Yale University
Department of Computer Science

Stable Dimension Permutations on Boolean Cubes

Ching-Tien Ho and S. Lennart Johnsson

YALEU/DCS/TR-617
October 1988

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

† A preliminary version of this paper was presented at the Third Conference on Hypercube Concurrent Computers and Applications, January, 1988, Pasadena, CA. A revised edition of TR620.

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 lower bounds and algorithms optimal within a small constant factor for *stable dimension permutations* on Boolean cubes. A *stable dimension permutation* is a permutation where a global address $(a_{q-1}a_{q-2}\ldots a_0)$ receives its new content from a global address $(a_{\delta(q-1)}a_{\delta(q-2)}\ldots a_{\delta(0)})$, where δ is a permutation function on $\{0,1,\ldots,q-1\}$. With communication restricted to one port at a time for each processor, the lower bound has a term proportional to the number of processor dimensions being part of the dimension permutation for the number of communications in sequence, and a term for the data transfer time that is proportional to the same number of dimensions *and* the size of the data set per processor. With concurrent communication on all ports of every processor, the bound for the data transfer time is reduced to become proportional only to the size of the data set per processor. We also show that for an optimal algorithm the time for a dimension permutation cannot be reduced by using the cube dimensions not being part of the dimension permutation, if data is allocated to the entire cube. However, if data is only allocated to a subcube, then the dimensions not being part of the subcube can be used to reduce the time complexity of the dimension permutation. 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*. 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.

1 Introduction

A dimension permutation is a permutation defined on the bits of the address field, while an arbitrary permutation is a permutation on the address field. There are $(\log_2 M)!$ possible dimension permutations compared to $M!$ arbitrary permutations for an address space of size M. Examples of stable dimension permutations are k-shuffle/unshuffle permutations, matrix transposition [5], [8], bit-reversal [11], and conversion between various data structures, such as consecutive and cyclic storage [5], [8]. Shuffle operations can be used to reconfigure a two dimensional partitioning to a three dimensional partitioning of a matrix for multiplication with maximum concurrency [7]. They may also be used for data (re)alignment for certain Fast Fourier Transform algorithms [6], [11].

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

The main focus of this paper is on lower bounds for *stable* dimension permutations for communication restricted to one port at a time for each processor, *one-port communication*, and concurrent communication on all ports of every processor, *n-port communication*, and several optimal algorithms for *n-port communication*. (Algorithms for *one-port communication* are trivially derived from algorithms for GSH with *one-port communication*.) The presentation of the algorithms also illustrates a general methodology for devising communication algorithms for *n-port communication*. A *stable dimension permutation* is a permutation within (sub)cubes such that every node holds data both before and after the permutation. *Unstable* dimension permutations are treated in [12]. Stable generalized shuffle permutations, a subclass of dimension permutations have also been studied by Flanders [1] on mesh-connected array processors, and by Swarztrauber [15] on Boolean cubes. Flanders and Swarztrauber give almost identical algorithms for communication restricted to one port at a time for each processor, *one-port communication*. Lower bounds and optimal algorithms for concurrent communication on all ports of every processor, *n-port communication*, as well as some improved *one-port communication* algorithms are given in [9]. The notation and definitions used throughout the paper are introduced in Section 2. In Section 3, we discuss lower bounds. Stable dimension permutation algorithms are described in Section 4. The algorithms are labeled A1 - A6. Algorithm A1 is described in [9]. Algorithms A2 and A3 are based on algorithms of all-to-all personalized communication [10], [14] and matrix transposition with two-dimensional partitioning [8], [14], respectively. Algorithms A4 - A6 are new. We conclude with a few remarks in Section 5.

2 Preliminaries

2.1 Address spaces and Boolean cubes

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

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

The *distance* between two nodes x and y in a Boolean n-cube is $Hamming(x,y)$. The number of nodes at distance j from any node is $\binom{n}{j}$. The number of disjoint paths between any pair of nodes x and y is n. $Hamming(x,y)$ paths are of length $Hamming(x,y)$ and $n - Hamming(x,y)$ paths are of length $Hamming(x,y) + 2$ [13]. $||a||$ denotes the number of 1-bits in the binary representation of a, i.e., $||a|| = Hamming(a,0)$. $|S|$ is the cardinality of set S.

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

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

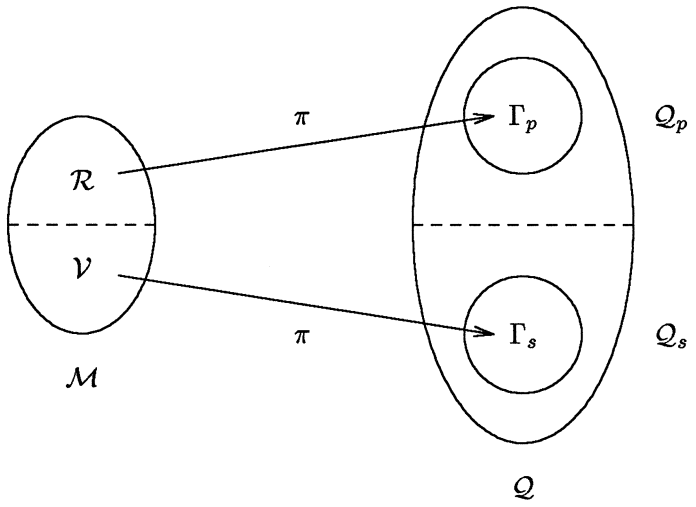

Figure 1: The relationship between the sets of address dimensions.

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 set of *logic dimensions* is $\mathcal{M} = \{0, 1, \ldots, m-1\}$. The relationship between the number of processor dimensions, the local storage dimensions, and the number of logic dimensions is arbitrary. For instance, if $m \leq q - n$ and $m \leq n$, then the entire data set can be allocated to the local storage of a single processor, or across processors with one element per processor using 2^m processors.

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

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

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

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

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

2.2 Classification of dimension permutations

A dimension permutation implies a change in allocation from $\pi^b(\mathcal{M})$, before the permutation, to $\pi^a(\mathcal{M})$, after it. Let $\mathcal{R}^b$ be the set of logic dimensions mapped to processor dimensions before the permutation. Let $\mathcal{R}^a$ be the set of logic dimensions mapped to processor dimensions after it. $\mathcal{V}^b$ and $\mathcal{V}^a$ are defined similarly. The sets of machine dimensions used before and after the permutation are denoted $\Gamma^b = \Gamma^b_p \cup \Gamma^b_s$ 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*. Note that we classify the case where $\Gamma^b_p = \Gamma^a_p$ and $\Gamma^b_s \neq \Gamma^a_s$ as an unstable dimension permutation. The restriction of *stable* dimension permutations to use the same local address space before and after the permutation is made for notational convenience. The algorithms for stable dimension permutations as defined here also work for the case $\Gamma^b_p = \Gamma^a_p$ and $\Gamma^b_s \neq \Gamma^a_s$ with the same complexity as in the stable case, if the time for local data rearrangement is ignored.

In the following we only consider *stable* dimension permutations. For convenience, let $\Gamma = \Gamma^b = \Gamma^a$, $\Gamma_p = \Gamma^b_p = \Gamma^a_p$ and $\Gamma_s = \Gamma^b_s = \Gamma^a_s$. Unstable permutations are treated in [4].

