Yale University
Department of Computer Science

Optimal Algorithms for Stable Dimension Permutations on Boolean Cubes

Ching-Tien Ho and S. Lennart Johnsson

YALEU/DCS/TR-620
March 1988

This work has in part been supported by the Office of Naval Research under contract N00014-86-K-0564. Approved for public release: distribution is unlimited.

† To appear in the Proceedings for the Third Conference on Hypercube Concurrent Computers and Applications, January, 1988, Pasadena, CA.

Optimal Algorithms for Stable Dimension Permutations on Boolean Cubes

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

Abstract. In this paper we present algorithms optimal within a small constant factor for *stable dimension permutations* on Boolean cubes. A *stable dimension permutation* is a permutation such that element $(w_{m-1}w_{m-2}\ldots w_0)$ is relocated to the location of element $(w_{\delta(m-1)}w_{\delta(m-2)}\cdots w_{\delta(0)})$ after the permutation, or $i \rightarrow \delta(i)$, where $\delta(\cdot)$ is a permutation function on $\{0,1,\ldots, m-1\}$. 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 bandwidth of the Boolean cube is fully explored by dividing the data set to be communicated between a pair of processors into subsets, one for each path between the pair of processors. The *k-shuffle* permutation, the *bit-reversal* permutation, and matrix transposition, are special cases of *stable dimension permutations*. Experimental results on the Intel iPSC are also provided.

1 Introduction

Dimension permutations are a class of permutations where data elements are moved according to the value of individual bits in their address. We assume that the address space is implemented on a Boolean n-cube, where each node has substantial local storage. The access time to local storage is assumed negligible compared to the access time to storage in another processor.

In this paper, we give lower bounds and algorithms optimal within a small constant factor for *stable dimension permutations*, i.e., dimension permutations within (sub)cubes such that every node holds data both before and after the permutation. A more extensive treatment of stable dimension permutations is given in [4], and unstable dimension permutations are treated in [12]. 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 stable dimension permutations are k-shuffle/unshuffle permutations, matrix transposition [6], [9], bit-reversal [11], and conversion between various data structures, such as between consecutive and cyclic storage [6], [9]. Shuffle operations can be used to reconfigure a two dimensional partitioning to a three dimensional partitioning of a matrix for multiplication

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

with maximum concurrency [8], and for data (re)alignment for certain Fast Fourier Transform algorithms [7], [11].

Stable dimension permutations have also been studied by Flanders [1] on mesh-connected array processors, and by Swarztrauber [16] on Boolean cubes. The notation and definitions used throughout the paper are introduced in Section 2, and lower bounds are given in Section 3. Stable dimension permutation algorithms are described in Section 4. Section 5 discusses some implementation issues on the Intel iPSC. Conclusion are found in Section 6.

2 Preliminaries

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 w and w' with binary encodings $w = (w_{m-1}w_{m-2}\ldots w_0)$ and $w' = (w'_{m-1}w'_{m-2}\ldots w'_0)$ is $Hamming(w, w') = \sum_{i=0}^{m-1}(w_i \oplus w'_i)$.

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 $Hamming(x, y) + 2$ [13]. $\mathcal{N}$ is used to denote the set $\{0, 1, \ldots, 2^n - 1\}$. $||w||$ denotes the number of 1-bits in the binary representation of w, i.e., $||w|| = Hamming(w, 0)$.

For the algorithm design and the complexity analysis we distinguish between the *machine address space* $\mathcal{A}$ and the *logic address space* $\mathcal{L}$. The *machine address space* $\mathcal{A} = \{(a_{q-1}a_{q-2}\ldots a_0) | a_i = 0, 1;\ 0 \leq i < q\}$ is the Cartesian product of the *processor address space* and *local memory address space*. The processor address space requires n bits, or *dimensions*, for a unique encoding in an n-cube. The machine address space covers 2^q addressable objects, which we refer to as elements. The storage per node is 2^{q-n} elements. The machine address space is divided such that 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* $\mathcal{L} = \{(w_{m-1}w_{m-2}\ldots w_0) | w_i = 0, 1;\ 0 \leq i < m\}$ encodes a set of $|\mathcal{L}| = 2^m$ elements. The address of an element is $w = (w_{m-1}w_{m-2}\ldots w_0)$. The set of *logic dimensions* is $\mathcal{M} = \{0, 1, \ldots, m-1\}$. Clearly, $m \leq q$.

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, $|\mathcal{R}| = m_p \leq n$, $|\mathcal{V}| = m_s \leq q-n$, $\mathcal{R} \cup \mathcal{V} = \mathcal{M}$, $\mathcal{R} \cap \mathcal{V} = \phi$, $m_p + m_s = m$. We also define $\Gamma_p = \{\pi(i) | \forall i \in \mathcal{R}\}$, the set of processor dimensions used for the allocation of the set of elements, $\Gamma_s = \{\pi(i) | \forall i \in \mathcal{V}\}$, the set of local storage dimensions used for the allocation of the set of elements, and $\Gamma = \Gamma_p \cup \Gamma_s$. 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. We will refer to the dimensions in $\mathcal{R}$ as *real dimensions* and the dimensions in $\mathcal{V}$ as *virtual dimensions.*

Definition 3 The *real distance* between two elements with addresses w and w', $w, w' \in \mathcal{L}$, is $Hamming_r(w, w') = \sum_{j=0}^{m_{\mathbf{R}}-1} (w_{r_j} \oplus w'_{r_j})$ and the *virtual distance* between w and w' is $Hamming_v(w, w') = \sum_{j=0}^{m_s-1} (w_{v_j} \oplus w'_{v_j})$.

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

If the dimension allocation function is such that 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* [6].

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 and $\mathcal{R}^a$ 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 $\Gamma^b = \Gamma^b_p \cup \Gamma^b_s$ and $\Gamma^a = \Gamma^a_p \cup \Gamma^a_s$, where $\Gamma^b_p, \Gamma^a_p \subseteq \mathcal{Q}_p$ and $\Gamma^b_s, \Gamma^a_s \subseteq \mathcal{Q}_s$. Clearly $|\Gamma^b| = |\Gamma^a|$, since the number of elements is conserved. If $\Gamma^b = \Gamma^a$ (i.e., $\Gamma^b_p = \Gamma^a_p$ and $\Gamma^b_s = \Gamma^a_s$) then the dimension permutation is *stable*; otherwise, it is *unstable.* In the following we only consider *stable* dimension permutations, and due to the space limitation we omit several proofs [4]. Unstable permutations are treated in [5].

Definition 4 A *stable dimension permutation* (SDP), δ, on a logic address space $\mathcal{M}$ is a one-to-one mapping $\mathcal{M} \rightarrow \mathcal{M}$ with $\delta = \pi^{-b} \circ \pi^a$. (π^{-b} denotes $(\pi^b)^{-1}$.) The *index set* $\mathcal{J}$ of the dimension permutation is the set of logic dimensions $\{i | \delta(i) \neq i\} = \mathcal{J}$. The *order* of the dimension permutation is $\sigma = |\mathcal{J}|$. Alternatively, one can define a *dimension permutation*, δ', on the set of machine dimensions, i.e., $\delta' : \Gamma \rightarrow \Gamma$ with $\delta' = \pi^b \circ \pi^{-a}$.

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

The order of the identity permutation is 0. A subscript σ on δ, δ_σ, is used to denote the order of the SDP being σ, whenever necessary. The permutation function δ applies to the set of processor dimensions. The notation for the permutation on data elements is $dp(w)$.

Throughout the paper we define the SDP on the logic address space (δ), but an SDP can also be defined on the machine dimensions (δ'). An SDP can be viewed as relocating logic dimension i to the machine dimension that was assigned to logic dimension $\delta(i)$. Let j and i be

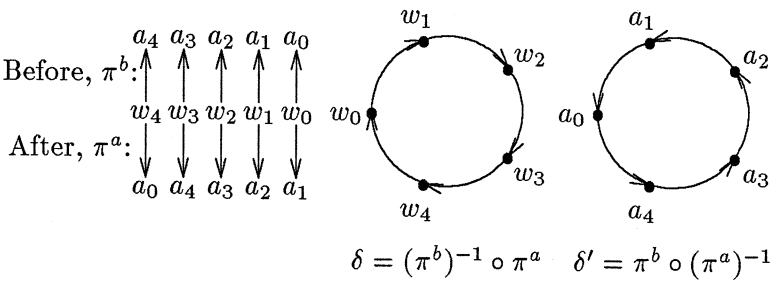

Figure 1: The definition of δ and δ' and cycles formed by traversing δ and δ'.

logic dimensions and k a machine dimension such that $\pi^b(j) = k = \pi^a(i)$, i.e., logic dimension j is assigned to the machine dimension k before the SDP, and logic dimension i is assigned to the machine dimension k after the SDP. Clearly, $i \rightarrow \delta(i) = \pi^{-b} \circ \pi^a(i) = j$. The address for element $(w_{m-1}w_{m-2}\ldots w_0)$ becomes the address for element $(w_{\delta^{-1}(m-1)}w_{\delta^{-1}(m-2)}\cdots w_{\delta^{-1}(0)})$. Figure 1 shows a shuffle permutation. 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 an SDP 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 an SDP for which the data set only occupies a fraction of the cube: $\Gamma_p \subset \mathcal{Q}_p$.

