence uses every dimension once, except the fixed exchange dimension. The last sequence uses all dimensions once, except the starting dimension which is used twice, and the fixed exchange dimension. These three exchange sequences are the basis for our Boolean cube algorithms. A shuffle permutation with bit-complementation can be accomplished by complementing the appropriate bits in the exchange operation.

The essential ideas used to achieve concurrency in communication is: 1) pipelining, 2) starting the cyclic shifts in several dimensions, 3) factoring of cycles into several cycles. Independent concurrent exchange sequences are created by partitioning the local data set, and defining one sequence per partition. By properly choosing the sequences a uniform load on the communication system is achieved. Dimension permutations can also be performed as recursive matrix transpositions [4, 13, 14], or by performing *all-to-all personalized communication* twice [6, 14].

The main focus below is on algorithms for mixed, stable dimension permutations. We first consider the case with a single mixed GSH, then consider shuffle permutations that can be factored into several mixed GSH. We conclude by considering two algorithms for real GSH.

## 4.1 Mixed, generalized shuffle algorithms

### 4.1.1 A single GSH

A mixed GSH with an index set $\mathcal{J}$ consisting of a single block of processor dimensions followed by one virtual dimension can be performed in $\sigma_r$ exchanges by using $\alpha_0$ as the fixed exchange dimension, where the GSH is defined by $(\alpha_{\sigma_r}\alpha_{\sigma_r-1}\ldots\alpha_1|\alpha_0)$. If the index set $\mathcal{J}$ consists of one block of processor dimensions and several storage dimensions, then the GSH is factored into two cycles: one on the block of processor dimensions and the memory dimension immediately to the right of it, one on all memory dimensions. For instance, a cyclic shift on the set $(\alpha_{\sigma-1}\alpha_{\sigma-2}\ldots\alpha_{\sigma-\sigma_r}|\alpha_{\sigma-\sigma_r-1}\alpha_{\sigma-\sigma_r-2}\ldots\alpha_0)$ is factored into a cyclic shift on $(\alpha_{\sigma-1}\alpha_{\sigma-2}\ldots\alpha_{\sigma-\sigma_r}|\alpha_{\sigma-\sigma_r-1})$ followed by a cyclic shift on $(\alpha_{\sigma-\sigma_r-1}\alpha_{\sigma-\sigma_r-2}\ldots\alpha_0)$. The second GSH is *virtual*. For *n-port communication* we will first describe a pipelined algorithm in which all data items start their exchange sequence in the first processor dimension, then an algorithm for which some of the exchange sequences are initiated in other processor dimensions than the first.

**Pipelining**

The mixed GSH with a single local storage dimension and $K \geq 2$ elements per processor requires that $\frac{K}{2}$ elements be exchanged in each dimension. For *n-port communication* the data transfers can be pipelined. The number of element transfers in sequence is $\frac{K}{2} + \sigma_r - 1$. Figure 2 shows the scheduling of local memory locations paired with respect to the exchange dimension. The table entries indicate the processor dimension in which one element in a local pair is subject to exchange. Figure 3 shows an example of memory locations exchanging data in the case of eight processors. Two memory locations marked by the same symbol, for instance X0, are exchanged with each other. For more than two local memory locations the exchange pattern is repeated for all local pairs. The exchange algorithm for two local memory locations is as follows

**for** $i := 1$ **to** $\sigma_r$ **do**
  **forall** $(\alpha_{\sigma_r}\alpha_{\sigma_r-1}\ldots\alpha_1|\alpha_0)$ **do**
    **if** $\alpha_0 \oplus \alpha_i = 1$ **then**
      $(\alpha_{\sigma_r}\alpha_{\sigma_r-1}\ldots\alpha_{i+1}\alpha_i\alpha_{i-1}\ldots\alpha_1|\alpha_0) \rightarrow (\alpha_{\sigma_r}\alpha_{\sigma_r-1}\ldots\alpha_{i+1}\overline{\alpha_i}\alpha_{i-1}\ldots\alpha_1|\overline{\alpha_0})$
    **endif**
  **endforall**
**endfor**

| | Time step | | | | | | |
|---|---|---|---|---|---|---|---|
| M | $d_0$ | 1 | 2 | 3 | | | |
| e | $d_1$ | | 1 | 2 | 3 | | |
| m | $d_2$ | | | 1 | 2 | 3 | |
| | $d_3$ | | | | 1 | 2 | 3 |

Figure 2: Exchange dimensions for a generalized shuffle permutation through pipelining.

| P0 | P1 | P2 | P3 | P4 | P5 | P6 | P7 |
|---|---|---|---|---|---|---|---|
| Initial allocation | | | | | | | |
| 0 | 2 | 4 | 6 | 8 | 10 | 12 | 14 |
| 1 | 3 | 5 | 7 | 9 | 11 | 13 | 15 |
| Step 1 | | | | | | | |
| | X0 | | X1 | | X2 | | X3 |
| X0 | | X1 | | X2 | | X3 | |
| Step 2 | | | | | | | |
| | | X0 | X1 | | | X2 | X3 |
| X0 | X1 | | | X2 | X3 | | |
| Step 3 | | | | | | | |
| | | | | X0 | X1 | X2 | X3 |
| X0 | X1 | X2 | X3 | | | | |
| Final allocation | | | | | | | |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |

Figure 3: Data exchanges for a mixed generalized shuffle permutation of real order three.

A local memory reordering before and after the exchange sequence allows the sending and receiving local addresses in an exchange to be the same. Indirect addressing can be avoided for all data interchanges. Half of the processors use the lower address in a local pair, half of the processors the upper address. The initial and final reordering, and the pairs of memory locations involved in the exchanges are illustrated in Figure 4. The reordering and exchange algorithm is defined by

Align

**forall** $(\alpha_{\sigma_r}\alpha_{\sigma_r-1}\ldots\alpha_1|\alpha_0)$ **do**
  **if** $\alpha_{\sigma_r} \oplus \alpha_{\sigma_r-1} \oplus \ldots \oplus \alpha_1 = 1$ **then**
    $(\alpha_{\sigma_r}\alpha_{\sigma_r-1}\ldots\alpha_1|\alpha_0) \rightarrow (\alpha_{\sigma_r}\alpha_{\sigma_r-1}\ldots\alpha_1|\overline{\alpha_0})$
  **endif**
**endforall**

Permute

**for** $i := 1$ **to** $\sigma_r$ **do**
  **forall** $(\alpha_{\sigma_r}\alpha_{\sigma_r-1}\ldots\alpha_1|\alpha_0)$ **do**
    **if** $\alpha_{\sigma_r} \oplus \alpha_{\sigma_r-1} \oplus \ldots \oplus \alpha_{i+1} \oplus \alpha_{i-1} \oplus \ldots \oplus \alpha_1 \oplus \alpha_0 = 1$ **then**

$(\alpha_{\sigma_r}\alpha_{\sigma_r-1}\ldots\alpha_{i+1}\alpha_i\alpha_{i-1}\ldots\alpha_1|\alpha_0) \rightarrow (\alpha_{\sigma_r}\alpha_{\sigma_r-1}\ldots\alpha_{i+1}\overline{\alpha_i}\alpha_{i-1}\ldots\alpha_1|\alpha_0)$

**endif**

**endforall**

**endfor**

Realign

**forall** $(\alpha_{\sigma_r}\alpha_{\sigma_r-1}\ldots\alpha_1|\alpha_0)$ **do**

**if** $\alpha_{\sigma_r} \oplus \alpha_{\sigma_r-1} \oplus \ldots \oplus \alpha_1 = 1$ **then**

$(\alpha_{\sigma_r}\alpha_{\sigma_r-1}\ldots\alpha_1|\alpha_0) \rightarrow (\alpha_{\sigma_r}\alpha_{\sigma_r-1}\ldots\alpha_1|\overline{\alpha_0})$

**endif**

**endforall**

The correctness follows from the following consideration.