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

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

The order of the identity permutation is 0. A subscript σ on δ, δ_σ, is used to denote the order of the SDP being σ. The permutation function δ applies to the subset of machine dimensions. For convenience, we use $\delta(a)$, $a \in \mathcal{A}$, to denote $(a_{\delta(q-1)}a_{\delta(q-2)}\ldots a_{\delta(0)})$ where δ is extended to a function of $\mathcal{Q} \rightarrow \mathcal{Q}$ with $\delta(i) = i$, $i \in \mathcal{Q} - \mathcal{J}$.

Throughout the paper we define the SDP on the machine dimensions (δ), but an SDP can also be defined on the logic dimensions (δ'). In an SDP a logic dimension k assigned to machine dimension $i = \pi^b(k)$ is reassigned to machine dimension $\delta(i) = \pi^a(k)$. Clearly, $\delta = \pi^a \circ \pi^{-b}$ and is consistent with Definition 4. Following the definition, we have the corollary below.

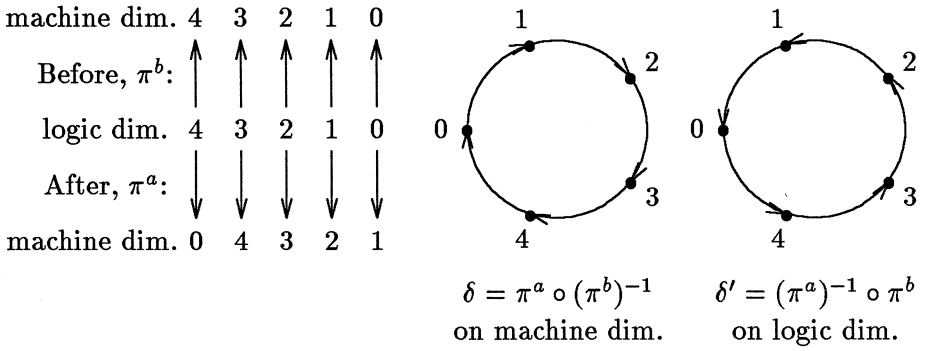

Figure 2: The definition of δ and δ', and cycles formed by traversing δ and δ'.

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

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

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

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

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

Definition 6 A *full-cube permutation* (FCP) is 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 *generalized shuffle permutation* (GSH) of order σ, gsh_σ, is an SDP such that $\delta_\sigma(\alpha_0) = \alpha_1, \delta_\sigma(\alpha_1) = \alpha_2, \ldots, \delta_\sigma(\alpha_{\sigma-1}) = \alpha_0$, $\alpha_i \neq \alpha_j$, $i \neq j$, $\alpha_i, \alpha_j \in \mathcal{J}$, $0 \leq i, j < \sigma$, and $\delta_\sigma(i) = i$, $\forall i \in \Gamma - \mathcal{J}$.

For convenience, we let $\mathcal{J} = \{\alpha_0, \alpha_1, \ldots, \alpha_{\sigma-1}\}$ be an ordered index set with the order implied. An SDP consists of a number of independent GSH's (or cycles). The number of independent GSH's 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 8 A *real* GSH is a GSH such that $\mathcal{J} \subseteq \Gamma_p$ and $\mathcal{J} \neq \phi$. A *virtual* GSH is a GSH such that $\mathcal{J} \subseteq \Gamma_s$ and $\mathcal{J} \neq \phi$. A GSH that is neither real nor virtual and $\mathcal{J} \neq \phi$ is a *mixed* GSH.

Definition 9 An SDP is *separable*, if it can be decomposed into *independent* GSH's (i.e., with their index sets disjoint) that are either real or virtual; it is *non-separable*, otherwise. A *real* SDP is an SDP consisting only of all real GSH's.

Definition 10 The *real order* σ_p of an SDP on a machine address space is $|\{j|j \neq \delta(j), j \in \Gamma_p\}| = |\mathcal{J} \cap \Gamma_p|$, and its *virtual order* σ_s is $|\{j|j \neq \delta(j), j \in \Gamma_s\}| = |\mathcal{J} \cap \Gamma_s|$.

For an SDP where $\mathcal{R}^b \cap \mathcal{R}^a = \phi$, the permutation is an *all-to-all personalized communication* [10,8]. Moreover, some SDP's 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, $\delta^2(i) = \delta(\delta(i)) = i$, $\forall i \in \Gamma$ [8].

3 The complexity of stable dimension permutations

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

The time complexity for the different 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 *sdp* for a stable dimension permutation. The superscript $*$ is either lb for a lower bound, or an algorithm identifier for an upper bound. The first argument for T_{sdp} 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 "allocated" processor, i.e., $K = 2^{m_s}$.

3.1 Some properties of stable dimension permutations

In [9] we show that for a GSH of order σ, $2^{m-\sigma}$ elements are subject to the same permutation. We also show that a GSH of real order σ_p consists of $2^{m_p-\sigma_p}$ permutations in subcubes of size 2^{σ_p}. Furthermore, we show the following lemma.

Lemma 2 *A GSH of real order $\sigma_p < m_p$ cannot be improved by communication in the $m_p - \sigma_p$ processor dimensions that are not included in the GSH, if the original algorithm fully utilizes the bandwidth.*

These lemmas also apply to stable dimension permutations. The arguments in the proofs still hold. 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 *real* and *mixed* GSH's in the following. 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 $\mathcal{J}_i \subset \Gamma_p$. Also, let σ_{p_i} be the *real order* of the GSH defined by $\mathcal{J}_i$. Clearly, $\delta_\sigma(i) = i, \forall i \in \Gamma - \mathcal{J}$, and $\sigma_p = \sum_{i=0}^{\beta-1} \sigma_{p_i} \leq \sum_{i=0}^{\beta-1} \sigma_i = \sigma$. We also let $\sigma_{max} = \max_i\{\sigma_{p_i}\}$ and $\sigma_{min} = \min_i\{\sigma_{p_i}\}$ for all $0 \leq i < \beta$.

Lemma 3 [9] *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.

Lemma 4 [9] *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.

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

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

for one-port communication, and

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

for n-port communication. (Recall that oJ is the number of sets such that σ_i is odd and $\mathcal{J}_i \subset \Gamma_p$.)

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 $\max\{Hamming_r(a, \delta(a))\}, \forall a \in \mathcal{A}$, which is obtained by maximizing the *real* distance for each index set $\mathcal{J}_i$. By Lemmas 3 and 4, the maximum real 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 SDP and at most σ_p for some SDP's (as a tight bound).*

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.

An extended cube permutation of real order σ_p can be improved by communication in the $n - m_p$ processor dimensions not used for the allocation of the data array. A possible algorithm is a composition of a subcube expansion, full cube permutation, and subcube compression algorithms. The permutation is then performed on a data set reduced by a factor of 2^{n-m_p}. The subcube expansion permutation is of type *one-to-all personalized communication* [3].

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*

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

for one-port communication, and

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

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 SDP 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 *separable* and *non-separable* stable dimension permutations (SDP's). 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. We consider *full-cube permutations* (FCP) and *extended-cube permutations* (ECP). Algorithms of the ECP will utilize algorithms for the FCP as primitives.