Definition 7 A *shuffle permutation* sh^1 is an SDP such that $\delta_m(j) = (j+1) \bmod m, \forall j \in \mathcal{M}$, and an *unshuffle permutation* sh^{-1} is an SDP such that $\delta_m(j) = (j-1) \bmod m, \forall j \in \mathcal{M}$. A *k-shuffle* is the permutation $sh^k = sh^1 \circ sh^{k-1}$.

A shuffle permutation is a left cyclic shift on the logic dimensions, and an unshuffle permutation is a right cyclic shift. The index set of the shuffle permutation is the entire set of dimensions. The order of a shuffle permutation is m. Clearly, $sh^1 \circ sh^{-1} = sh^{-1} \circ sh^1 = I$. Furthermore, $sh^k = I, \forall k \bmod m = 0$.

Definition 8 A *generalized shuffle permutation* (GSH) of order σ is a shuffle permutation 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 $\delta_\sigma(i) = i$, $\forall i \in \mathcal{M} - \mathcal{J}$.

For a GSH the index set is a subset of the logic dimensions, and the mapping of dimensions under the permutation function is not necessarily strictly increasing, or decreasing. However, we require that the GSH forms a single cycle over the dimensions in the index set. An SDP consists of a number of independent GSH (or cycles). The number of independent GSH's (cycles) for a given SDP is denoted β, and the order of the ith GSH by σ_i, $0 \leq i < \beta$; $\sigma = \sum_{i=0}^{\beta-1} \sigma_i$.

Definition 9 A *real* GSH on a logic address space is a GSH such that $\mathcal{J} \subseteq \mathcal{R}$. A *virtual* GSH on a logic address space is a GSH such that $\mathcal{J} \subseteq \mathcal{V}$. A GSH that is neither real nor virtual is a *mixed* GSH.

Definition 10 An SDP on a logic address space is *separable*, if it can be decomposed into a sequence of GSH's (cycles) that are either real or virtual; it is *non-separable*, otherwise.

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 *virtual* GSH 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 involves both processor and local storage dimensions in the index set $\mathcal{J}$. We only consider SDP's consisting of real or mixed GSH's in the lower bound analysis. For algorithms, we only consider SDP's consisting of real GSH's. See [4] for discussion of algorithms for SDP's consisting of mixed GSH's.

The reason for distinguishing between the real and virtual permutations, is that different models apply for local storage references and communication between processors. Local references are often considerably faster, and can with good approximation be ignored. In standard random access memories (RAM) the access time is independent of the address, but the inter-processor access time is often a function of the distance.

Definition 11 The *real order* σ_p of an SDP on a logic address space is $|\{j|j \neq \delta(j), j \in \mathcal{R}\}| = |\mathcal{J} \cap \mathcal{R}|$, and its *virtual order* σ_s is $|\{j|j \neq \delta(j), j \in \mathcal{V}\}| = |\mathcal{J} \cap \mathcal{V}|$.

For an SDP such that $\mathcal{R}^b \cap \mathcal{R}^a = \phi$, the permutation is an *all-to-all personalized communication* [10,9]. Moreover, some such permutations are their own inverse. For instance, for a bit-reversal permutation, or the transposition of a matrix partitioned into $\sqrt{N} \times \sqrt{N}$ blocks, $\delta^2(i) = i, \forall i \in \mathcal{M}$ [9].

In summary we view SDP's as composed of a set of GSH's. In devising algorithms and analyzing their complexity we distinguish between *full-cube* and *extended-cube permutations*, and for each we further distinguish between *separable*, and *non-separable* SDP's. Separable SDP's are decomposed into sets of real and virtual GSH's. Non-separable SDP's are decomposed into real, virtual and mixed GSH's.

With each internode communication is associated a 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 SDP's are denoted $T^*_{type}(ports, \sigma_p, m_p, K)$, where *type* is the type of SDP, such as *gsh* for a generalized shuffle permutation, or *dp* for a dimension permutation. The superscript $*$ is either *lb* for a lower bound, or an algorithm identifier for an upper bound. The first argument for T_{dp} is the number of ports per processor used concurrently, the second argument the real order of the SDP, the third argument the number of processor dimensions being used, and the last argument the data volume per processor, i.e., $K = 2^{m_s}$.

3 The complexity of dimension permutations

3.1 Some properties of dimension permutations

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 2 *An SDP 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 SDP, if the original algorithm fully utilizes the bandwidth.*

Proof: Let P and P' be two problems such that $\mathcal{V} = \mathcal{V}'$ and subject to the same SDP, 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'': an SDP 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 2 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

We now give lower bounds for different instances of SDP's. The index sets for each GSH out of which the SDP is composed are disjoint. Each GSH defines a cycle on its index set. For convenience, we assume that a separable SDP consists of only *real* GSH's, and a non-separable SDP consists of only *real* and *mixed* GSH's in the following, since the complexity of virtual GSH's often can be ignored. Let the SDP consist of β *real* or *mixed* GSH's, and let the corresponding index sets be $\mathcal{J}_0, \mathcal{J}_1, \ldots, \mathcal{J}_{\beta-1}$, where $\mathcal{J}_i \cap \mathcal{J}_j = \phi$, $i \neq j$, $\mathcal{J} = \cup_{i=\{0,1,\ldots\beta-1\}}\mathcal{J}_i$, and $\mathcal{J}_i = \{\alpha_{i0}, \alpha_{i1}, \ldots, \alpha_{i(\sigma_i-1)}\}$ and $\delta_\sigma(\alpha_{ij}) = \alpha_{i((j+1) \bmod \sigma_i)}$, $\forall (i,j) \in \{0,1,\ldots,\beta-1\} \times \{0,1,\ldots,\sigma_i-1\}$. Let oJ be the number of sets such that σ_i is odd and $\mathcal{J}_i \subset \mathcal{R}$. Also, let σ_{p_i} be the *real order* of the GSH defined by $\mathcal{J}_i$. Clearly, $\delta_\sigma(i) = i$, $\forall i \in \mathcal{M} - \mathcal{J}$, and $\sigma_p \leq \sum_{i=0}^{\beta-1} \sigma_i = \sigma$.

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 2.

Lemma 3 *The lower bound for a full-cube, real GSH of order σ_p, $\sigma_p > 0$, on an n-cube is*

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

for one-port communication, and

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

for n-port communication.

Proof: Consider the minimum number of start-ups first. Let w be such that $w_{\alpha_i} = 0$, if $i \bmod 2 = 0$, and $w_{\alpha_i} = 1$ otherwise, $0 \leq i < \sigma_p$. It follows that $Hamming_r(w, gsh(w)) = \sigma_p$, 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 $w_{\alpha_i} \neq w_{\alpha_{(i+1) \bmod \sigma_p}}$, $\forall 0 \leq i < \sigma_p$, 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 2, we can consider the bandwidths for each σ_p-cube. For each $i \in \mathcal{J}$ with $\delta(i) = j$, $i \neq j$, and $i, j \in \mathcal{R}$, only half of the nodes need to send elements across cube dimension $\pi^b(i)$. 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 1 *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 and all processors using the same dimension during the same routing cycle.*

Proof: In order to realize the minimum data transfer time described in Lemma 3, all links should be used "effectively" and evenly during every routing cycles. However, during the first and last routing cycles, half of the cube links can not support effective communication for the GSH, assuming cube links are used effectively during all other cycles. The same argument applies to the first and last routing cycles for any cube dimension used by all processors. ∎

Lemma 4 *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 $\max\{Hamming_r(w, gsh(w))\}, \forall w \in \mathcal{M}$. Choose any α_j such that $\alpha_j \in \mathcal{R}^b$ and $\alpha_{(j-1) \bmod \sigma} \in \mathcal{V}^b$. Define w such that $w_{\alpha_i} = 0$, if i is even, $0 \leq i < \sigma$; and $w_{\alpha_i} = 1$, otherwise. Moreover, change $w_{\alpha_{(j-1) \bmod \sigma}}$ to be different from w_{α_j} (when σ is odd). Clearly, $Hamming_r(w, gsh(w)) = \sigma_p$ for the w so defined. It is easily seen that $\max_w\{Hamming(w, gsh(w))\} \leq \sigma_p$, since $w_i = w_{\delta(i)}, \forall i \in \mathcal{M} - \mathcal{J}$.