Step 1:
$(\alpha_{\sigma_r}\alpha_{\sigma_r-1}\ldots\alpha_1|\alpha_0) \rightarrow (\alpha_{\sigma_r}\alpha_{\sigma_r-1}\ldots\alpha_1|\alpha_{\sigma_r} \oplus \alpha_{\sigma_r-1} \oplus \ldots \oplus \alpha_1 \oplus \alpha_0)$.
Steps $2 \leq j \leq \sigma_r + 1$:
$(\alpha_{\sigma_r}\alpha_{\sigma_r-1}\ldots\alpha_1|\alpha_0) \rightarrow (\alpha_{\sigma_r}\alpha_{\sigma_r-1}\ldots\alpha_j\{\alpha_{\sigma_r} \oplus \alpha_{\sigma_r-1} \oplus \ldots \oplus \alpha_1 \oplus \alpha_0\}\alpha_{j-2}\ldots\alpha_1|\alpha_0)$.
Step $\sigma_r + 2$:
$(\alpha_{\sigma_r}\alpha_{\sigma_r-1}\ldots\alpha_1|\alpha_0) \rightarrow (\alpha_{\sigma_r}\alpha_{\sigma_r-1}\ldots\alpha_1|\alpha_{\sigma_r} \oplus \alpha_{\sigma_r-1} \oplus \ldots \oplus \alpha_1 \oplus \alpha_0)$.

Tracing the steps yields

$$\begin{aligned}
(\alpha_{\sigma_r}\alpha_{\sigma_r-1}\ldots\alpha_1|\alpha_0) &\rightarrow (\alpha_{\sigma_r}\alpha_{\sigma_r-1}\ldots\alpha_1|\alpha_{\sigma_r} \oplus \alpha_{\sigma_r-1} \oplus \ldots \oplus \alpha_1 \oplus \alpha_0)\\
&\rightarrow (\alpha_{\sigma_r}\alpha_{\sigma_r-1}\ldots\alpha_2\alpha_0|\alpha_{\sigma_r} \oplus \alpha_{\sigma_r-1} \oplus \ldots \oplus \alpha_1 \oplus \alpha_0)\\
&\rightarrow (\alpha_{\sigma_r}\alpha_{\sigma_r-1}\ldots\alpha_3\alpha_1\alpha_0|\alpha_{\sigma_r} \oplus \alpha_{\sigma_r-1} \oplus \ldots \oplus \alpha_1 \oplus \alpha_0)\\
&\vdots \qquad\qquad\qquad\qquad \vdots\\
&\rightarrow (\alpha_{\sigma_r}\alpha_{\sigma_r-2}\ldots\alpha_2\alpha_1\alpha_0|\alpha_{\sigma_r} \oplus \alpha_{\sigma_r-1} \oplus \ldots \oplus \alpha_1 \oplus \alpha_0)\\
&\rightarrow (\alpha_{\sigma_{r-1}}\alpha_{\sigma_r-2}\ldots\alpha_2\alpha_1\alpha_0|\alpha_{\sigma_r} \oplus \alpha_{\sigma_r-1} \oplus \ldots \oplus \alpha_1 \oplus \alpha_0)\\
&\rightarrow (\alpha_{\sigma_{r-1}}\alpha_{\sigma_r-2}\ldots\alpha_2\alpha_1\alpha_0|\alpha_{\sigma_r}).
\end{aligned}$$

Note that precisely one local storage location in a pair defined by the exchange dimension is subject to an exchange in any step. Moreover, after the first exchange even and odd data are separated into two distinct subcubes identified by the first exchange dimension.

### Pipelining with arbitrary starting dimension

With multiple local elements it is possible to initiate exchange sequences in different dimensions. All such sequences can be executed concurrently with *n-port communication*. For an arbitrary real starting dimension $\sigma_r + 1$ exchanges are required. The fixed local exchange dimension, $v$, cannot be $\alpha_0$. The local memory is partitioned into sets of four elements identified by $v$ and $\alpha_0$. The pairs in a set of four elements are denoted $d_j^{\alpha_0=0}$ and $d_j^{\alpha_0=1}$, $j \geq 0$, where $d_j^{\alpha_0=0}$ and $d_j^{\alpha_0=1}$ identifies pairs. The set $j$ starts the exchange in dimension $\alpha_{\sigma_r-j}$. A set that starts the exchange with real dimension $\alpha_1$ does not need to use dimension $v$ for the exchange, and grouping into four elements is unnecessary. The grouping into sets of four elements is due

| P0 | P1 | P2 | P3 | P4 | P5 | P6 | P7 |
|---|---|---|---|---|---|---|---|
| Initial reordering | | | | | | | |
| | X0 | X1 | | X2 | | | X3 |
| | X0 | X1 | | X2 | | | X3 |
| Step 1 | | | | | | | |
| | | X1 | X1 | X2 | X2 | | |
| X0 | X0 | | | | | X3 | X3 |
| Step 2 | | | | | | | |
| | X1 | | X1 | X2 | | X2 | |
| X0 | | X0 | | | X3 | | X3 |
| Step 3 | | | | | | | |
| | X1 | X2 | | | X1 | X2 | |
| X0 | | | X3 | X0 | | | X3 |
| Final reordering | | | | | | | |
| | X0 | X1 | | X2 | | | X3 |
| | X0 | X1 | | X2 | | | X3 |

Figure 4: Alignment/realignment and data exchanges for a mixed generalized shuffle permutation of real order three.

to the exchange between dimensions $\alpha_0$ and $v$. This exchange need not be performed explicitly, but enforces a synchronization between exchange sequences. Exchanges subsequent to the one in which the exchange should have taken place can simply use $\alpha_0$ instead of $v$ as the fixed exchange dimension. The maximum number of local elements that can be handled concurrently using sets of four elements is $2(\sigma_r - 1)$ for $\sigma_r$ odd, and $2(\sigma_r - 2)$ for $\sigma_r$ even. For sufficiently many local elements we combine the concurrent exchange scheme with pipelining, Figure 5.