The SDP algorithms presented here are all based on the GSH algorithms in [9]. 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 executing the algorithms for the different GSH's making up the SDP one after the other, Corollary 3. In the *n-port communication* case an optimal algorithm for an SDP is obtained by using an optimal algorithm for 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, one for each GSH. The ith part participates in the GSH's specified by the sequence of index sets: $\mathcal{J}_i, \mathcal{J}_{(i+1) \bmod \beta}, \ldots, \mathcal{J}_{(i-1) \bmod \beta}$. For each partition the data is further subdivided into σ_{p_i} parts for maximum bandwidth utilization. Hence, in every step σ_p dimensions are used, which is optimum for full-cube permutations.

In most systems the start-up time cannot be ignored, and the orders of the GSH's of the SDP are, in general, not the same. Moreover, the GSH algorithms we present are of optimal order, but not strictly optimal. For a real GSH Algorithm A1 [9] based on a sequence of exchanges with a fixed virtual dimension requires one start-up and a data transfer time of $\frac{K}{2\sigma} t_c$ in excess of the lower bound. We use this algorithm as a base algorithm for the SDP algorithms we present for the *n-port communication* case. A discussion of the consequences of using Algorithm A1′ [9] in which the data transfer time is approximately doubled, but one less start-up required is given at the end of this section. By performing the SDP by Algorithm A1 for each GSH in sequence, β start-ups in excess of the lower bound are required, which is minimal using Algorithm A1, but the communication bandwidth of the Boolean cube is not used effectively. By applying the algorithm concurrently to all GSH's of the SDP the communication bandwidth is fully utilized, but the number of start-ups is $\beta(\sigma_{max} + 1)$. Algorithm A4 requires $\sigma_p + \beta$ start-ups and performs $\sigma_p + \beta + \sigma_{min} - 1$ permutations concurrently. Algorithms A5 and A6 apply two different strategies for reducing the number of start-ups compared to using Algorithm A1 for all GSH's of the SDP concurrently. In Algorithm A5 the value of σ_{max} is reduced by partitioning

a GSH into several GSH's that are performed concurrently followed by one GSH that couples the parts. In Algorithm A6 the GSH's are grouped into sets such that the number of start-ups is proportional to the number of sets, and the maximum number of start-ups required for any set. The grouping of GSH's is performed recursively.

4.2 Algorithms

4.2.1 The base algorithm.

By performing the dimension permutation as a sequence of exchange operations between a fixed dimension not part of the index set, say dimension v_{k-1}, $n+1$ exchange steps are required for a GSH of real order n [9]. By using a dimension used for local storage addresses as the virtual dimension, the communication in each cycle becomes *bidirectional*, i.e., exchange operations. During each step, all communications occur in the same dimension of the cube. Processors in subcube 0 exchange the second half of the data with the first half of the data of the processors in subcube 1. This is Algorithm A1 (A1′) [9].[2]

The sequence of exchange steps for a shuffle permutation can be illustrated as follows:

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

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

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

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

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

Note that all data are sent through some shortest path in A1′, since each dimension is only routed once (unlike A1). The total bandwidth required is the same as the lower bound. With *one-port communication*, the data transfer time nKt_c is exactly twice the lower bound.

The *n-port* version the data is partitioned into n equal sized subsets for a permutation of order n. Assume $\log_2 n = \eta$ is an integer, then exchange sequence i, $0 \leq i < n$, can be represented as

$$(v_{k-1}\dots v_{k-\eta}\underline{v_{k-\eta-1}}v_{k-\eta-2}\dots v_0|r_{n-1}r_{n-2}\dots r_{i+2}r_{i+1}\underline{r_i}r_{i-1}\dots r_0)$$

[2]It is called Algorithm A2 (A2′) in [9].

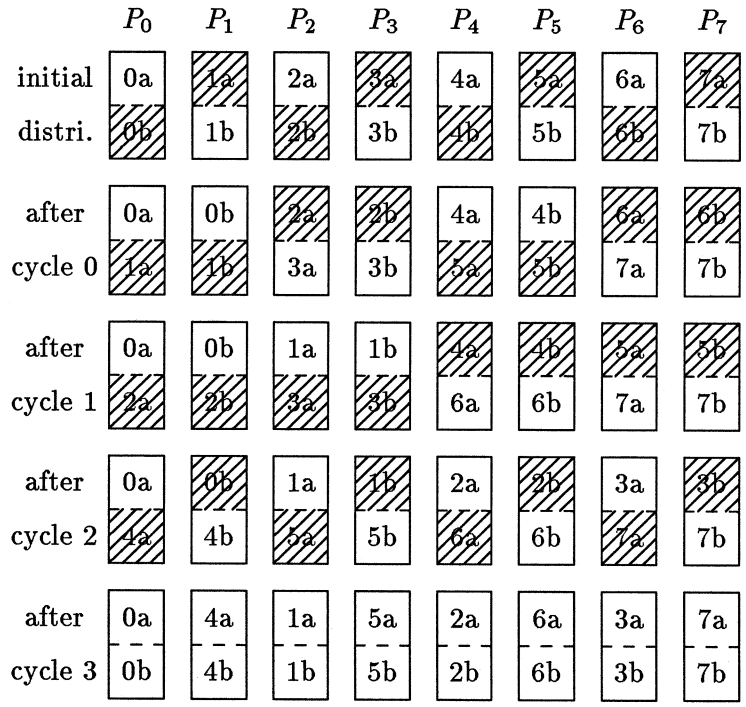

Figure 3: The 4 exchange steps in a 3-cube. The shaded areas are the parts of data subject to exchange during the next step.

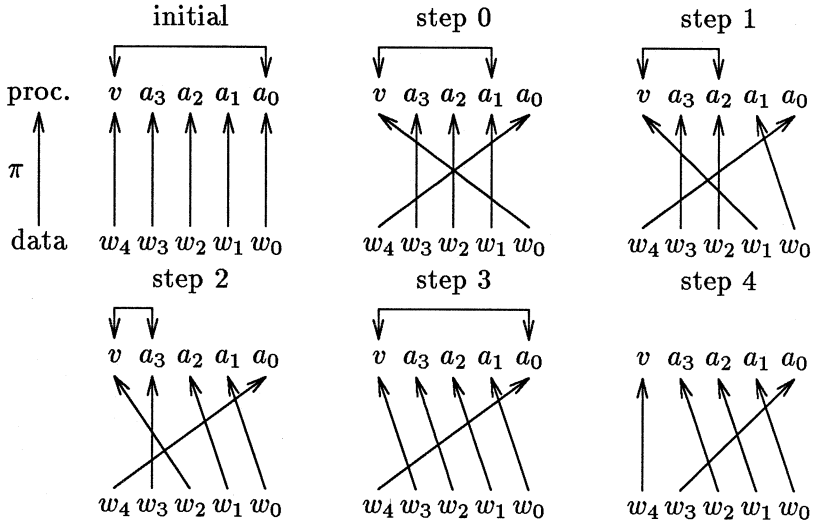

Figure 4: The changes of data allocations along time.