The minimum data transfer time can be proved as in Lemma 3. ∎

Theorem 1 *The lower bound for the communication complexity for a full-cube SDP of real order σ_p on an n-cube is*

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

for one-port communication, and

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

for n-port communication.

Proof: The data transfer time is bounded from below by the total bandwidth requirement divided by the maximum number of links available per routing cycle. By Lemmas 3 and 4, the total bandwidth required for each permutation in subcubes of dimension σ_p (identified by $\mathcal{J}$) is $\sigma_p 2^{\sigma_p - 1} K$. The number of available links per routing step for each subcube is 2^{σ_p} for *one-port* and $\sigma_p 2^{\sigma_p}$ for *n-port communication.*

The minimum number of start-ups is bounded from below by the longest real processor distance between the initial allocation and final allocation for any data item. The maximum real distance between w and $dp(w)$, $\forall w \in \mathcal{L}$ is obtained by maximizing the distance for each index set $\mathcal{J}_i$. By Lemmas 3 and 4, the maximum distance is $\sigma_p - oJ$. ∎

Corollary 2 *The minimum number of start-ups for a full-cube, separable SDP of real order σ_p is at least $\frac{2\sigma_p}{3}$ for any δ_{σ_p} and at most σ_p for some δ_{σ_p}.*

Corollary 3 *With one-port communication, an SDP can be performed as a sequence of GSH's with disjoint index sets without loss of efficiency, assuming the algorithm chosen for the GSH is optimum.*

Proof: The minimum data transfer time (start-up time) of an SDP is the sum of the minimum data transfer time (start-up time) for each of the GSH's of which the SDP consists. ∎

The lower bound in Theorem 1 can be improved for some combinations of K, τ, t_c and σ_i, by considering the sum of the lower bounds of each GSH of which the SDP consists. For the algorithm analysis, we use Theorem 1, which is simpler to evaluate, and for most cases the same as the tighter bound.

Corollary 4 *The lower bound for the communication complexity for an extended-cube SDP ($m_p < n$) of real order σ_p on an n-cube is*

$$T_{dp}^{lb}(1, \sigma_p, m_p, K) = \max\left((K + \sigma_p - oJ - 1)t_c, \frac{\sigma_p \cdot K}{2^{n-m_p+1}}t_c, (\sigma_p - oJ)\tau\right)$$

for one-port communication, and

$$T_{dp}^{lb}(n, \sigma_p, m_p, K) = \max\left((\frac{K}{n - m_p + \sigma_p} + \sigma_p - oJ - 1)t_c, \frac{K}{2^{n-m_p+1}}t_c, (\sigma_p - oJ)\tau\right)$$

for n-port communication.

Proof: By Lemma 2, the lower bound for an SDP 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 DP 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 Theorem 1, 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 for separable dimension permutations

4.1 Overview

In this section we present algorithms for *real generalized shuffle* permutations (GSH) and *separable, stable dimension permutations* SDP ($\sum_{i=0}^{\beta-1} \sigma_{p_i} = \sum_{i=0}^{\beta-1} \sigma_i = \sigma_p = \sigma$). For algorithms of *mixed* GSH and *non-separable* SDP, see [4]. We consider *full-cube permutations* (FCP) and *extended-cube permutations* (ECP). Algorithms of the ECP will utilize algorithms for the FCP as primitives. If an optimal algorithm for a GSH is known for the *one-port communication* case, then an optimal algorithm for an SDP with *one-port communication* is obtained by simply running the algorithms for each GSH making up the SDP sequentially, Corollary 3. In the *n-port communication* case the situation is somewhat more complicated. If an optimal algorithm for a GSH is known, then an optimal algorithm for an SDP is obtained by considering each GSH independently, if either there is no start-up time, or all the GSH's have the same real order. The data set is split into β equal parts, and data of the ith part participates in the GSH's specified by the sequence of index sets: $\mathcal{J}_i, \mathcal{J}_{(i+1) \bmod \beta}, \ldots, \mathcal{J}_{(i-1) \bmod \beta}$. The partition index specifies the index of the first GSH, and all partitions use the same sequence, cyclically. For each partition the data is further subdivided into σ_i parts for maximum bandwidth utilization. Hence, in every step σ_p dimensions are used, which is optimum for full-cube permutations.

If there is a non-zero start-up time, and the GSH's are of different order, then one can still try to find σ_p dimension permutations that can be performed concurrently on different data

sets by considering the total index set properly modified according to the algorithm used (for instance by introducing (multiple) virtual dimensions). In one of the algorithms presented below a *generating path* is defined by simply considering the concatenated index sets for the GSH's of the SDP, and additional edge-disjoint paths created through *rotations* on the dimensions. In the other algorithm GSH's are grouped into sets hierarchically, such that the expected execution time for each group is approximately the same. All groups are subject to permutations concurrently, and data elements permuted according to groups sequentially. Within groups different elements are subject to the GSH's in different order. These two algorithms are referred to as Algorithms A5 and A6, and are described in this section.

Definition 12 A *dimension exchange* function $E(w, i, j)$ is an SDP of order two such that $\delta_2(i) = j$, $\delta_2(j) = i$, $i \neq j$, and $\delta_2(k) = k, \forall k \in \mathcal{M} - \{i, j\}$.

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

$$E(\cdots(E(E(w, i, (i-1) \bmod m), (i-1) \bmod m, (i-2) \bmod m),$$
$$\cdots, (i+2) \bmod m, (i+1) \bmod m) = sh^1(w).$$

A dimension exchange $E(w, i, j)$ requires $Hamming_r(i, j)$ routing cycles. If $i, j \in \mathcal{R}$, then the set of processors are partitioned into four groups with respect to the values of w_i and w_j. The groups are labeled as {00, 11, 01, 10} groups. The dimension exchange operation implies no movement for the processors in the 00- and 11-groups. 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 new algorithms we present are based on a sequence of exchange operations between pairs of processors through communication in the same two dimensions for all pairs. If both dimensions are processor dimensions, then the exchange operation can be realized by two nearest-neighbor communications. If only one of the dimensions is a processor dimension, then the exchange operation can be realized by one nearest-neighbor communication. Algorithm A0 is based on exchanges between processors differing in two address bits. Every dimension, except the first and last, is traversed twice in successive exchanges. By combining the successive communications through a look-ahead scheme the number of start-ups is reduced by a factor of two, approximately. The modified algorithm is referred to as Algorithm A1. 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. The data transfer time is reduced approximately by a factor of two, Algorithm A1a.

By introducing a virtual dimension and using it for every exchange operation nearest-neighbor communication suffices, Algorithm A2. Such an algorithm has been proposed by Swarztrauber [16].

$$E(\cdots(E(E(w,i,(i+1) \bmod m),i,(i+2) \bmod m),\cdots,i,(i-1) \bmod m) = sh^1(w).$$

Dimension permutation algorithms can also be obtained by using a matrix transposition algorithm recursively [9], [15], [14], or by using an *all-to-all personalized communication* twice [15], [10]. The complexity estimates for the different algorithms are summarized in Table 2.

4.2 Full-cube, real shuffle algorithms

4.2.1 One-port communication

Algorithm A0–SH through successive element exchanges at real distance 2. The algorithm performs a shuffle permutation of real order n through a sequence of $n-1$ exchanges on adjacent dimensions. The path from a node of origin x to the corresponding destination node $sh(x)$ is defined by a sequence of dimension pairs:

$$([n-1,n-2],[n-2,n-3],\ldots,[1,0]).$$

Each pair $[i,i-1]$ is interpreted as an exchange operation, if $y_i \neq y_{i-1}$, where y is the node currently holding the data; and a dummy operation, otherwise.