In Figure 5 the first eight memory locations have been scheduled to start in a dimension other than the first, while the following 10 locations start the exchange in the first dimension. A bullet in the table illustrates a synchronization cycle for an exchange with the next row. A synchronization cycle must succeed the exchange $v \leftrightarrow \alpha_{\sigma_r}$ and precede the exchange $\alpha_0 \leftrightarrow \alpha_1$,

| | Time step | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Memory | $d_0^0$ | $v,5$ | $v,6$ | • | | | 0,1 | 0,2 | 0,3 | 0,4 | 0,5 | | | | |
| | $d_0^1$ | | $v,5$ | $v,6$ | | | | 0,1 | 0,2 | 0,3 | 0,4 | 0,5 | | | |
| | $d_1^0$ | $v,3$ | $v,4$ | $v,5$ | $v,6$ | • | | | 0,1 | 0,2 | 0,3 | | | | |
| | $d_1^1$ | | $v,3$ | $v,4$ | $v,5$ | $v,6$ | | | | 0,1 | 0,2 | 0,3 | | | |
| | $d_2$ | 0,1 | 0,2 | 0,3 | 0,4 | 0,5 | 0,6 | | | | | | | | |
| | $d_3$ | | 0,1 | 0,2 | 0,3 | 0,4 | 0,5 | 0,6 | | | | | | | |
| | $d_4$ | | | 0,1 | 0,2 | 0,3 | 0,4 | 0,5 | 0,6 | | | | | | |
| | $d_5$ | | | | 0,1 | 0,2 | 0,3 | 0,4 | 0,5 | 0,6 | | | | | |
| | $d_6$ | | | | | 0,1 | 0,2 | 0,3 | 0,4 | 0,5 | 0,6 | | | | |
| | | | | | | | 0,1 | 0,2 | 0,3 | 0,4 | 0,5 | 0,6 | | | |
| | | | | | | | | 0,1 | 0,2 | 0,3 | 0,4 | 0,5 | 0,6 | | |
| | | | | | | | | | 0,1 | 0,2 | 0,3 | 0,4 | 0,5 | 0,6 | |
| | | | | | | | | | | 0,1 | 0,2 | 0,3 | 0,4 | 0,5 | 0,6 |

Figure 5: The exchange sequences for $\mathcal{J} = \{\alpha_6\alpha_5\alpha_4\alpha_3\alpha_2\alpha_1|\alpha_0\}$ and $K = 18$.

but can otherwise take place at an arbitrary time. Row $d_2$ through row $d_6$ are subject to pipelined exchanges starting in the first processor dimension. The unlabeled rows in the table are included to illustrate what the total time would have been if all exchanges had started in the first processor dimension. Then, rows $d_2$ through the end of the table would apply. Conceptually, one can view the schedule represented by rows $d_0^0$ through $d_6$, as being constructed by removing the bottom four unlabeled rows, and instead schedule these rows as defined by rows $d_0^0$ through $d_1^1$. Moving the bottom two rows above row $d_2$ saves two communications. Moving an additional two rows again saves two exchanges in the bottom part. But, the net savings is only one exchange due to the length of the exchange sequence in the top part.

**The communication complexity for a single mixed GSH**

The pipelined algorithm with all elements starting their exchange in the first processor dimension requires $\frac{K}{2}+\sigma_r-1$ element exchanges in sequence for *n-port communication*. Starting some sequences in other dimensions than the first yields $\sigma_r+3$ exchanges, if $K \leq 2(\sigma_r+1)$. For $K \geq 2(\sigma_r+1)$ the number of element exchanges in sequence is $\frac{K}{2}+2$. The minimum number of start-ups is $\sigma_r+3$ for the second type of algorithm. The block size $B$ for this number of start-ups is $\lceil \frac{K}{2(\sigma_r+1)} \rceil$.

**Theorem 1** *A mixed GSH of real order $\sigma_r = \sigma - 1$ and $K$ elements per processor requires at most*

$$T = \begin{cases} \frac{K}{2} + \sigma_r - 1 & \sigma_r \leq 3, K \leq 8, \\ \sigma_r + 3 & 8 \leq K \leq 2(\sigma_r+1), \\ \frac{K}{2} + 2, & K \geq 2(\sigma_r+1), \sigma_r \geq 3. \end{cases}$$

*element exchanges in sequence. The minimum number of start-ups is at least $\sigma_r$, and at most $\sigma_r+3$.*

For $K \leq 8$, or $\sigma_r \leq 3$ all elements should have their first exchange in the first processor dimension of the GSH. If $K > 8$, but less than $2(\sigma_r+1)$, then some elements should start their exchange sequence in a dimension other than the first processor dimension in the GSH. The time complexity is determined by elements that start their exchange in a dimension other than $\alpha_1$. The partitioning of the $K/\sigma_r$ space with respect to scheduling algorithms is illustrated in Figure 6. The communication complexities are summarized in Table 1. The expressions in the table include the effects of finite sizes of the communication packets (buffers) $B$, and a start-up time $\tau$ for each such packet. The overhead and the transmission times are assumed additive.

**Remarks:**

- The same local memory dimension can be used for different concurrent exchange sequences, but memory locations must be distinct. Four memory locations are needed per sequence, if the starting dimension is different from the first in the cycle.
- The alignment is made in the fixed exchange dimension, and is controlled by the parity of all processor dimensions in the shuffle, except the fixed exchange dimension. Hence, the alignment is made on the local memory dimension in the shuffle, if the exchange sequence

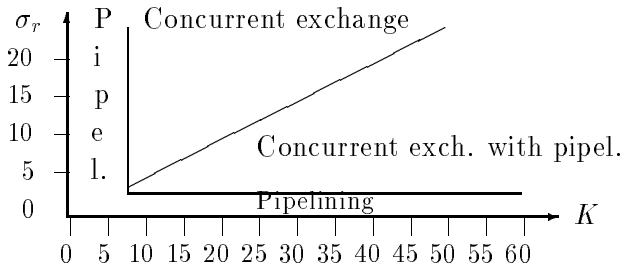


Figure 6: The partitioning of the $K/\sigma_r$ space.

| Comm. model | Time |
|---|---|
| *one-port* comm. | $\sigma_r \frac{K}{2} t_c + \sigma_r \lceil \frac{K}{2B} \rceil \tau$ |
| *n-port* comm. | $\begin{cases} (\frac{K}{2} + (\sigma_r - 1)B)t_c + (\lceil \frac{K}{2B} \rceil + \sigma_r - 1)\tau & \text{pipelining} \\ (\sigma_r + 3)(Bt_c + \tau) & \text{concurrent exchange} \\ (\frac{K}{2} + 2B)t_c + (\lceil \frac{K}{2B} \rceil + 2)\tau & \text{conc. exch. with pipel.} \end{cases}$ |

Table 1: The communication complexity for a single mixed GSH.

starts in the first real dimension in the shuffle. Otherwise, the alignment is made on the extra memory dimension used for the exchanges.