$$\rightarrow (v_{k-1}\ldots v_{k-\eta}\underline{r_i}v_{k-\eta-2}\ldots v_0|r_{n-1}r_{n-2}\ldots r_{i+2}\underline{r_{i+1}}v_{k-\eta-1}r_{i-1}\ldots r_0)$$
$$\rightarrow (v_{k-1}\ldots v_{k-\eta}\underline{r_{i+1}}v_{k-\eta-2}\ldots v_0|r_{n-1}r_{n-2}\ldots \underline{r_{i+2}}r_iv_{k-\eta-1}r_{i-1}\ldots r_0)\rightarrow\cdots$$
$$\rightarrow (v_{k-1}\ldots v_{k-\eta}\underline{r_{i-2}}v_{k-\eta-2}\ldots v_0|r_{n-2}r_{n-3}\ldots r_iv_{k-\eta-1}\underline{r_{i-1}}r_{i-3}\ldots r_0r_{n-1})$$
$$\rightarrow (v_{k-1}\ldots v_{k-\eta}\underline{r_{i-1}}v_{k-\eta-2}\ldots v_0|r_{n-2}r_{n-3}\ldots r_i\underline{v_{k-\eta-1}}r_{i-2}\ldots r_0r_{n-1})$$
$$\rightarrow (v_{k-1}\ldots v_{k-\eta}v_{k-\eta-1}v_{k-\eta-2}\ldots v_0|r_{n-2}r_{n-3}\ldots r_ir_{i-1}r_{i-2}\ldots r_0r_{n-1}),$$

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

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

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

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

$$\text{Seq}_0 = \alpha_0, \alpha_1, \alpha_2, \alpha_0.$$
$$\text{Seq}_1 = \alpha_1, \alpha_2, \alpha_0, \alpha_1.$$
$$\text{Seq}_2 = \alpha_2, \alpha_0, \alpha_1, \alpha_2.$$

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

The *one-port* version of Algorithm A1′ uses a single dimension during each exchange step. An *n-port* version of the algorithm can be created by defining the exchange sequences Seq_i, $0 \leq i < n$,

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

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

4.3 Full-cube, separable dimension permutation algorithms

4.3.1 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*

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

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 the GSH's are optimum, then the resulting algorithm for the SDP remains optimum.

With *n-port communication*, σ_i ports can be used concurrently for each GSH, and all GSH's can be performed concurrently. The lemma below is independent of the algorithm chosen for the GSH.

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*

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

by exploiting concurrency within each GSH, and in time

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

by exploiting the concurrency fully.

4.3.2 Algorithms not using generalized shuffle permutations.

Algorithm A2. If the number of local storage dimensions $q - n$ is at least equal to the order σ_p of a separable SDP, then an arbitrary permutation can be performed as two successive *all-to-all personalized communications* in subcubes of dimension σ_p [14]. In *all-to-all personalized communication* [10] every processor has a unique piece of data for every other processor. For each dimension $j \in \mathcal{J}$ there exists a unique dimension $k_j \in \Gamma_s$, such that the SDP can be performed in two steps $E(k_j, j), \forall j \in \mathcal{J}$ followed by $E(\delta(j), k_j), \forall j \in \mathcal{J}$. Assume for simplicity that the σ_p lowest order storage dimensions are used in the permutations, and that the SDP involves the σ_p lowest order processor dimensions. Then, the two *all-to-all personalized communications* can be illustrated by

$$(v_{s-1}v_{s-2}\ldots v_{\sigma_p}\underline{v_{\sigma_p-1}\ldots v_0}|r_{n-1}r_{n-2}\ldots r_{\sigma_p}\underline{r_{\sigma_p-1}r_{\sigma_p-2}\ldots r_0})$$

$$\rightarrow (v_{s-1}v_{s-2}\ldots v_{\sigma_p}\underline{r_{\sigma_p-1}\ldots r_0}|r_{n-1}r_{n-2}\ldots r_{\sigma_p}\underline{v_{\sigma_p-1}v_{\sigma_p-2}\ldots v_0}),$$

and

$$(v_{s-1}v_{s-2}\ldots v_{\sigma_p}\underline{r_{\sigma_p-1}\ldots r_0}|r_{n-1}r_{n-2}\ldots r_{\sigma_p}\underline{v_{\sigma_p-1}v_{\sigma_p-2}\ldots v_0})$$

$$\rightarrow (v_{s-1}v_{s-2}\ldots v_{\sigma_p}\underline{v_{\sigma_p-1}\ldots v_0}|r_{n-1}r_{n-2}\ldots r_{\sigma_p}\underline{r_{\delta^{-1}(\sigma_p-1)}r_{\delta^{-2}(\sigma_p-2)}\ldots r_{\delta^{-1}(0)}}).$$

With *one-port communication*, an optimum routing algorithm (within a factor of two) is described in [13] and [10]. With *n-port communication*, optimum algorithms (within a factor of two) are given in [10], [14].

Algorithm A3. An SDP can also be implemented by exchanging subsets of dimensions recursively in $\lceil \log_2 \sigma_p \rceil$ steps for a GSH of order σ_p, and $\lceil \log_2 \sigma_{max} \rceil$ steps for an SDP. Each step is of the same complexity as that of matrix transposition with two dimensional partitioning [8]. With *one-port communication*, the Single Path recursive Transpose (SPT) algorithm without pipelining [8] 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 A1. With *n-port communication*, the best known algorithm for matrix transposition with two-dimensional partitioning is described in [2], [8] and [14]. The complexity of Algorithm A3 is given in Table 3. The complexity of Algorithm A3 is in general an order of $\log \sigma_{max}$ higher than the lower bound. But for the special case where $\sigma_i = 2$ for all i's, the SDP degenerates to matrix transposition, and an optimal transpose algorithm should be used.

4.3.3 Combining generalized shuffle permutations

Algorithm A4. Algorithm A4 is designed for *n-port communication*, and hence uses multiple paths for each source destination pair to make good use of the available communication bandwidth. Algorithm A1 is used for each path. Thereby, one dimension is used twice for each GSH and each path. The paths are defined by the ordered set of dimensions

$$\begin{aligned} \text{Seq}_0 &= \hat{\mathcal{J}}_0, \alpha_{00}, \hat{\mathcal{J}}_1, \alpha_{10}, \ldots, \hat{\mathcal{J}}_{\beta-1}, \alpha_{(\beta-1)0}. \\ \text{Seq}_k &= d_0, d_1, \ldots, d_{\sigma_p+\beta-1}, \\ \text{Seq}_{k+1} &= d_1, d_2, \ldots, d_{\sigma_p+\beta-1}, \delta(d_0), \qquad \forall 0 \le k < \sigma_p + \beta + \sigma_{min} - 2. \end{aligned}$$

Note that $\delta(d_0)$ is not necessary d_1, Table 1. The first dimension of every GSH in every sequence is subject to two communications. The number of paths that can be generated by the left shift and appending the first exchange dimension subject to the constraint of at most two communications in any dimension during any routing cycle is $\sigma_p + \beta + \sigma_{min} - 1$.

Sequence 0 is the dimension exchange sequence for Algorithm A1 for a real SDP and *one-port communication*. Sequence $k+1$ is defined in terms of sequence k by a left cyclic shift of Seq_k, followed by the application of the δ function to the last entry. In most cases, the last entry is the same as the first entry, but it is different when the first GSH in the sequence is completed. Table 1 shows an example of 13 sequences of processor dimensions, which are used to perform the real SDP: $\hat{\mathcal{J}}_0 = 0, 1, 2, 3$, $\hat{\mathcal{J}}_1 = 4, 5, 6$, and $\hat{\mathcal{J}}_2 = 7, 8$. Each sequence represents an SDP for one data partition and all 13 SDP's are performed concurrently. Row i applies to the ith SDP, and column j to cycle j. The table entry is the processor dimension subject to communication.