In a 4-cube, an element w initially located in node $x = w$ will travel along the path:

$$(\underline{x_3x_2}x_1x_0) \rightarrow (x_2\underline{x_3x_1}x_0) \rightarrow (x_2x_1\underline{x_3x_0}) \rightarrow (x_2x_1x_0x_3),$$

where the underlined dimensions are subject to exchange during the next step. 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 concurrently. After step k there are 2^k independent shuffle operations of order $n-k$. Each exchange operation consists of $n-2$ independent operations on 2-cubes. 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
```

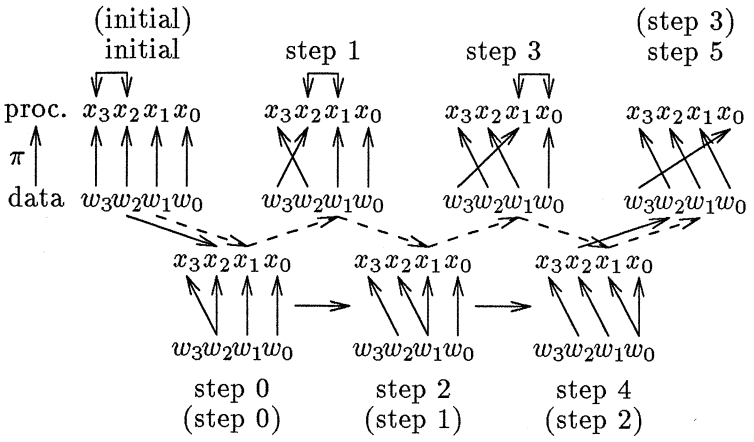

Figure 2: The changes of data allocations along time for Algorithms A0 and A1.

Figure 2 illustrates the changes of data allocations as a function of time on a 4-cube. The three intermediate steps are shown below the exchange steps. The dashed arrows between different allocations show the sequence of communications. The solid arrows between different allocations and the step numbers in parentheses apply to Algorithm A1.

Algorithms A1 (A1a)–SH through successive element exchanges at real distance 2 with look-ahead. Algorithm A0 needs $2(n-1)$ routing cycles for a real shuffle of order n, but the worst case lower bound is n cycles. All dimensions, except dimensions $n-1$ and 0 are subject to exchange operations twice. The number of routing cycles can be reduced to n by combining successive communications along the same dimension and removing redundant communications. The modified routing has n cycles (starting cycle 0), where cycle i, except the first and the last cycles, is the composition of routing cycles $2i-1$ and $2i$ of Algorithm A0.

Consider a dimension j occurring in two consecutive exchange operations $[j+1, j]$ and $[j, j-1]$, $j \neq n-1$ and $j \neq 0$. After the first of the two nearest-neighbor communications of the exchange step $[j+1, j]$, nodes such that $x_{j+1} = x_j$ contain $2K$ elements, while nodes such that $x_{j+1} \neq x_j$ do not contain any elements. The first set of nodes are in the *holding state* and the second set are in the *empty state*. Nodes with $x_{j+1} = x_j \neq x_{j-1}$ need to send data across dimension j (to change from a holding state to an empty state), and nodes with $x_{j+1} \neq x_j = x_{j-1}$ need to receive data across dimension j (to change from an empty state to a holding state). There is no communication for nodes whose states are preserved, i.e., nodes with $x_{j+1} = x_{j-1}$.

From Figure 2, element w initially in node $x = w$ traverses the path

$$(\underline{x_3}x_2x_1x_0) \rightarrow (x_2\underline{x_2}x_1x_0) \rightarrow (x_2\underline{x_3}x_1x_0) \rightarrow (x_2x_1\underline{x_1}x_0)$$
$$\rightarrow (x_2x_1\underline{x_3}x_0) \rightarrow (x_2x_1x_0\underline{x_0}) \rightarrow (x_2x_1x_0x_3)$$

for Algorithm A0. The combined sequence for Algorithm A1 is

$$(\underline{x_3}x_2x_1x_0) \rightarrow (x_2\underline{x_2}x_1x_0) \rightarrow (x_2x_1\underline{x_1}x_0) \rightarrow (x_2x_1x_0\underline{x_0}) \rightarrow (x_2x_1x_0x_3).$$

Note that in each intermediate step at least two bits are always identical. Only $\frac{N}{2}$ processors are addressable in intermediate steps. 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 (i.e., those in the holding state) have $2K$ elements each. 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 shown in Table 2. 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 the communication is capable of supporting one send *and* one receive on different ports concurrently. In Algorithm A1, if a node issues a send (receive) during routing cycle j, then it does not issue a send (receive) during cycles $j-1$ and $j+1$ (if exist). Therefore, the communication during cycle j 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 $2K$,

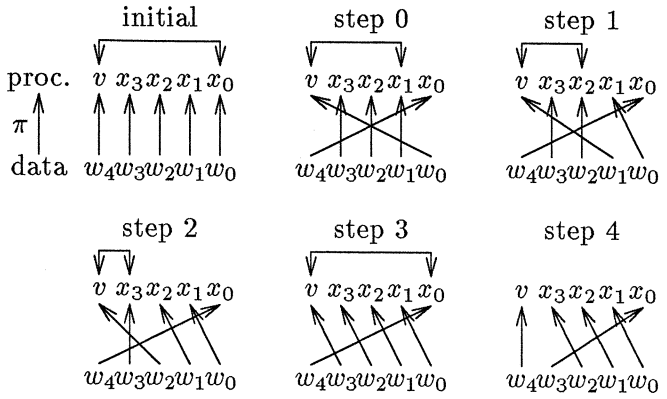

Figure 3: The changes of data allocations along time.

for each active link. This algorithm is labeled Algorithm A1a and its complexity shown in Table 2. Note that the complexity of Algorithms A1 (A1a) on a cube with unidirectional links is the same as that with bidirectional links.

Algorithm A2–SH through successive element exchanges at real distance 1. Algorithm A2 (in the *one-port* case) is due to Swarztrauber [16]. The algorithm requires $n+1$ exchange steps for a shuffle of real order n. 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. If each processor is viewed as consisting of two virtual processors, then the *virtual* processor dimension is exchanged with some *real* processor dimension. The following sequence of exchange operations realizes a shuffle permutation, where the virtual processor dimension (denoted by v) and the processor dimension subject to exchange are underlined.

$$(\underline{v}|x_{n-1}x_{n-2}\ldots x_1\underline{x_0}) \rightarrow (\underline{x_0}|x_{n-1}x_{n-2}\ldots x_2\underline{x_1}v) \rightarrow (\underline{x_1}|x_{n-1}x_{n-2}\ldots \underline{x_2}x_0v) \rightarrow \cdots$$

$$\rightarrow (\underline{x_{n-2}}|\underline{x_{n-1}}x_{n-3}\ldots x_1x_0v) \rightarrow (\underline{x_{n-1}}|x_{n-2}x_{n-3}\ldots x_1x_0\underline{v}) \rightarrow (v|x_{n-2}x_{n-3}\ldots x_1x_0x_{n-1}).$$

Figure 3 shows the changes of data allocations along time. The communication complexity is included in Table 2.

4.2.2 n-port communication

Algorithm A1a With *n-port communication*, the data set is divided into $2n$ equal parts, and each part is sent along a unique path. Path i, $0 \le i < 2n$, is defined as a sequence of dimensions:

$$\text{path } i = \begin{cases} (i, (i-1) \bmod n, (i-2) \bmod n, \ldots, (i+2) \bmod n, (i+1) \bmod n), & \text{if } 0 \le i < n, \\ (i', (i'+1) \bmod n, (i'+2) \bmod n, \ldots, (i'-2) \bmod n, (i'-1) \bmod n), & \\ \quad \text{with } i' = i-n, & \text{if } n \le i < 2n. \end{cases}$$

Path $n-1$ is identical to the path in the *one-port communication* case. For example, for $n=5$, the 10 paths are:

path 0: (0, 4, 3, 2, 1).
path 1: (1, 0, 4, 3, 2).
path 2: (2, 1, 0, 4, 3).
path 3: (3, 2, 1, 0, 4).
path 4: (4, 3, 2, 1, 0).
path 5: (0, 1, 2, 3, 4).
path 6: (1, 2, 3, 4, 0).
path 7: (2, 3, 4, 0, 1).
path 8: (3, 4, 0, 1, 2).
path 9: (4, 0, 1, 2, 3).

During the intermediate cycles, each dimension j of the first (last) n paths represents a send (receive) from node x along dimension j, if $x_{j+1} = x_j \neq x_{j-1}$, a receive (send) if $x_{j+1} \neq x_j = x_{j-1}$, and idle otherwise. Note that at most two of the $2n$ paths traverse the same link during any cycle. Furthermore, except for the first and last cycles, the two paths that traverse a link during a cycle do so in opposite directions [4]. During the first and last cycles, it is also the case that at most two paths traverse the same link, but they traverse the link in the same direction, and the messages are combined. Therefore, each link carries at most $\frac{K}{n}$ elements during any cycle. The complexity of the algorithm is shown in Table 2. For $n = 2$, the complexity of an algorithm using only the first two paths is the same as the algorithm using all four paths.