- The alignment and exchanges are controlled entirely by the dimensions in the shuffle.
- The local storage is partitioned into two blocks with respect to exchange schedules: one for exchange sequences starting in a dimension other than the first real dimension, and consisting of $2(\sigma_r - 1)$ locations for $\sigma_r$ odd, or consisting of $2(\sigma_r - 2)$ locations for $\sigma_r$ even, and one block for exchange sequences starting in the first real dimension and consisting of the remainder of the local storage. The first block is aligned on dimension $v$, and the second on dimension $\alpha_0$.

#### 4.1.2 Multiple GSH

In general, the index set $\mathcal{J}$ for a GSH can be factored into a number of mixed GSH, each for a block of unique processor dimensions and one unique storage dimension, and one virtual GSH, as illustrated in Figure 7. We refer to the mixed GSH's in the factored GSH as constituting GSH. The number of such GSH is $\beta$. All constituting, mixed GSH can be performed concurrently. For each location in local storage the exchange dimensions for different constituting GSH can be interleaved in any order. Only the order within each GSH is fixed. The techniques for scheduling of exchanges described for a single mixed GSH can be applied for each constituting GSH. The scheduling algorithms presented below are optimal within two element exchanges.

The set of exchanges defined by one dimension exchange for all constituting GSH consists

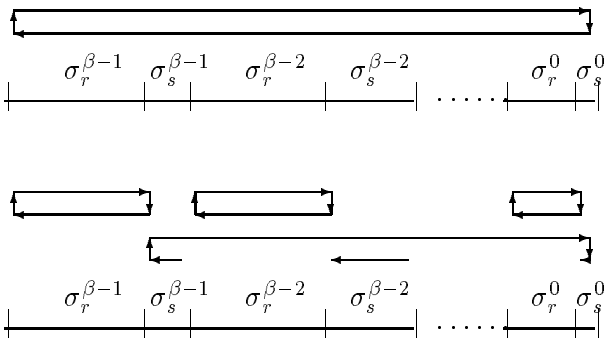


Figure 7: Factoring of a cycle into independent cycles.

of a number of independent 2-cycles. This permutation is equivalent to matrix transposition, or bit-reversal. We refer to it as all-to-all personalized communication [6]. Each processor holds a unique piece of data for every other processor. Any algorithm for all-to-all personalized communication (AAPC) can be used repeatedly to accomplish the required permutation. If all constituting GSH are of the same order, then the use of any AAPC algorithm is straightforward. In [8] AAPC algorithms are presented that allow for a pipeline delay of $\beta$ cycles between each new AAPC application for $\beta$ mixed GSH.

The difference between the AAPC based algorithms and the concurrent/pipelined algorithms is in the scheduling of exchanges for the local storage, or in the ordering of the loops. This difference results in a difference in the pipeline filling time, which is slightly longer for the AAPC based schedules. The alignment and realignment is the same for both types of algorithms.

**Alignment and realignment**

Let the constituting GSH be $(\alpha^j_{\sigma_r^j}\alpha^j_{\sigma_r^j-1}\ldots\alpha^j_1|\alpha^j_0)$ for $0 \leq j < \beta$, where $\sigma_r^j$ is the real order of the $j$th constituting GSH. Furthermore, let $p^j = \alpha^j_{\sigma_r^j}\alpha^j_{\sigma_r^j-1}\ldots\alpha^j_1$ and $x^j = \alpha^j_{\sigma_r^j} \oplus \alpha^j_{\sigma_r^j-1} \oplus \ldots \oplus \alpha^j_1 \oplus \alpha^j_0$. Then, the alignment operation is the local storage reordering defined by $(p^{\beta-1}p^{\beta-2}\ldots p^0|\alpha_0^{\beta-1}\alpha_0^{\beta-2}\ldots\alpha_0^0) \rightarrow (p^{\beta-1}p^{\beta-2}\ldots p^0|x^{\beta-1}x^{\beta-2}\ldots x^0)$. The realignment is the exact same operation.

If an extra storage dimension $v$ is used for the $j$th constituting GSH, then $x^j_v = \alpha^j_{\sigma_r^j} \oplus \alpha^j_{\sigma_r^j-1} \oplus \ldots \oplus \alpha^j_1 \oplus v$. Moreover, if $\alpha^j_0$ is used for exchanges for some of the storage locations, and $v$ for others, then the same alignment can be performed on both local dimensions, $(p^j|\alpha^j_0 v) \rightarrow (p^j|x^j x^j_v)$. This alignment guarantees that for both type of exchanges the sending and receiving processors use the same local storage address.

**Lemma 3** *The alignment for different constituting GSH can be made on the same exchange dimension $v$ preserving the property that exchanges are always made between locations with the same local address.*

The lemma follows from the fact that the exchanges for each constituting GSH take place within subcubes for which all other processor address bits are the same. However, by performing the alignment for different GSH on the same storage dimension the local addresses are not the

| | | Time step | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $d_0$ | $0,1$ | $0,2$ | $0,3$ | $0,4$ | $0,5$ | | | $0',1'$ | $0',2'$ | $0',3'$ | $0',4'$ | $0',5'$ | | | | |
| M | $d_1$ | | $0,1$ | $0,2$ | $0,3$ | $0,4$ | $0,5$ | | | $0',1'$ | $0',2'$ | $0',3'$ | $0',4'$ | $0',5'$ | | | |
| | $d_2$ | | | $0,1$ | $0,2$ | $0,3$ | $0,4$ | $0,5$ | | | $0',1'$ | $0',2'$ | $0',3'$ | $0',4'$ | $0',5'$ | | |
| e | $d_3$ | | | | $0,1$ | $0,2$ | $0,3$ | $0,4$ | $0,5$ | | | $0',1'$ | $0',2'$ | $0',3'$ | $0',4'$ | $0',5'$ | |
| | $d_4$ | | | | | $0,1$ | $0,2$ | $0,3$ | $0,4$ | $0,5$ | | | $0',1'$ | $0',2'$ | $0',3'$ | $0',4'$ | $0',5'$ |
| m | $d_5$ | $0',1'$ | $0',2'$ | $0',3'$ | $0',4'$ | $0',5'$ | $0,1$ | $0,2$ | $0,3$ | $0,4$ | $0,5$ | | | | | | |
| | $d_6$ | | $0',1'$ | $0',2'$ | $0',3'$ | $0',4'$ | $0',5'$ | $0,1$ | $0,2$ | $0,3$ | $0,4$ | $0,5$ | | | | | |
| o | $d_7$ | | | $0',1'$ | $0',2'$ | $0',3'$ | $0',4'$ | $0',5'$ | $0,1$ | $0,2$ | $0,3$ | $0,4$ | $0,5$ | | | | |
| | $d_8$ | | | | $0',1'$ | $0',2'$ | $0',3'$ | $0',4'$ | $0',5'$ | $0,1$ | $0,2$ | $0,3$ | $0,4$ | $0,5$ | | | |
| r | $d_9$ | | | | | $0',1'$ | $0',2'$ | $0',3'$ | $0',4'$ | $0',5'$ | $0,1$ | $0,2$ | $0,3$ | $0,4$ | $0,5$ | | |
| | $d_{10}$ | | | | | | $0',1'$ | $0',2'$ | $0',3'$ | $0',4'$ | $0',5'$ | $0,1$ | $0,2$ | $0,3$ | $0,4$ | $0,5$ | |
| y | $d_{11}$ | | | | | | | $0',1'$ | $0',2'$ | $0',3'$ | $0',4'$ | $0',5'$ | $0,1$ | $0,2$ | $0,3$ | $0,4$ | $0,5$ |