Lemma 7 *The SDP's defined by the sequences Seq_i, $0 \le i < \sigma_p + \beta + \sigma_{min} - 1$, have the property that any processor dimension is used by at most two SDP's during any routing cycle.*

Proof: We consider three cases: (1) dimensions α_{i0}, (2) dimension α_{i1}, and (3) other dimensions. Let α_{i0} occur in Seq_0 at cycles j and $j + \sigma_i$ (with cycle 0 being the starting cycle). Then, from the construction of Seq_{k+1} out of Seq_k, α_{i0} also occurs in Seq_k at the cycle(s) defined below:

$$Position(\alpha_{i0}) = \begin{cases} j-k, j+\sigma_i-k, & 0 \le k \le j; \\ j+\sigma_i-k, & j+1 \le k \le j+\sigma_i-1; \\ 0, \sigma_p+\beta-1, & k = j+\sigma_i; \\ \sigma_p+\beta-1-k+j+\sigma_i, & j+\sigma_i+1 \le k \le \sigma_p+\beta+\sigma_i-2. \end{cases}$$

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 A1 with *n-port communication*.

Let α_{i1} occur in Seq_0 at cycle j, then it also occurs as follows:

$$Position(\alpha_{i1}) = \begin{cases} j-k, & 0 \le k < j; \\ 0, \sigma_p+\beta-1, & k = j; \\ \sigma_p+\beta-1-k+j, & j+1 \le k \le j+\sigma_i-1; \\ \sigma_p+\beta-1-k+j, \sigma_p+\beta-1-k+j+\sigma_i, & j+\sigma_i \le k \le \sigma_p+\beta+\sigma_i-2. \end{cases}$$

Let α_{il}, $2 \le l < \sigma_i$, occur in Seq_0 at cycle j, then it also occurs as follows:

$$Position(\alpha_{il}) = \begin{cases} j-k, & 0 \le k < j; \\ 0, \sigma_p+\beta-1, & k = j; \\ \sigma_p+\beta-1-k+j, & j+1 \le k \le \sigma_p+\beta+\sigma_i-2. \end{cases} \quad \blacksquare$$

Figure 5 illustrates the positions for α_{i0} and α_{i1} with their positions interconnected by solid and dashed lines. Each row represents a sequence and each column represents the dimensions used by all sequences during the same routing cycle.

Each sequence performs the SDP for $\frac{K}{\sigma_p+\beta+\sigma_{min}-1}$ elements. The complexity is

$$\begin{aligned} & T^{A4}_{sdp}(\sigma_p, \sigma_p, n, K) \\ &= \frac{\sigma_p+\beta}{\sigma_p+\beta+\sigma_{min}-1} K t_c + \lceil \frac{K}{(\sigma_p+\beta+\sigma_{min}-1)B} \rceil (\sigma_p+\beta)\tau \\ &= \frac{\sigma_p+\beta}{\sigma_p+\beta+\sigma_{min}-1} K t_c + (\sigma_p+\beta)\tau \qquad \text{with } B_{opt} = \frac{K}{\sigma_p+\beta+\sigma_{min}-1} \\ &\le \frac{\sigma_p+\beta}{\sigma_p+\beta+1} K t_c + \frac{3}{2}\sigma_p\tau. \end{aligned}$$

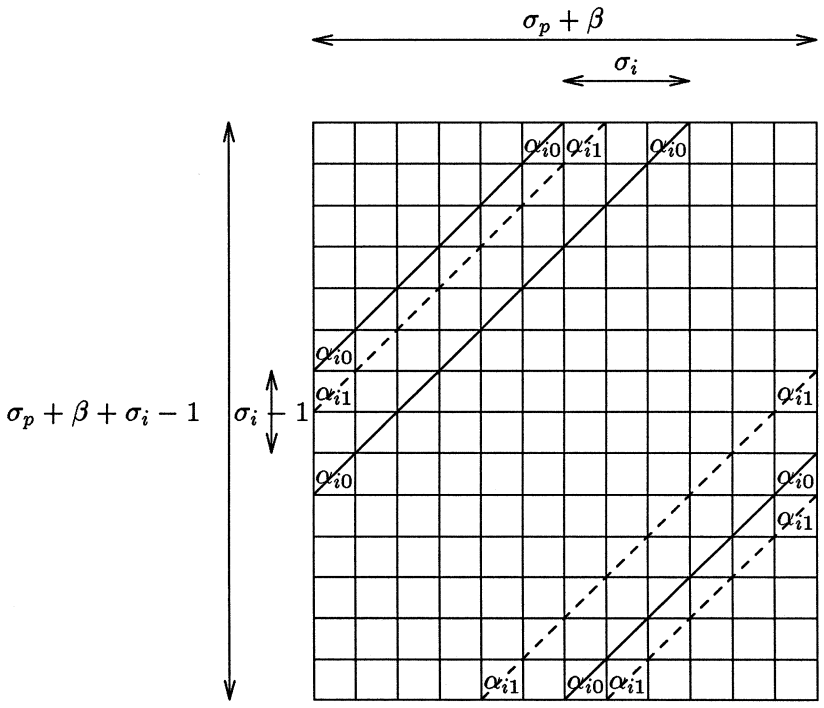

Figure 5: The cycles during which dimensions α_{i0} (α_{i1}) is used.

Note that the number of start-ups is minimal using Algorithm A1. The data transfer time is also of optimum order, and higher than minimum by at most a factor of two. This is mostly due to the fact that two communications are needed for some dimensions in every cycle. Note further, that this fact is used to generate a number of exchange sequences that exceed the number of dimensions that need to be routed, thereby improving somewhat on the use of the available communication bandwidth. Algorithm A4 degenerates to the *n-port* version of Algorithm A1 when the SDP degenerates to a GSH, i.e., $\beta = 1$.

Algorithm A5. As in Algorithm A4, Algorithm A5 is designed for *n-port communication*. In Algorithm A5 the number of start-ups is reduced by partitioning each GSH of order greater than $\gamma \leq \sigma_{max}$, to be chosen optimally, into multiple GSH's of order γ. An interpartition GSH is required to realize the original GSH.

As an example, consider the shuffle operation

$$(a_8a_7a_6a_5a_4a_3a_2a_1a_0) \rightarrow (a_7a_6a_5a_4a_3a_2a_1a_0a_8).$$

It can be realized as

$$(a_8a_7a_6a_5a_4a_3\underline{a_2a_1a_0}) \rightarrow (a_8a_7a_6\underline{a_5a_4a_3}a_1a_0a_2) \rightarrow (\underline{a_8a_7a_6}a_4a_3a_5a_1a_0a_2)$$
$$\rightarrow (a_7a_6\underline{a_8}a_4a_3\underline{a_5}a_1a_0\underline{a_2}) \rightarrow (a_7a_6a_5a_4a_3a_2a_1a_0a_8).$$

The index set $\mathcal{J} = \{0,1,2,3,4,5,6,7,8\}$ in the example is partitioned into three sets $\mathcal{J}_0 = \{0,1,2\}$, $\mathcal{J}_1 = \{3,4,5\}$, and $\mathcal{J}_2 = \{6,7,8\}$ with a combining set $\mathcal{J}_3 = \{3,6,0\}$.