Another variation of Algorithm A0 is to pipeline the communications along one path only, say path $n-1$. The data set from each source node is divided into several packets. It can be shown that if one packet is sent out from the source node every other cycle then the routing can be arranged to be conflict free. The resulting complexity with optimum packet size, $(\sqrt{2Kt_c} + \sqrt{(n-2)\tau})^2$, is higher than Algorithm A1a, in general.

Algorithm A2 We define n exchange sequences where sequence i, $0 \leq i < n$, is

$$(\underline{v}|x_{n-1}x_{n-2}\ldots x_{i+2}x_{i+1}\underline{x_i}x_{i-1}\ldots x_0) \rightarrow (\underline{x_i}|x_{n-1}x_{n-2}\ldots x_{i+2}\underline{x_{i+1}}vx_{i-1}\ldots x_0)$$
$$\rightarrow (\underline{x_{i+1}}|x_{n-1}x_{n-2}\ldots \underline{x_{i+2}}x_ivx_{i-1}\ldots x_0) \rightarrow \cdots \rightarrow (\underline{x_{i-2}}|x_{n-2}x_{n-3}\ldots x_iv\underline{x_{i-1}}x_{i-3}\ldots x_0x_{n-1})$$
$$\rightarrow (\underline{x_{i-1}}|x_{n-2}x_{n-3}\ldots x_i\underline{v}x_{i-2}\ldots x_0x_{n-1}) \rightarrow (v|x_{n-2}x_{n-3}\ldots x_ix_{i-1}x_{i-2}\ldots x_0x_{n-1}).$$

Data is partitioned into n equal subsets. Each exchange sequence carries out the shuffle permutation for a distinct subset of $\frac{K}{n}$ elements. During any routing cycle, different sequences use edges in different dimensions.

4.3 Full-cube, real, generalized shuffle algorithms

For a *real* GSH of oder n on an n-cube, one can modify any algorithm for a *real* shuffle permutation on an n-cube by considering dimensions $\alpha_0, \alpha_1, \ldots, \alpha_{n-1}$, instead of 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.

4.4 Full-cube, separable dimension permutation algorithms

4.4.1 Using generalized shuffle permutation algorithms

Lemma 5 *With one-port communication, a full-cube, separable SDP of real order σ_p on an n-cube can be performed in a time of at most*

$$T^*_{dp}(1, \sigma_p, n, K) \leq \sum_{i=0}^{\beta-1} T^*_{gsh}(1, \sigma_i, n, K).$$

The SDP is simply obtained by performing β GSH's in sequence. The ith GSH consists of $2^{n-\sigma_i}$ independent permutations in subcubes of dimension σ_i, concurrently. If the algorithms chosen for GSH's are optimum, then the resulting algorithm for the SDP remains optimum. The complexities by utilizing Algorithms A1a and A2 are shown in Table 2. For a maximum $\beta = \frac{\sigma_p}{2}$, the data transfer time of Algorithm A2 is a factor of $\frac{3}{4}$ of that of Algorithm A1a, while the start-up time of Algorithm A2 is a factor of $\frac{3}{2}$ of that of Algorithm A1a. In fact, when $\sigma_i = 2$, $0 \leq i < \beta = \frac{\sigma_p}{2}$, Algorithm A1a (and A1) is essentially equivalent to the Single Path Transpose Algorithm without pipelining described in [9] for transposing a matrix with two-dimensional partitioning, if the dimensions are relabeled in a proper way.

With *n-port communication*, σ_i ports can be used concurrently for each GSH, and all GSH's can be performed concurrently.

Lemma 6 *With n-port communication, a full-cube, separable SDP of real order σ_p on an n-cube can be performed in a time of at most*

$$T^*_{dp}(\max_{i=0}^{\beta-1}\{\sigma_i\}, \sigma_p, n, K) \leq \sum_{i=0}^{\beta-1} T^*_{gsh}(\sigma_i, \sigma_i, n, K)$$

by exploiting concurrency within each GSH, and in time

$$T^*_{dp}(\sigma_p, \sigma_p, n, K) \leq \beta \times \max_{i=0}^{\beta-1}\{T^*_{gsh}(\sigma_i, \sigma_i, \sigma_i, \frac{K}{\beta})\}$$

by exploiting the concurrency fully.

These results are independent of the algorithm chosen for the GSH. For k-shuffle permutations of order 2 for each cycle, exploiting the concurrency within the GSH as well as between different GSH's based on Algorithm A1a for each GSH, the algorithm degenerates to the Double Path Transpose Algorithm (DPT) [9] without pipelining. Note that by exploiting the concurrency within the GSH as well as between different GSH's, the data transfer time can be optimal, if the algorithm chosen for each GSH is optimal with appropriate scheduling. However, the number of start-ups is equal to the product of the number of GSH permutations (β) defining the permutation and the number of start-ups required for the GSH of maximum real order, for instance $\beta \times \max_i\{\sigma_i + 1\}$ for Algorithm A2, which is not optimum, in general. For instance, if one cycle is of real order $\frac{n}{2}$, and other cycles are of real order 2, then the number of start-ups can be as large as $O(n^2)$. Algorithms A5 and A6 both reduce the complexity compared to naively exploiting the concurrency of different GSH permutations. They both have data transfer times and start-up times optimum within constant factors.

4.4.2 Algorithms not using generalized shuffle permutations

Algorithm A3–SDP through all-to-all personalized communication. If the data volume $K \geq O(N)$, then an arbitrary permutation can be performed as two successive *all-to-all personalized communications* [15]. In *all-to-all personalized communication* [10] every processor has a unique piece of data for every other processor. Consider a pair of nodes (i, j) (source, destination) for the permutation. During the first phase of the all-to-all personalized communication, data from node i to node j is distributed (scattered) to *all* nodes. During the second phase, the data is collected (gathered) into node j.

With *one-port communication*, an optimum routing algorithm (within a factor of two) is described in [13] and [10]. With *n-port communication*, the optimum algorithms (within a factor of two) are due to [10] and [15], independently. The resulting complexity of A3 is in Table 2. Clearly, Algorithm A3 is inferior to Algorithms A1a and A2, but may be preferable for non-separable SDP [4].

Algorithm A4–SDP through recursive dimension exchanges. An SDP can also be implemented by exchanging dimensions between subsets of dimensions recursively. The number of sets doubles for each recursion step, and the cardinality of the sets is halved. The number of steps is $\lceil \log_2 \sigma_p \rceil$ for a GSH, and $\max_i\{\lceil \log_2 \sigma_i \rceil\}$ for an SDP. Each step is of the same complexity as that of matrix transposition with two dimensional partitioning [9]. With *one-port communication*, the Single Path recursive Transpose (SPT) algorithm without pipelining [9] can be used. The complexity of the SPT algorithm is

$$\sigma_p K t_c + \sigma_p \left\lceil \frac{K}{B} \right\rceil \tau, \qquad \text{or} \quad \frac{3\sigma_p K}{4} t_c + \frac{3\sigma_p}{2} \left\lceil \frac{K}{2B} \right\rceil \tau.$$