Figure 8: The exchange sequences for $\mathcal{J} = \{\alpha_5\alpha_4\alpha_3\alpha_2\alpha_1|\alpha_0\}$, $\mathcal{J}' = \{\alpha'_5\alpha'_4\alpha'_3\alpha'_2\alpha'_1|\alpha'_0\}$, and $K = 24$ using pipelining.

same in all subcubes.

### Pairing of local memory locations

In the exchanges in any dimension only half of the local data is exchanged. For a single GSH one local memory bit ($v$ or $\alpha_0$) is used to control the exchange, and either all the local data with the bit set or not set is exchanged. It is convenient to view storage locations in pairs, where one location or the other in a pair is exchanged. When more than one local storage dimension is used to control the exchanges, then the pairing of locations shall be made with respect to all such dimensions, i.e., the pairing shall be made on $v$, $\alpha_0^0$, $\alpha_0^1$, ..., and $\alpha_0^{\beta-1}$. For locations with a first exchange for each constituting GSH in its first processor dimension $v$ need not be included in the alignment and the pairing. The values of the local storage dimensions in a pair are complements of each other. For instance, for three control dimensions the pairs are $(000, 111)$, $(001, 110)$, $(010, 101)$, and $(011, 100)$. If there are additional local storage dimensions the pairing is simply repeated as many times as necessary.

### Pipelined Algorithms

*Concurrent pipelined single GSH algorithms*

The algorithm for a single mixed GSH can be generalized to multiple GSH. With *n-port communication* and sufficiently many local data elements, the different constituting GSH can be initiated and performed concurrently. Figure 8 illustrates the case where all exchange sequences start in the first processor dimension of each GSH. The number of element exchanges in sequence is $\frac{K}{2} + \frac{\sigma_r}{\beta} - 1$, if all constituting GSH are of the same real order, and $K \geq 2\sigma_r$. If the constituting GSH are of different order, then the number of exchanges are $\frac{K}{2} + \max_j \sigma_r^j - 1$, where $(\sum_{j=0}^{\beta-1} \sigma_r^j = \sigma_r)$. If $K \leq 2\sigma_r$ then the number of exchanges in sequence is $\lceil \frac{K}{2\beta} \rceil + \sigma_r - 1$, whether or not all constituting GSH are of the same order. The local storage is partitioned into blocks of $\lfloor \frac{K}{2\beta} \rfloor$ and $\lceil \frac{K}{2\beta} \rceil$ locations. The data in partition $j$, $0 \leq j < \beta$ is permuted according to constituting GSH $j, (j+1) \bmod \beta, \cdots, (j-1) \bmod \beta$.

Figure 9 illustrates the case in which some exchange sequences start in a dimension other than the first of any constituting GSH. The number of element exchanges in sequence is $\frac{K}{2} + 2$, if $K \geq 2(\sigma_r + 1)$. For $K \leq 2(\sigma_r + 1)$ the number of exchanges is $\sigma_r + 3$. The algorithm does not

| | | Time step | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $d_0^0$ | $v,5$ | $v,6$ | • | | $0',1'$ | $0',2'$ | $0',3'$ | $0',4'$ | $0',5'$ | $0',6'$ | $0,1$ | $0,2$ | $0,3$ | $0,4$ | $0,5$ | |
| M | $d_0^1$ | | $v,5$ | $0,6$ | | | $0',1'$ | $0',2'$ | $0',3'$ | $0',4'$ | $0',5'$ | $0',6'$ | $0,1$ | $0,2$ | $0,3$ | $0,4$ | $0,5$ |
| | $d_1^0$ | $v,3$ | $v,4$ | $v,5$ | $v,6$ | • | | $0',1'$ | $0',2'$ | $0',3'$ | $0',4'$ | $0',5'$ | $0',6'$ | $0,1$ | $0,2$ | $0,3$ | |
| e | $d_1^1$ | | $v,3$ | $v,4$ | $v,5$ | $v,6$ | | | $0',1'$ | $0',2'$ | $0',3'$ | $0',4'$ | $0',5'$ | $0',6'$ | $0,1$ | $0,2$ | $0,3$ |
| | $d_2$ | $0,1$ | $0,2$ | $0,3$ | $0,4$ | $0,5$ | $0,6$ | | | $0',1'$ | $0',2'$ | $0',3'$ | $0',4'$ | $0',5'$ | $0',6'$ | | |
| m | $d_3$ | | $0,1$ | $0,2$ | $0,3$ | $0,4$ | $0,5$ | $0,6$ | | | $0',1'$ | $0',2'$ | $0',3'$ | $0',4'$ | $0',5'$ | $0',6'$ | |
| | $d_4^0$ | $v,5'$ | $v,6'$ | $0,1$ | $0,2$ | $0,3$ | $0,4$ | $0,5$ | $0,6$ | | | $0',1'$ | $0',2'$ | $0',3'$ | $0',4'$ | $0',5'$ | |
| o | $d_4^1$ | | $v,5'$ | $v,6'$ | $0,1$ | $0,2$ | $0,3$ | $0,4$ | $0,5$ | $0,6$ | • | | $0',1'$ | $0',2'$ | $0',3'$ | $0',4'$ | $0',5'$ |
| | $d_5^0$ | $v,3'$ | $v,4'$ | $v,5'$ | $v,6'$ | $0,1$ | $0,2$ | $0,3$ | $0,4$ | $0,5$ | $0,6$ | | | $0',1'$ | $0',2'$ | $0',3'$ | |
| r | $d_5^1$ | | $v,3'$ | $v,4'$ | $v,5'$ | $v,6'$ | $0,1$ | $0,2$ | $0,3$ | $0,4$ | $0,5$ | $0,6$ | • | | $0',1'$ | $0',2'$ | $0',3'$ |
| | $d_6$ | $0',1'$ | $0',2'$ | $0',3'$ | $0',4'$ | $0',5'$ | $0',6'$ | $0,1$ | $0,2$ | $0,3$ | $0,4$ | $0,5$ | $0,6$ | | | | |
| y | $d_7$ | | $0',1'$ | $0',2'$ | $0',3'$ | $0',4'$ | $0',5'$ | $0',6'$ | $0,1$ | $0,2$ | $0,3$ | $0,4$ | $0,5$ | $0,6$ | | | |
| | $d_8$ | | | $0',1'$ | $0',2'$ | $0',3'$ | $0',4'$ | $0',5'$ | $0',6'$ | $0,1$ | $0,2$ | $0,3$ | $0,4$ | $0,5$ | $0,6$ | | |
| | $d_9$ | | | | $0',1'$ | $0',2'$ | $0',3'$ | $0',4'$ | $0',5'$ | $0',6'$ | $0,1$ | $0,2$ | $0,3$ | $0,4$ | $0,5$ | $0,6$ | |