Lemma 8 *A GSH with index set $\mathcal{J}$ of order σ can be performed as μ independent GSH's on index sets $\mathcal{J}_j$, $0 \le j < \mu$, where $\mathcal{J}_j = \{\alpha_{f(j)}, \alpha_{f(j)+1}, \alpha_{f(j)+2}, \ldots, \alpha_{f(j+1)-1}\}$ and $0 = f(0) < f(1) < \cdots < f(\mu) = \sigma$, followed by a GSH on the index set $\mathcal{J}_\mu = \{\alpha_{f(1)}, \ldots, \alpha_{f(\mu-1)}, \alpha_{f(0)}\}$.*

Proof: Define GSH_j, $0 \le j \le \mu$, to be a GSH with index set $\mathcal{J}_j$. Also, let σ_j be the order of GSH_j. Then by definition, GSH_j, $0 \le j < \mu$, realizes $\alpha_{f(j)+i} \rightarrow \alpha_{f(j)+(i+1) \bmod \sigma_j}, 0 \le i < \sigma_j$, which is equal to

$$\begin{cases} \alpha_i \rightarrow \alpha_{(i+1) \bmod \sigma}, & \text{if } i \ne f(j+1)-1, 0 \le j < \mu, \\ \alpha_i = \alpha_{f(j+1)-1} \rightarrow \alpha_{f(j)}, & \text{if } i = f(j+1)-1, 0 \le j < \mu. \end{cases}$$

But, GSH_μ realizes $\alpha_{f(j)} \rightarrow \alpha_{f((j+1) \bmod \mu)}, 0 \le j < \mu$, and the proof is complete. ∎

If each GSH is performed according to Algorithm A1, then a direct application would yield $\sum_{i=0}^{\mu-1}(\sigma_i+1)+\mu+1 = \sigma+2\mu+1$. But, it is unnecessary to perform the last restoring dimension exchange for *any* of the index sets, as illustrated next.

$$(v_2 v_1 \underline{v_0} | r_8 r_7 r_6 r_5 r_4 r_3 r_2 r_1 \underline{r_0}) \rightarrow (v_2 \underline{v_1} r_0 | r_8 r_7 r_6 r_5 r_4 \underline{r_3} r_2 r_1 v_0) \rightarrow (\underline{v_2} r_3 r_0 | r_8 r_7 \underline{r_6} r_5 r_4 v_1 r_2 r_1 v_0)$$

$$\rightarrow (r_6 r_3 \underline{r_0} | r_8 r_7 v_2 r_5 r_4 v_1 r_2 \underline{r_1} v_0) \rightarrow (r_6 \underline{r_3} r_1 | r_8 r_7 v_2 r_5 \underline{r_4} v_1 r_2 r_0 v_0) \rightarrow (\underline{r_6} r_4 r_1 | r_8 \underline{r_7} v_2 r_5 r_3 v_1 r_2 r_0 v_0)$$

$$\rightarrow (r_7 r_4 \underline{r_1} | r_8 r_6 v_2 r_5 r_3 v_1 \underline{r_2} r_0 v_0) \rightarrow (r_7 \underline{r_4} r_2 | r_8 r_6 v_2 \underline{r_5} r_3 v_1 r_1 r_0 v_0) \rightarrow (\underline{r_7} r_5 r_2 | \underline{r_8} r_6 v_2 r_4 r_3 v_1 r_1 r_0 v_0)$$

$$\rightarrow (r_8 r_5 \underline{r_2} | r_7 r_6 v_2 r_4 r_3 \underline{v_1} r_1 r_0 v_0) \rightarrow (r_8 \underline{r_5} v_1 | r_7 r_6 \underline{v_2} r_4 r_3 r_2 r_1 r_0 v_0) \rightarrow (\underline{r_8} v_2 v_1 | r_7 r_6 r_5 r_4 r_3 r_2 r_1 r_0 \underline{v_0})$$

$$\rightarrow (v_0 v_2 v_1 | r_7 r_6 r_5 r_4 r_3 r_2 r_1 r_0 r_8).$$

Note that a local shuffle is required to restore the original local storage map.

For Algorithm A5, we partition GSH_i of the SDP into shuffles on subsets according to Lemma 8 such that $|\mathcal{J}_{i_j}| = \gamma$, $0 \le i < \beta$, $0 \le j < \mu_i$ where $\mu_i = \lceil \frac{\sigma_i}{\gamma} \rceil$ is the number of subsets for the ith GSH. If σ_i is not a multiple of γ, then "dummy" dimensions ϕ are added to $\mathcal{J}_i$ such that $|\mathcal{J}_i|$ is a multiple of γ. The total number of GSH's after the partitioning is $\mu = \sum_{i=0}^{\beta-1} \mu_i$, each of order γ. For example, for an SDP with $\beta = 2$, $\mathcal{J}_0 = \{0,1,2,3,4,5,6,7\}$, $\mathcal{J}_1 = \{8,9,10\}$ and γ chosen to be 3, then $\mathcal{J}_0$ becomes $\{0,1,2,3,4,5,6,7,\phi\}$, $\mathcal{J}_1$ is still $\{8,9,10\}$, $\mu_0 = 3$, $\mu_1 = 1$ and $\mu = \mu_0 + \mu_1 = 4$. $\mathcal{J}_{0_0} = \{0,1,2\}$, $\mathcal{J}_{0_1} = \{3,4,5\}$, $\mathcal{J}_{0_2} = \{6,7,\phi\}$, $\mathcal{J}_{1_0} = \{8,9,10\}$.

We will now construct $\gamma\mu$ sequences that can be performed concurrently. Each sequence requires $(\gamma+1)\mu$ routing cycles, if the GSH permutations are performed sequentially, using μ different virtual dimensions. During routing cycle i virtual dimension $v_{i \bmod \mu}$ is used. For the first μ sequences of real dimensions, we first define sequences $\hat{\mathcal{S}}_k(0)$ of length μ which will serve as a basis.

$$\hat{\mathcal{S}}_k(0) = \begin{cases} \alpha^0_{0\ k}, \alpha^0_{0\ \gamma+k}, \ldots, \alpha^0_{0\ (\mu_0-1)\gamma+k}, \alpha^0_{1\ k}, \alpha^0_{1\ \gamma+k}, \ldots, \alpha^0_{1\ (\mu_1-1)\gamma+k}, \ldots, \\ \quad \alpha^0_{\beta-1\ k}, \alpha^0_{\beta-1\ \gamma+k}, \ldots, \alpha^0_{\beta-1\ (\mu_{\beta-1}-1)\gamma+k}, & \text{if } 0 \le k < \gamma, \\ \alpha^0_{0\ \gamma}, \alpha^0_{0\ 2\gamma}, \ldots, \alpha^0_{0\ (\mu_0-1)\gamma}, \alpha^0_{0\ 0}, \alpha^0_{1\ \gamma}, \alpha^0_{1\ 2\gamma}, \ldots, \alpha^0_{1\ (\mu_1-1)\gamma}, \alpha^0_{1\ 0}, \ldots, \\ \quad \alpha^0_{\beta-1\ \gamma}, \alpha^0_{\beta-1\ 2\gamma}, \ldots, \alpha^0_{\beta-1\ (\mu_{\beta-1}-1)\gamma}, \alpha^0_{\beta-1\ 0}, & \text{if } k = \gamma, \end{cases}$$

where $\alpha^0_{i\ j} = \alpha_{i\ j}$. The sequence $\hat{\mathcal{S}}_k(0)$, $0 \le k < \gamma$ simply consists of the kth dimension of each of the ordered sets $\mathcal{J}_{i_j}$. The sequence $\hat{\mathcal{S}}_\gamma(0)$ consists of the first dimension of every subset $\mathcal{J}_{i_j}$

taken in order for each i, left rotated one step, and then concatenated in order of increasing i. This sequence performs the coupling between the partitions within each GSH_i. For the same SDP example, $\hat{\mathcal{S}}_0(0) = 0,3,6,8$, $\hat{\mathcal{S}}_1(0) = 1,4,7,9$, $\hat{\mathcal{S}}_2(0) = 2,5,\phi,10$ and $\hat{\mathcal{S}}_3(0) = 3,6,0,8$.