The latter is based on the fact that in a 2-cube, node (01) and node (10) can exchange K elements in a time of $\frac{3K}{2} t_c + 3\tau$ by employing one virtual dimension, i.e., a special case of Algorithm A2. With *n-port communication*, the best known algorithm for matrix transposition with two-dimensional partitioning is described in [2], [9] and [15]. The resulting complexity of A4 is in Table 2. Algorithm A4 is in general an order of $\log \sigma_p$ higher than the lower bound. But for a special case where $\sigma_i = 2$ for all i's, then the SDP degenerates to a matrix transposition and this algorithm should be used.

4.4.3 Combining generalized shuffle permutations

Algorithm A5–Combining GSH's Algorithm A5 is based on distinct sequences of dimension exchange operations performed concurrently such that each processor dimension is used by at most two such sequences. The sequences are dimension rotations of each other, or address shuffles, except for the last dimension in the sequence. The permutation for each sequence is performed by Algorithm A2. It requires $\sigma_p + \beta$ routing cycles. One processor dimension per GSH is used twice, once for initiating the GSH, and once for leaving the cycle. For *n-port communication* the data set K can be divided into $\sigma_p + \beta + \min_i\{\sigma_i\} - 1$ parts such that each part runs through a different sequence of exchange operations.

$$\text{Seq}_0 \quad = \quad \alpha_{00}, \alpha_{01}, \ldots, \alpha_{0(\sigma_0-1)}, \alpha_{00}, \alpha_{10}, \alpha_{11}, \ldots, \alpha_{1(\sigma_1-1)}, \alpha_{10},$$

Cycle	0	1	2	3	4	5	6	7	8	9	10	11
Seq_0	0	1	2	3	0	4	5	6	4	7	8	7
Seq_1	1	2	3	0	4	5	6	4	7	8	7	1
Seq_2	2	3	0	4	5	6	4	7	8	7	1	2
Seq_3	3	0	4	5	6	4	7	8	7	1	2	3
Seq_4	0	4	5	6	4	7	8	7	1	2	3	0
Seq_5	4	5	6	4	7	8	7	1	2	3	0	1
Seq_6	5	6	4	7	8	7	1	2	3	0	1	5
Seq_7	6	4	7	8	7	1	2	3	0	1	5	6
Seq_8	4	7	8	7	1	2	3	0	1	5	6	4
Seq_9	7	8	7	1	2	3	0	1	5	6	4	5
Seq_{10}	8	7	1	2	3	0	1	5	6	4	5	8
Seq_{11}	7	1	2	3	0	1	5	6	4	5	8	7
Seq_{12}	1	2	3	0	1	5	6	4	5	8	7	8

Table 1: Combining different GSH's by Algorithm A2 with *n-port communication.*

$$\begin{aligned} & \cdots, \alpha_{(\beta-1)0}\alpha_{(\beta-1)1}, \cdots, \alpha_{(\beta-1)(\sigma_{\beta-1}-1)}, \alpha_{(\beta-1)0}. \\ \text{LetSeq}_k &= d_0, d_1, \ldots, d_{\sigma_p+\beta-1}, \\ \text{thenSeq}_{k+1} &= d_1, d_2, \ldots, d_{\sigma_p+\beta-1}, \delta(d_0), \qquad \forall 0 \le k < \sigma_p + \beta + \min_i\{\sigma_i\} - 2. \end{aligned}$$

Sequence 0 is constructed by including the first dimension for each GSH twice by making it also become the last dimension subject to an exchange operation for that GSH, assuming Algorithm A2 is used, and concatenating the dimension sequences for each GSH. Sequence $k+1$ is defined in terms of sequence k by a left cyclic shift excluding the first entry and appending the next dimension of the previous first entry in the same index set. Table 1 shows an example of 13 sequences of processor dimensions, which are used to perform the SDP: $\mathcal{J}_0 = \{0,1,2,3\}$, $\mathcal{J}_1 = \{4,5,6\}$, and $\mathcal{J}_2 = \{7,8\}$. Row i applies to GSH i, and column j to cycle j. The table gives the processor dimension subject to communication.

Lemma 7 [4] *The GSH's defined by the sequences* $\text{Seq}_i, 0 \le i < \sigma_p + \beta + \min_i\{\sigma_i\} - 1$ *has the property that any dimension is used by at most two GSH's during any routing cycle.*

Each sequence performs the SDP for $\frac{K}{n+\beta+\min_i\{\sigma_i\}-1}$ elements. The complexity is given in Table 2.

Algorithm A6–Hierarchical partitioning of the data set. This algorithm has a lower data transfer time than Algorithm A5, but the number of start-ups are in the range $[\sigma_p + \beta, 2(\sigma_p + \beta)]$. The strategy for Algorithm A6 is to group the GSH's out of which the SDP is composed into sets such that all sets require approximately the same time in realizing the SDP defined by the GSH's in the group. Within each group, the strategy is applied recursively. For

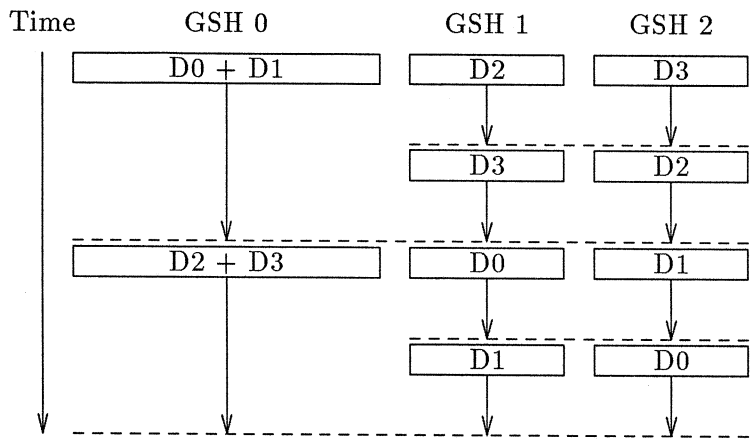

Figure 4: Hierarchical partitioning of data set for Algorithm A6.

instance, if the SDP consists of three GSH's of real order 7, 3, and 3, then the data set is partitioned into two parts of $\frac{K}{2}$ elements each. One part is permuted through the GSH of real order 7. The other part is partioned further into $\frac{K}{4}$ elements each, one for each of the GSH of real order 3. All three GSH permutations are performed concurrently, and for each GSH the data set is further subdivided for maximum concurrency within the GSH. By using the *n-port* version of Algorithm A2 each part of $\frac{K}{2}$ elements requires the same number of start-ups. When the permutation for all sets is complete, the data is repartitioned for a new set of permutations by applying the permutation defined by the next set of GSH's, modulo the number of sets, to the data. Hence, the data permuted for the GSH of order 7 is then permuted according to the two GSH's or order 3 each, and conversely the data already permuted according to these two GSH's will be permuted according to the GSH of order 7.

Figure 4 illustrates the data movement for this example, where D0 to D3 are the four equal partitions of the data set. One difference between Algorithms A5 and A6 is that in Algorithm A5 the data set is partitioned into disjoint subsets of the same size, and data in the same subset is carried through the same dimension during the same routing cycle. In Algorithm A6, partitioning of the data depends on the group level and the order of each GSH. We choose the number of groups at each level to be two in the following. Hence, the data set is partitioned into two disjoint subsets at each level. A GSH at the jth level operates on a data subset of $\frac{K}{2^j}$ elements. There are 2^j invokations of Algorithm A2.

The goal of the grouping of GSH's is to minimize the product of the number of sets of GSH's and the maximum time for any set of GSH's. Since the data transfer time is almost independent of the real order of a GSH for *n-port communication* using Algorithms A1a or A2, we focus on minimizing the total number of start-ups. The lemma below gives a bottom-up algorithm for binary grouping of the GSH's and the data set. The total number of start-ups is less than $2(\sigma_p + \beta)$.

Lemma 8 [4] *Let* $\mathcal{S} = \{s_0, s_1, \ldots, s_{\beta-1}\}$ *be a multi-set of positive integers, i.e., may contain*

the same integer more than once. Define $f(S)$ as follows:

$$f(S) = \begin{cases} i, & \text{if } S = \{i\}, \\ \min_{\forall k \text{ partitioning of } S}\{k \times \max_i f(S_i), \text{ where } S = \cup_{i=\{0,1,\ldots,k-1\}} S_i\}, & \text{otherwise.} \end{cases}$$