Figure 9: The exchange sequences for $\mathcal{J} = \{\alpha_6\alpha_5\alpha_4\alpha_3\alpha_2\alpha_1|\alpha_0\}$, $\mathcal{J}' = \{\alpha'_6\alpha'_5\alpha'_4\alpha'_3\alpha'_2\alpha'_1|\alpha'_0\}$, and $K = 28$.

require that all constituting GSH are of the same order.

**Theorem 2** *Any generalized shuffle of real order $\sigma_r > 0$ can be realized in at most*

$$T = \begin{cases} \lceil \frac{K}{2\beta} \rceil + \sigma_r - 1 & \max_i \sigma_r^i \geq 4, K \leq 8\beta, \text{ or } \max_i \sigma_r^i \leq 3, K \leq 2\sigma_r, \\ \frac{K}{2} + \max_i \sigma_r^i - 1 & K \geq 2\sigma_r, \ \max_i \sigma_r^i \leq 3 \\ \sigma_r + 3 & \max_i \sigma_r^i \geq 4, \ 8\beta < K \leq 2(\sigma_r + 1), \\ \frac{K}{2} + 2, & K \geq 2(\sigma_r + 1), \max_i \sigma_r^i > 3. \end{cases}$$

*element exchanges in sequence with n-port communication. The minimum number of communication start-ups is $\sigma_r$ for a block size of $\lceil \frac{K}{2\beta} \rceil$ and $\sigma_r + 3$ for a block size of $\lceil \frac{K}{2(\sigma_r+1)} \rceil$.*

*All-to-all personalized communication algorithms*

Employing an AAPC algorithm with a delay of $\beta$ exchanges between successive AAPCs on different dimensions [8] yields $\frac{K}{2} + \sigma_r - \beta$ exchanges in sequence, if all constituting GSH are of the same order, $\frac{\sigma_r}{\beta}$. As in previous algorithms some storage locations can start their exchange sequences in dimensions other than the first of any constituting GSH, thereby reducing the pipeline delay [9]. For the AAPC algorithms blocks of storage locations are scheduled together. The block sizes depend upon the AAPC algorithm chosen.

$$T = \begin{cases} \frac{K}{2} + \sigma_r - \beta & \sigma_r \leq 3\beta, \text{ or } \sigma \geq 4\beta, K \leq 8\beta \text{ (pipelining)} \\ \sigma_r + 3\beta & \sigma \geq 4\beta, 8\beta \leq K \leq 2(\sigma_r + \beta) \text{ (concurrent exchange)} \\ \frac{K}{2} + 2\beta & \sigma_r \geq 4\beta, K \geq 2(\sigma_r + \beta) \text{ (concurrent exchange with pipelining)} \end{cases}$$

The use of AAPC algorithms for a succesion of AAPC's of different orders is not entirely straightforward. If the orders are non-increasing and the number of dimensions in one AAPC does not divide the number of dimensions in the preceding AAPC, then a delay of up to one less than the number of dimensions in the AAPC to be performed is introduced for some algorithms, while other algorithms may require an even greater delay [9].

| Algorithm | Constituting GSH |
|---|---|
| | *one-port communication* |
| | $\sigma_r \frac{K}{2} t_c + \sigma_r \lceil \frac{K}{2B} \rceil \tau$ |
| | *n-port communication* |
| | same order |
| pipelined AAPC | $(\lceil \frac{K}{2B} \rceil + \sigma_r - \beta)(Bt_c + \tau)$ |
| pipelined GSH | $\begin{cases} (\lceil \frac{K}{2\beta} \rceil + (\sigma_r - 1)B)t_c + (\lceil \frac{K}{2\beta B} \rceil + \sigma_r - 1)\tau, & K \leq 2B\sigma_r \\ (\lceil \frac{K}{2} \rceil + (\frac{\sigma_r}{\beta} - 1)B)t_c + (\lceil \frac{K}{2B} \rceil + \frac{\sigma_r}{\beta} - 1)\tau, & K \geq 2B\sigma_r \end{cases}$ |
| concurrent pipelined GSH | $\begin{cases} (\lceil \frac{K}{2\beta} \rceil + (\sigma_r - 1)B)t_c + (\lceil \frac{K}{2\beta B} \rceil + \sigma_r - 1)\tau, & \sigma_r \geq 4\beta, K \leq 8\beta B, \\ & \text{or } \sigma_r \leq 3\beta, K \leq 2B\sigma_r \\ (\lceil \frac{K}{2} \rceil + (\frac{\sigma_r}{\beta} - 1)B)t_c + (\lceil \frac{K}{2B} \rceil + \frac{\sigma_r}{\beta} - 1)\tau, & \sigma_r \leq 3\beta, K \geq 2B\sigma_r \\ (\sigma_r + 3)(Bt_c + \tau), & 8\beta B < K \leq 2B(\sigma_r + 1) \\ (\frac{K}{2} + 2B)t_c + (\lceil \frac{K}{2B} \rceil + 2)\tau, & K \geq 2B(\sigma_r + 1), \sigma_r > 3\beta \end{cases}$ |
| | different order |
| pipelined GSH | $\begin{cases} \lceil \frac{K}{2\beta} \rceil + \sigma_r - 1, & K \leq 2\sigma_r \\ \frac{K}{2} + \max_i \sigma_r^i - 1, & K \geq 2\sigma_r \end{cases}$ |
| concurrent pipelined GSH | $\begin{cases} \lceil \frac{K}{2\beta} \rceil + \sigma_r - 1, & \max_i \sigma_r^i \geq 4, K \leq 8\beta \text{ or } \max_i \sigma_r^i \leq 3, K \leq 2\sigma_r \\ \frac{K}{2} + \max_i \sigma_r^i - 1, & \max_i \sigma_r^i \leq 3, K \geq 2\sigma_r \\ \sigma_r + 3, & \max_i \sigma_r^i \geq 4,\ 8\beta < K \leq 2(\sigma_r + 1) \\ \frac{K}{2} + 2, & K \geq 2(\sigma_r + 1), \max_i \sigma_r^i > 3 \end{cases}$ |

Table 2: The number of element transfer in sequence for concurrent/pipelined algorithms for multiple mixed GSH.