The exchange sequences $\mathrm{Seq}_j, 0 \le j < \mu$ is defined by

$$\mathrm{Seq}_j = L^j(\hat{\mathcal{S}}_0(0)), L^j(\hat{\mathcal{S}}_1(0)), \ldots, L^j(\hat{\mathcal{S}}_\gamma(0)), \quad 0 \le j < \mu.$$

See the first four sequences in Table 2 for an example. Additional sequences are formed by forming new sets $\hat{\mathcal{S}}_k(\ell)$ by partitioning the ordered sets obtained through an ℓ step left rotation of each set $\hat{\mathcal{J}}_i$, $0 \le i < \beta$. So,

$$\hat{\mathcal{S}}_k(\ell) = \begin{cases} \alpha^\ell_{0\ k}, \alpha^\ell_{0\ \gamma+k}, \ldots, \alpha^\ell_{0\ (\mu_0-1)\gamma+k}, \alpha^\ell_{1\ k}, \alpha^\ell_{1\ \gamma+k}, \ldots, \alpha^\ell_{1\ (\mu_1-1)\gamma+k}, \ldots, \\ \quad \alpha^\ell_{\beta-1\ k}, \alpha^\ell_{\beta-1\ \gamma+k}, \ldots, \alpha^\ell_{\beta-1\ (\mu_{\beta-1}-1)\gamma+k}, & \text{if } 0 \le k < \gamma, \\ \alpha^\ell_{0\ \gamma}, \alpha^\ell_{0\ 2\gamma}, \ldots, \alpha^\ell_{0\ (\mu_0-1)\gamma}, \alpha^\ell_{0\ 0}, \alpha^\ell_{1\ \gamma}, \alpha^\ell_{1\ 2\gamma}, \ldots, \alpha^\ell_{1\ (\mu_1-1)\gamma}, \alpha^\ell_{1\ 0}, \ldots, \\ \quad \alpha^\ell_{\beta-1\ \gamma}, \alpha^\ell_{\beta-1\ 2\gamma}, \ldots, \alpha^\ell_{\beta-1\ (\mu_{\beta-1}-1)\gamma}, \alpha^\ell_{\beta-1\ 0}, & \text{if } k = \gamma, \end{cases}$$

where $\alpha^\ell_{i\ j} = \alpha_{i\ (j+\ell) \bmod \mu_i \gamma}$. In general,

$$\mathrm{Seq}_{\mu\ell+j} = L^j(\hat{\mathcal{S}}_0(\ell)), L^j(\hat{\mathcal{S}}_1(\ell)), \ldots, L^j(\hat{\mathcal{S}}_\gamma(\ell)), \quad 0 \le j < \mu, 0 \le \ell < \gamma.$$

For the same SDP example, $\hat{\mathcal{S}}_0(1) = 1,4,7,9$, $\hat{\mathcal{S}}_1(1) = 2,5,\phi,10$, $\hat{\mathcal{S}}_2(1) = 3,6,0,8$ and $\hat{\mathcal{S}}_3(1) = 4,7,1,9$. Note that $\hat{\mathcal{S}}_k(\ell) = \hat{\mathcal{S}}_{k+j}(\ell - j)$ for $\ell \le j \le \mu - k$. Table 2 shows the 12 sequences for the same SDP example.

Note that μ virtual dimensions are required for each sequence. With a data set of size at least 2^μ per processor, the permutation corresponding to exchange sequences $\hat{\mathcal{S}}_k(\ell)$ are equivalent to *one-port* all-to-all personalized communication in a μ-dimensional subcube.

Lemma 9 *Each sequence of exchange operations defined above realizes the desired SDP.*

Proof: Since the original β GSH's have disjoint index sets, we only need to consider the relative order for each GSH. By construction, the virtual dimensions for each of the β GSH's are disjoint. The lemma applied to sequence Seq_0 follows directly from Lemma 8. But, since the starting index is immaterial, and all other sequences have the prescribed exchange order for the dimensions within each GSH, the proof is complete. ∎

Lemma 10 *The $\gamma\mu$ exchange sequences defined by different $Seq_{\mu\ell+j}$ can be executed concurrently without edge contention in the case of n-port communication.*

The lemma is easily proved from the construction of the sequences.

Figures 6 and 7 show the first four sequences of exchange operations (Seq_0 to Seq_3) with SDP of $\mathcal{J}_0 = \{0,1,2,3,4,5,6,7,8\}$, $\mathcal{J}_1 = \{9,10,11\}$ and SDP of $\mathcal{J}_0 = \{0,1,2,3,4,5,6,7\}$,

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

Table 2: The 12 sequences for an SDP with $\beta = 2$, $\mathcal{J}_0 = \{0,1,2,3,4,5,6,7\}$, $\mathcal{J}_1 = \{8,9,10\}$ and $\gamma = 3$.

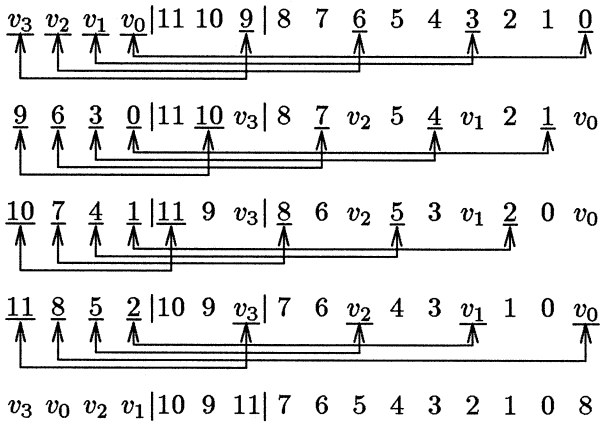

Figure 6: The first four sequences of exchange operations (Seq_0 to Seq_3) with SDP of $\mathcal{J}_0 = \{0,1,2,3,4,5,6,7,8\}$, $\mathcal{J}_1 = \{9,10,11\}$, $\beta = 2$ and $\gamma = 3$.