Then, $\sum_{i=0}^{\beta-1} s_i \leq f(S) < 2\sum_{i=0}^{\beta-1} s_i$.

At the bottom level of the partitioning, there is one GSH per partition. Any GSH algorithm can be employed for the set of GSH's in a partition. We only consider Algorithm A2, since it may be preferable to Algorithm A5 depending on the values of K, t_c and τ, etc. If Algorithm A1a is used, then the resulting algorithm has a higher complexity than Algorithm A5. Any SDP algorithm can be employed at any level.

Theorem 2 *Let* $S = \{\sigma_0+1, \sigma_1+1, \ldots, \sigma_{\beta-1}+1\}$ *and* $\min(S) = \min\{\sigma_0+1, \sigma_1+1, \ldots, \sigma_{\beta-1}+1\}$. *Then, there exists an algorithm which realizes the SDP with the number of start-ups* $\leq f(S)$ *and the data transfer time* $\leq \frac{\min(S)K}{2(\min(S)-1)}t_c$.

Proof: We show the theorem by induction on the level of the tree inherited from the construction of $f(S)$ starting from the last level. Each set S specify an SDP by adding one to the sequence of real orders in the index sets. Let $T(S,K)$ be the time required to realize the dimension permutation specified by the set S with data volume K by the hierarchical partitioning algorithm. By employing Algorithm A2, $T(\{\sigma_i+1\},K) = \frac{(\sigma_i+1)K}{2\sigma_i}t_c + (\sigma_i+1)\lceil\frac{K}{2\sigma_i B}\rceil\tau$. Assume $T(S_i,K) = \frac{\min(S_i)K}{2(\min(S_i)-1)}t_c + f(S_i)\tau$ with optimum packet size and S_i's, $0 \leq i < k$, are the k disjoint subsets of the set S that minimize $f(S)$. To realize the dimension permutation specified by the set S, first partition the data set of size K into k equal subsets, called subsets $0, 1, \ldots, k-1$. The data of subset i participates in a sequence of SDP_i, $\text{SDP}_{(i+1)\bmod k}, \ldots, \text{SDP}_{(i-1)\bmod k}$, where SDP_i is the SDP specified by the set S_i. Furthermore, the data of different subsets of the set K are permuted concurrently, one such subset for each partition of the set S. Since, all the k subsets are of the same size, the time to realize the SDP with optimum packet size is at most

$$\begin{aligned} T(S,K) &= k \times \max_i\{T(S_i, \frac{K}{k})\} \\ &= k \times \max_i\{\frac{\min(S_i)K}{2k(\min(S_i)-1)}t_c + f(S_i)\tau\} \\ &\leq \frac{\min(S)K}{2(\min(S)-1)}t_c + f(S)\tau. \end{aligned}$$

Note that with arbitrary packet size, the number of start-ups is

$$\max_i\{2^{l_i}(\sigma_i+1)\lceil\frac{K}{\sigma_i 2^{l_i+1}B}\rceil\}\tau,$$

where l_i is the level of σ_i in the binary tree defined in the proof of Lemma 8 [4]. ∎

The complexity is given in Table 2. The optimum packet size is $\frac{K}{2\sigma_p}$, if the SDP degenerates to a GSH. For $\beta \geq 2$, the optimum packet size can be shown to be bounded from above by $\frac{K}{8}$. The worst case occurs when $\beta = 2$ and $\sigma_0 = 2$. So, the maximum packet size is $\max(\frac{K}{8}, \frac{K}{2\sigma_p})$.

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. The first phase, is an expansion phase in which each processor with data partitions it into 2^{n-m_p} pieces and performs a *one-to-all personalized communication* for each of the 2^{m_p} subcubes of dimension $n - m_p$ concurrently. In the second phase, the best algorithm for a full-cube permutation is used, and is performed 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 [3,10] 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

For a separable SDP, the data transfer time of Algorithm A1a is a factor of $\frac{2\sigma_p}{\sigma_p+\beta}$ higher than that of Algorithm A2 for *one-port communication* and about a factor of $\frac{2\min_i\{\sigma_i\}}{\min_i\{\sigma_i\}+1}$ higher for *n-port communication*. The factor ranges from 1.33 to 2 for both *one-port* and *n-port communications*. The data transfer time of Algorithm A2 is a factor of 1 – 1.5 higher than the lower bound. With optimum packet size, the number of start-ups of Algorithm A2 is a factor of $1 + \frac{\beta}{\sigma_p}$ higher than Algorithm A1a for *one-port communication* and a factor of $1 + \frac{1}{\max_i\{\sigma_i\}}$ higher for *n-port communication*. 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 A1a is relatively more competitive for *n-port communication* than for *one-port communication* with optimum packet size. The break-even point between Algorithms A1a and A2 is $\tau = \frac{\sigma_p-\beta}{2\beta} K t_c$ for *one-port communication*, and $\tau = (1 - \frac{1}{\max_i\{\sigma_i\}})\frac{K t_c}{2\beta}$ for *n-port communication* and $\tau > \frac{K t_c}{12}$.

With *n-port communication*, if the start-up time is not negligible and not all cycles are of the same real order, then Algorithms A5 or A6 should be used, which are optimum within constant factors. Algorithms A5 and A6 are competitive. The break-even point between them is approximately $\tau = \frac{K t_c}{6\sigma_p - 4}$ for the worst case of both, i.e., $\beta = \frac{\sigma_p}{2}$ and $\min_i\{\sigma_i\} = 2$. Algorithm A3 is always inferior to Algorithm A2, but may be preferable in the non-separable SDP case

Comm.	Alg.	B_{opt}	Communication complexity	t_c factor/$l\,b$	τ factor/$l\,b$
sep. SDP *one-port* comm.	A0	K	$2(\sigma_p-\beta)Kt_c+2(\sigma_p-\beta)\lceil\frac{K}{B}\rceil\tau$	[2, 4)	2
	A1	$2K$	$2(\sigma_p-\beta)Kt_c+((\sigma_p-2\beta)\lceil\frac{2K}{B}\rceil+2\beta\lceil\frac{K}{B}\rceil)\tau$	[2, 4)	1
	A1a	K	$\sigma_pKt_c+\sigma_p\lceil\frac{K}{B}\rceil\tau$	2	1
	A2	$\frac{K}{2}$	$\frac{(\sigma_p+\beta)}{2}Kt_c+(\sigma_p+\beta)\lceil\frac{K}{2B}\rceil\tau$	(1, 1.5]	(1, 1.5]
	A3	$\frac{K}{2}$	$nKt_c+2n\lceil\frac{K}{2B}\rceil\tau$	$\frac{2n}{\sigma_p}$	$\frac{2n}{\sigma_p}$
	A4	K	$\lceil\log_2\sigma_{max}\rceil(\sigma_pKt_c+\sigma_p\lceil\frac{K}{B}\rceil\tau)$	$[2, 2\lceil\log_2\sigma_{max}\rceil]$	$[1, \lceil\log_2\sigma_{max}\rceil]$
sep. SDP *n-port* comm.	A1a	$\frac{K}{\sigma_{max}\beta}$	$Kt_c+\beta\sigma_{max}\lceil\frac{K}{\sigma_{max}\beta B}\rceil\tau$	2	$[1, \frac{(\sigma_p+2)^2}{8}]$
	A2	$\frac{K}{2\sigma_{min}\beta}$	$\beta\max_i\{\frac{(\sigma_i+1)K}{2\sigma_i\beta}t_c+(\sigma_i+1)\lceil\frac{K}{2\sigma_i\beta B}\rceil\tau\}$	(1, 1.5]	$(1, \frac{(\sigma_p+3)^2}{8}]$
	A3	$\frac{K}{2n}$	$Kt_c+2n\lceil\frac{K}{2nB}\rceil\tau$	2	$\frac{2n}{\sigma_p}$
	A4	$\sqrt{\frac{K\tau}{2t_c}}$	$\lceil\log_2\sigma_{max}\rceil(\frac{K}{2B}+1)(\tau+Bt_c)$	$(1, 2\lceil\log_2\sigma_{max}\rceil]$	
	A5	$\frac{K}{\sigma_p+\beta+\sigma_{min}-1}$	$\frac{(\sigma_p+\beta)Kt_c}{\sigma_p+\beta+\sigma_{min}-1}+\lceil\frac{K}{(\sigma_p+\beta+\sigma_{min}-1)B}\rceil(\sigma_p+\beta)\tau$	(1, 2)	(1, 1.5]
	A6	$\max(\frac{K}{2\sigma_p}, \frac{K}{8})$	$(\frac{1}{2}+\frac{1}{2\sigma_{min}})Kt_c+(2\sigma_p+2\beta-1)\tau$	(1, 1.5]	(1, 3)