*Comparison of algorithms*

The pipelined GSH algorithm is always preferable over the pipelined AAPC algorithm. Similarly, the strategy to combine concurrent exchange sequences with pipelining always yields a better result for the algorithms completing each constituting GSH for each pair of data elements before initiating another constituting GSH, than the same strategy applied to AAPC based algorithms. The pipeline filling time is longer for the latter algorithms. The complexity estimates are summarized in Table 2. For all constituting GSH of the same order we have included the effects of limited communication buffers and a start-up overhead for each communication action.

## 4.2 Real shuffle algorithms

A mixed GSH has at least one local storage dimension as part of the permutation. In a real GSH no local storage dimension is part of the permutation. But, if there are at least two data elements per processor, then using an algorithm with the storage dimension as a fixed exchange dimension allows each exchange to involve only one processor dimension, as in the algorithms above for mixed GSH. The real shuffle $(\alpha_{\sigma_r-1}\alpha_{\sigma_r-2}\ldots\alpha_0|v) \rightarrow (\alpha_{\sigma_r-2}\alpha_{\sigma_r-3}\ldots\alpha_0\alpha_{\sigma_r-1}|v)$ performed through a sequence of exchanges between $v$ and a processor dimension requires a minimum of $\sigma_r + 1$ exchanges [15], Algorithm A1, Figure 10. The number of element exchanges in sequence is $(\sigma_r + 1)\frac{K}{2}$ for *one-port communication*. For *n-port communication* pipelining or concurrent exchange sequences can be used. Pipelining requires $\frac{K}{2} + \max(\frac{K}{2}, \sigma_r)$ element exchanges in sequence, whereas the use of up to $\sigma_r$ concurrent exchange sequences leads to $(\sigma_r + 1)\lceil \frac{K}{2\sigma_r} \rceil$

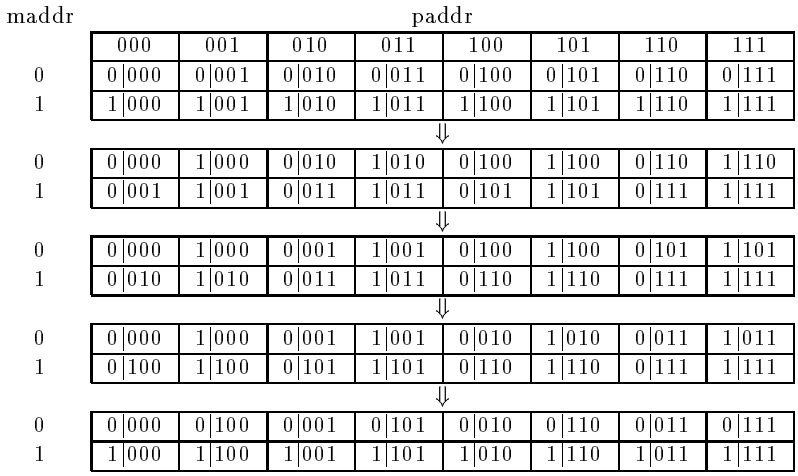


Figure 10: A shuffle permutation of order $\sigma_r$ using $\sigma_r + 1$ communications.

element exchanges in sequence. The pipelined algorithm always requires a larger number of element transfers in sequence, and more communication start-ups, than the algorithm based on concurrent exchange sequences.

The algorithm outlined above using an extra local memory dimension for all exchanges is non-optimal by one exchange. For architectures with a high start-up time relative to the data transfer time, it may be desirable to find an algorithm with the optimal number of start-ups. In algorithm A1 half of the data is at its final destination after $\sigma_r$ exchanges. If the initial content in location zero in processors four through seven, and in location one in processors zero through three did not matter, then the permutation would be completed in $\sigma_r$ steps. In general, at the expense of doubling the memory requirements, and the data transfer time, the minimal number of communications start-ups can be achieved. The first step of the modified algorithm can be accomplished through the exchange

**if** $\alpha_{\sigma_r-1} \neq \alpha_0$ **then** $(\alpha_{\sigma_r-1}\alpha_{\sigma_r-2}\ldots\alpha_0|0) \rightarrow (\alpha_{\sigma_r-1}\alpha_{\sigma_r-2}\ldots\overline{\alpha_0}|1)$

The result is $(\alpha_{\sigma_r-1}\alpha_{\sigma_r-2}\ldots\alpha_0|\alpha_0)$, $\alpha_0 = \alpha_{\sigma_r-1}$ with the other processors being empty. During the communication in dimensions $\alpha_1$ through $\alpha_{\sigma_r-2}$ subcubes $(0\alpha_{\sigma_r-2}\ldots\alpha_1 1)$ and $(1\alpha_{\sigma_r-2}\ldots\alpha_1 0)$ are empty. In the last communication data is sent from subcube $(0\alpha_{\sigma_r-2}\ldots\alpha_1 0)$ to subcube $(1\alpha_{\sigma_r-2}\ldots\alpha_1 0)$, and from subcube $(1\alpha_{\sigma_r-2}\ldots\alpha_1 1)$ to subcube $(0\alpha_{\sigma_r-2}\ldots\alpha_1 1)$. $K$ elements are sent in the first and last communication, and exchanged in the $\sigma_r - 2$ other communications. This is algorithm A2.

The case with $\sigma_r = 2$ represents a special case for algorithm A2. Each channel is only used in one direction. Splitting the data set $K$ into two parts allows both minimum length paths to be used. For two equal parts the complexity is $(\frac{K}{2} + B)t_c + (\lceil\frac{K}{2B}\rceil + 1)\tau$. This is algorithm A2[2]. With an insignificant communication start-up, such as on the Connection Machine, the optimal value of $B$ is one, and only one element exchange in excess of the lower bound is required. The result generalizes to the case where the real shuffle consists of a number of independent cycles

| Comm. | Alg. | $B_{opt}$ | Mem | Communication complexity | $t_c$ factor/$l\,b$ | $\tau$ factor/$l\,b$ |
|---|---|---|---|---|---|---|
| *one-port* | A1 | $\frac{K}{2}$ | $K$ | $(\sigma_r+1)\frac{K}{2}t_c + (\sigma_r+1)\lceil\frac{K}{2B}\rceil\tau$ | (1, 1.5] | (1, 1.5] |
| comm. | A2 | $K$ | $2K$ | $\sigma_r K t_c + \sigma_r\lceil\frac{K}{B}\rceil\tau$ | 2 | 1 |
| *n-port* | A1 | $\lceil\frac{K}{2\sigma_r}\rceil$ | $K$ | $(\sigma_r+1)\lceil\frac{K}{2\sigma_r}\rceil t_c + (\sigma_r+1)\lceil\frac{K}{2\sigma_r B}\rceil\tau$ | (1, 1.5] | (1, 1.5] |
| comm. | A2 | $\lceil\frac{K}{\sigma_r}\rceil$ | $2K$ | $Kt_c + \sigma_r\lceil\frac{K}{\sigma_r B}\rceil\tau$ | 2 | 1 |
| | A2[2] | $\sqrt{\frac{K\tau}{2t_c}}$ | $K+B$ | $(\frac{K}{2}+B)t_c + (\lceil\frac{K}{2B}\rceil+1)\tau$ | $1+\frac{2B}{K}$ | 1 |

Table 3: The memory requirements and communication complexities for a real GSH of order $\sigma_r$.

of length two.