Table 2: Summary of the communication complexities for various algorithms. Note that $\sigma_i=\sigma_{p_i}$, $\sigma_{max}=\max_i\{\sigma_i\}$ and $\sigma_{min}=\min_i\{\sigma_i\}$. The complexity for Algorithm A6 is with $B=B_{opt}$. The complexity with arbitrary B can be derived from the recursion in the proof of Theorem 2.

[4]. The complexity of Algorithm A4 is, in general, an order of $O(\log_2\max_i\{\sigma_i\})$ higher than the lower bound, but has the lowest complexity of the considered algorithms for $\sigma_i\leq 2$, $\forall 0\leq i<\beta$, and *n-port communication*. Table 2 summarizes the complexities of different algorithms. The last two columns are the ratio of data transfer time (start-up time) to that of the lower bound. For convenience, we scale the minimum number of start-ups to σ_p instead of σ_p-oJ, Theorem 1. The complexity of Algorithm A4 (*n-port*) is $\max_i\{\lceil\log_2\sigma_i\rceil\}(\sqrt{\frac{Kt_c}{2}}+\sqrt{\tau})^2$ with B_{opt} and $\sigma_p\leq\sqrt{\frac{Kt_c}{2\tau}}$ [9]; the factor is by comparing to $\frac{K}{2}t_c+\sigma_p$. Most algorithms listed are optimum within constant factors for optimally chosen packet sizes, except if $\sigma_p<O(n)$ for Algorithm A3, and $\max_i\{\log_2\sigma_i\}>O(1)$ for Algorithm A4. Also, Algorithms A1a and A2 (in general) are not optimum for *n-port communication*, and optimally chosen packet sizes. However, Algorithms A1a and A2 are comparable with Algorithms A5 and A6 for a packet size that is very small compared to the data volume and *n-port communication*.

5 Implementation issues

We have implemented Algorithms A1, A2, and A3 (*one-port* version) on the Intel iPSC. For comparison we also implemented shuffle permutations using the routing logic. Figure 5 shows the measured times as a function of message length on a 5-cube. For 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. Algorithm A3 has the disadvantage that the data to be exchanged between pairs of neighbors may not be in contiguous memory locations, and therefore either more start-ups, or local data movement be required. Such movement requires a substantial amount of time on the Intel iPSC [9]. Figures 6 and 7 show the measured times as a function of cube dimensions

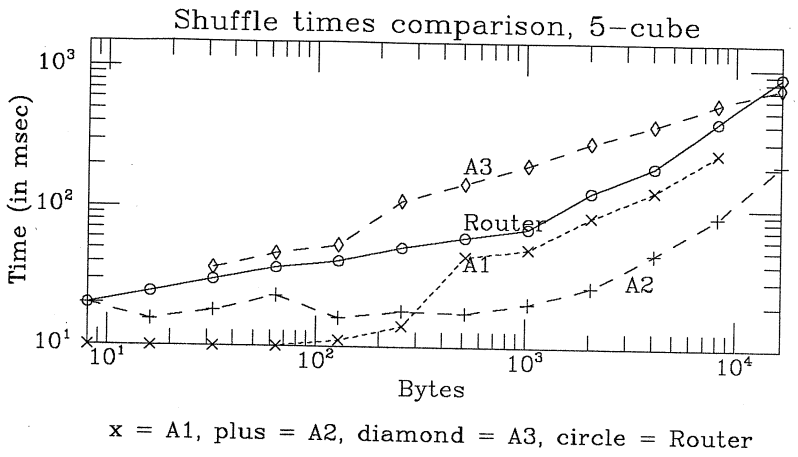

Figure 5: The measured shuffle times as a function of message lengths on an iPSC 5-cube.

with message size being 8 bytes and 1 *k*bytes, respectively. Figure 8 shows the measured times of matrix transposition on a 6-cube using Algorithms A1, A2, A3 and the iPSC router. For a matrix transposition on a 6-cube, the predicted data transfer times compare as 1 to $\frac{3}{4}$, and the predicted start-up times compare as 1 to $\frac{3}{2}$ for Algorithm A1 to Algorithm A2. The measured communication times of Algorithm A2 is higher than that of Algorithm A1 by a factor of 2 – 3. The difference between the predicted and measured behavior is due to the fact that the predictions are based on bidirectional concurrent communication; Algorithm A1 needs only one-way communication, but Algorithm A2 requires two-way communication. On the Intel iPSC the start-up time for a send *and* a receive is almost twice that of a send *or* a receive.

6 Summary and conclusions

We have devised a few algorithms provably optimum within small constant factors for stable dimension permutations on Boolean cubes. Algorithm A1a is comparable with Algorithm A2 [16], if communication is restricted to one port at a time. Depending on the machine parameters and data volume one or the other may have the lowest complexity with the ratio of communication times $\frac{\text{Algorithm A1a}}{\text{Algorithm A2}}$ varying in the range $[\frac{2}{3}, 2]$. When concurrent communication on all ports of each node is possible, Algorithms A5 and A6 are comparable. Implementations on the Intel iPSC showed that the measured communication time of Algorithm A1 is the lowest of Algorithms A1, A2, A3, and permutation by direct use of the router for shuffle permutations with a message size of up to a few hundred bytes. For the transposition of a two-dimensionally partitioned matrix and bit-reversal permutation of any message sizes the same result holds. For matrix transposition Algorithm A1 degenerates to Algorithm A4 in the *one-port communication*case. For bit-reversal permutations Algorithm A1 requires about half to one third the time of Algorithm A2.

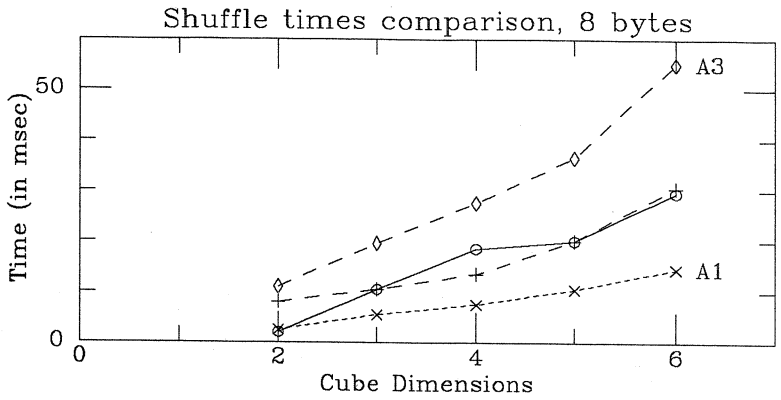

Figure 6: The measured shuffle times as a function of cube dimensions on an iPSC 6-cube with $K = 8$ bytes.

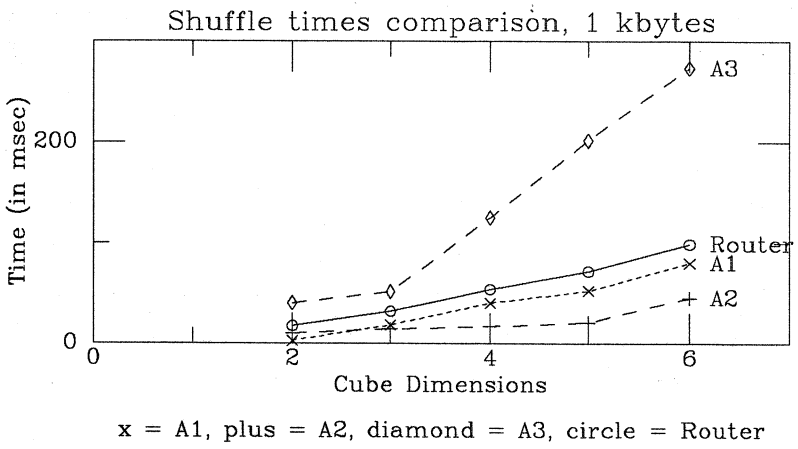

Figure 7: The measured shuffle times as a function of cube dimensions on an iPSC 6-cube with $K = 1$ $k$bytes.

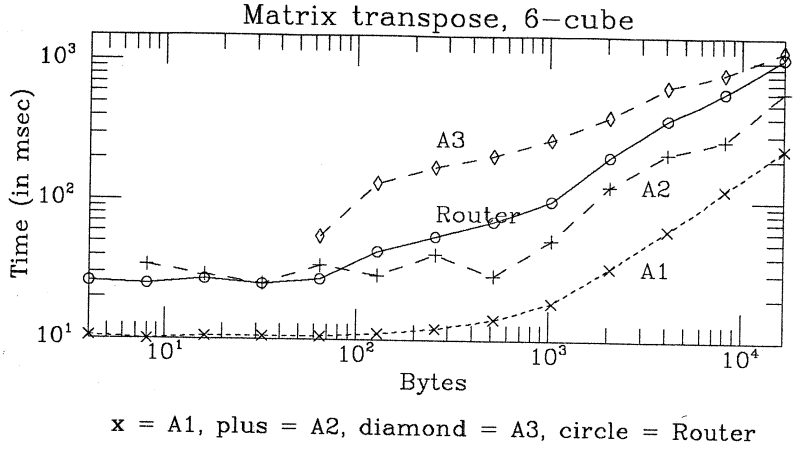

Figure 8: The measured matrix transpose times as a function of message lengths on an iPSC 6-cube.

Acknowledgement

The work has been supported by the Office of Naval Research under contracts N00014-86-K-0564.

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. Algorithms for matrix transposition on Boolean n-cube configured ensemble architectures. In *1987 International Conf. on Parallel Processing*, pages 621–629, IEEE Computer Society, 1987. Report YALEU/DCS/RR-577, April 1987.

[3] 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.

[4] Ching-Tien Ho and S. Lennart Johnsson. *Stable dimension permutation on Boolean cubes.* Technical Report YALEU/DCS/RR-617, Dept. of Computer Science, Yale Univ., New Haven, CT, March 1988.

[5] Ching-Tien Ho and S. Lennart Johnsson. *Unstable Dimension Permutations on Boolean Cubes.* Technical Report YALEU/DCS/RR-, Dept. of Computer Science, Yale Univ., New Haven, CT, 1988. in preparation.

[6] 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).

[7] 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.

[8] 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.

[9] S. Lennart Johnsson and Ching-Tien Ho. Matrix transposition on Boolean n-cube configured ensemble architectures. *SIAM J. on Algebraic and Discrete Methods.* To appear. YALE/DCS/RR-572, September 1987. (Revised edition of YALEU/DCS/RR-494 November 1986.).

[10] 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. To appear in IEEE Trans. Computers.

[11] 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*, Society of Photo-Optical Instrumentation Engineers, 1987. Report YALEU/DCS/RR-598, October 1987.

[12] S. Lennart Johnsson and Faisal Saied. *Convergence of substructuring in Poisson's equation.* Technical Report , Dept. of Computer Science, Yale Univ., New Haven, CT, In preparation 1987.

[13] 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.

[14] 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.

[15] 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.

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