The complexity expressions are summarized in Table 3. The break-even point between algorithms A1 and A2 for unbounded buffer sizes $B$ and $\sigma_r > 2$ is $\tau = (\sigma_r - 1)\frac{K}{2}t_c$ for *one-port communication*, and $\tau = (1 - \frac{1}{\sigma_r})\frac{K}{2}t_c$ for *n-port communication*.

# 5 Experiments

We have implemented the *one-port* version of algorithm A1 for real GSH on the Intel iPSC/1, and algorithm A2[2] on the Connection Machine (in the context of a bit-reversal routine). On the Intel iPSC/1 we also performed shuffle permutations by using the routing logic, by using all-to-all personalized communication twice, and by an algorithm that requires precisely $\sigma_r$ start-ups for a real shuffle of order $\sigma_r$ (algorithm A1 in [5]). Algorithm A2 presented above should behave similarly for small values of $K$, and be superior for large values of $K$, but was not known at the time of the implementation. For a real, sub-cube, shuffle permutation with a message size less than a few hundred bytes, algorithm A2 is the fastest, but for larger message sizes algorithm A1 is preferable, Figure 11. 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 Figure 11 is due to a hybrid implementation, that optimizes the sum of start-up time and the time for local data movement [4].

For the Connection Machine implementation of algorithm A2[2] the measured time complexity is $270 + (3.2 + 3.9\beta)K$ $\mu$sec, where $\beta$ is the number of 2-cycles in the real GSH.

# 6 Summary and conclusions

The algorithms for *mixed* shuffle permutations are optimal within two element exchanges for concurrent communication on all channels of every processor. The number of communication start-ups for an unlimited buffer size is either exactly optimal, or requires three excess start-ups

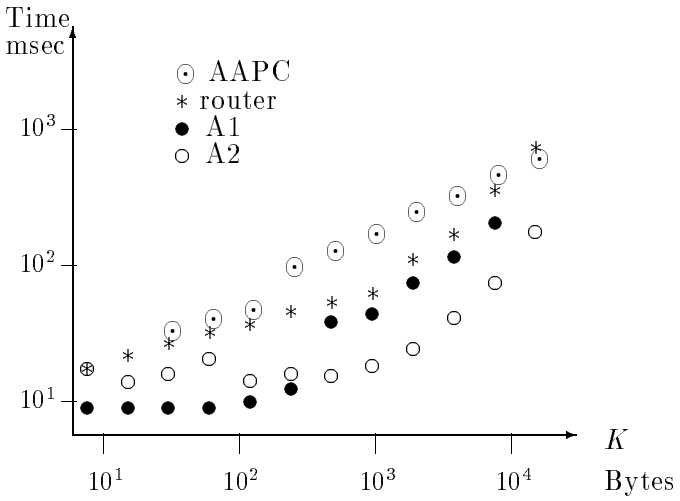


Figure 11: The measured shuffle times as a function of message lengths on an iPSC/1 5-cube.

depending upon the size of the local data set relative to the number of processor dimensions in the generalized shuffle, and the number of blocks of contiguous processor dimensions. The communication complexity for *n-port communication* is summarized in Table 2, page 16. For *one-port communication* the minimum number of start-ups is $\sigma_r$, and the minimum number of element transfers in sequence is $\sigma_r \frac{K}{2}$. The mixed shuffle algorithms are also valid when bit-complementation is required by appropriately modifying which data in a pair is exchanged.

We also show that by performing a local alignment before and after the data exchanges for mixed GSH, all data exchanged have the same relative address.

The communication complexities of a *real* GSH of order $\sigma_r$ are summarized in Table 3, page 18. The second and last columns contain the values of the data transfer times and start-up times relative to the respective lower bound. With an unbounded buffer size the number of start-ups is either exactly optimal, or suboptimal by one start-up. The number of element transfers in sequence exceeds the lower bound by a factor of $\frac{1}{\sigma_r}$ for the best algorithm, except if $\sigma_r = 2$, or there are a number of 2-cycles. Then, only one element transfer in excess of the lower bound is required. This result is not valid if bit-complementation is required.

Performing a GSH through *all-to-all personalized communication* [6, 14] is always inferior to the pipelined/concurrent algorithms presented here. Likewise, performing the permutation by recursively applying an optimal matrix transposition algorithm [4] yields higher complexity.

Finally we note that the control is identical for all processors, and can easily be distributed.

## Acknowledgement

The authors would like to thank the anonymous referees for several valuable suggestions that helped improve the manuscript, and for bringing important references to our attention.

The research reported here was in part supported by the Office of Naval Research under Contract No. N00014-86-K-0310, by the AFOSR under contract AFOSR-89-0382 and by NSF/DARPA under contract CCR-8908285. The Connection Machine implementation was made by Alan Edelman, Steve Heller and Mark Bromley, and is part of the CM System Software.

## 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. Optimal algorithms for stable dimension permutations on Boolean cubes. In *The Third Conference on Hypercube Concurrent Computers and Applications*, pages 725–736. ACM, 1988.

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

[4] 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.

[5] S. Lennart Johnsson and Ching-Tien Ho. Shuffle permutations on Boolean cubes. Technical Report YALEU/DCS/RR-653, Department of Computer Science, Yale University, October 1988.

[6] S. Lennart Johnsson and Ching-Tien Ho. Spanning graphs for optimum broadcasting and personalized communication in hypercubes. *IEEE Trans. Computers*, 38(9):1249–1268, September 1989.

[7] S. Lennart Johnsson and Ching-Tien Ho. The complexity of reshaping arrays on Boolean cubes. In *The Fifth Distributed Memory Computing Conference*, pages 370–377. IEEE Computer Society, April 1990.

[8] S. Lennart Johnsson and Ching-Tien Ho. Maximizing channel utilization for all-to-all personalized communication on Boolean cubes. In *The Sixth Distributed Memory Computing Conference*, pages 299–304. IEEE Computer Society Press, 1991.

[9] S. Lennart Johnsson and Ching-Tien Ho. Optimal communication channel utilization for matrix transposition and related permutations on Boolean cubes. *Discrete Applied Mathematics*, 1992.

[10] David Nassimi and Sartaj Sahni. An optimal routing algorithm for mesh-connected parallel computers. *JACM*, 27(1):6–29, January 1980.

[11] David Nassimi and Sartaj Sahni. Optimal BPC permutations on a cube connected SIMD computer. *IEEE Trans. Computers*, C-31(4):338–341, April 1982.

[12] 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.

[13] Quentin F. Stout and Bruce Wagar. 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.

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

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