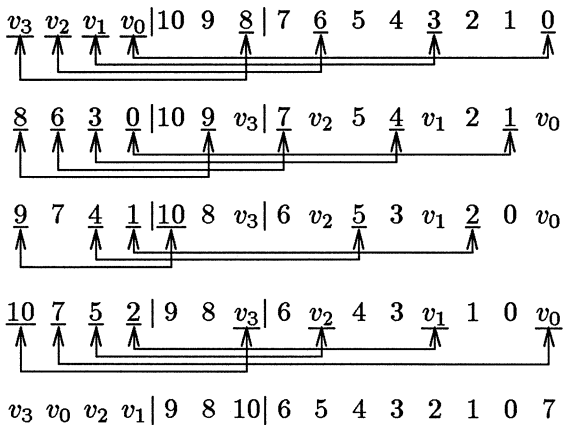

Figure 7: The first four sequences of exchange operations (Seq_0 to Seq_3) with SDP of $\mathcal{J}_0 = \{0,1,2,3,4,5,6,7\}$, $\mathcal{J}_1 = \{8,9,10\}$, $\beta = 2$ and $\gamma = 3$.

$\mathcal{J}_1 = \{8,9,10\}$, respectively, and both with $\beta = 2$ and $\gamma = 3$. For simplicity, each step in the figures combine four sequential communication steps.

For a given γ, the communication complexity with *n-port communication* is

$$\frac{\gamma+1}{2\gamma}Kt_c + (\gamma+1)\left\lceil\frac{K}{2\gamma B}\right\rceil\sum_{i=0}^{\beta-1}\left\lceil\frac{\sigma_{p_i}}{\gamma}\right\rceil\tau = \frac{\gamma+1}{2\gamma}Kt_c + (\gamma+1)\mu\tau \qquad \text{with} \qquad B = B_{opt}.$$

For example, for $\beta = 2$, $\sigma_0 = 9$, $\sigma_1 = 4$, the number of start-ups are 21, 20, 20, 18, and 20 for $\gamma = 2,3,4,5$, and 9, respectively. For $\beta = 2$, $\sigma_0 = 6$, $\sigma_1 = 4$, the number of start-ups are 15, 16, 15, and 14 for $\gamma = 2,3,4$ and 6, respectively.

For $\gamma = 2$ and optimum packet size, the complexity becomes

$$\frac{3}{4}Kt_c + \frac{3(\sigma_p + oJ')}{2}\tau \leq \frac{3}{4}Kt_c + 2\sigma_p\tau,$$

where oJ' is the number of GSH's with an odd real order. It is at most $\frac{\sigma_p}{3}$. For $\gamma = \sigma_{max}$, the number of start-ups is $\beta(\sigma_{max}+1)$. The start-up time is minimized for $2 \leq \gamma \leq \sigma_{max}$. A loose upper bound of the number of start-ups for any γ in the range of $[2, \sigma_{max}]$ is $\frac{3}{2}(\sigma_p + (\sigma_{max}-1)\beta)$, which is at most $\frac{3}{4}(\sigma_p^2 - \sigma_p)$.

Algorithm A6. As in Algorithm A4 we assume *n-port communication*. Since every data element has to be permuted according to all GSH's making up the SDP, performing all GSH's concurrently implies $\beta(\sigma_{max}+1)$ start-ups, since the GSH of maximum order has to be performed β times. By organizing the GSH's into groups it may be possible to reduce the number of start-ups, and the total communication time. Ideally, all groups require the same time. For instance, if a real SDP consists of three real GSH's of order 7, 3, and 3, then the data set is partitioned into two parts of $\frac{K}{2}$ elements each (assuming that Algorithm A1 is used for each real GSH). One part is permuted through the GSH of real order 7. The other part is partitioned further into

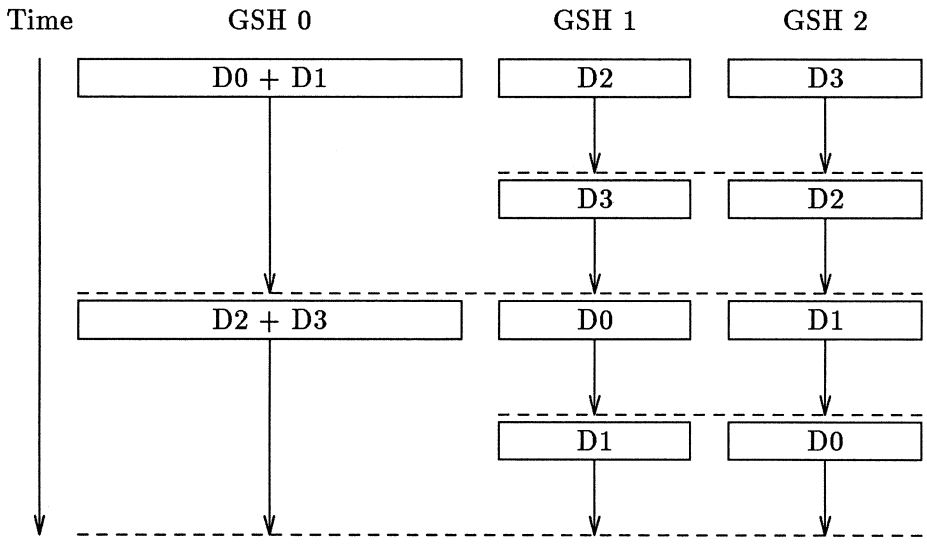

Figure 8: Hierarchical partitioning of the data set for Algorithm A6 and an SDP consisting of three GSH's of order 7, 3 and 3.

$\frac{K}{4}$ elements each, one for each real GSH of 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 A1 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 real GSH of order 7 is then permuted according to the two GSH's of 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. For non-separable SDP's it may not be necessary to extend the set of dimensions for Algorithm A1. Figure 8 illustrates the data movement for this example, where D0 to D3 are the four equal partitions of the data set.

Specifically, let SDP_0, SDP_1, ..., $\text{SDP}_{\mu-1}$ be some μ-partitioning of the given SDP, i.e., the β GSH's are distributed in the μ partitioned SDP's and $\mu \leq \beta$. Then, the given SDP can be solved recursively in time

$$T(\text{SDP}, K) = \mu * \max\{T(\text{SDP}_i, \frac{K}{\mu}), \forall 0 \leq i < \mu\},$$

where $T(\text{SDP}, K)$ is the time for an SDP with data volume K. This is derived simply by partitioning the data volume into μ equal parts, leting different parts participate in different sequences of SDP_i's while all sequences of SDP_i's are performed concurrently.

Recall that the complexity of Algorithm A1 for a GSH of order σ_i and data volume K is

$$\frac{(\sigma_i + 1)}{2\sigma_i} K t_c + (\sigma_i + 1)\tau.$$

The data transfer time is almost independent of the real order of the SDP using Algorithms A1. Hence, 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. The total number of start-ups is less than $2(\sigma_p + \beta)$ for any SDP and may be as small as $\sigma_p + \beta$ for some SDP's. Each element s_i of set S below will be interpreted to $\sigma_i + 1$ in Theorem 2, which is the number of start-ups required for the ith GSH. The minimum number of total start-ups, $f(S)$, is then defined recursively as the minimum of the largest start-ups per set ($\max_i f(S_i)$) multiplies the number of sets (μ) for any μ partitioning of S and any μ.

Lemma 11 *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. Denote $Sum(S) = \sum_{i=0}^{\beta-1} s_i$. Define $f(S)$ as follows:*

$$f(S) = \begin{cases} i, & \text{if } S = \{i\}, \\ \min_{\forall \mu \text{ partitioning of } S}\{\mu \times \max_i f(S_i), & \\ \quad \text{where } S_i\text{'s, } 0 \le i < \mu, \text{ are the } \mu \text{ partitioning sets of } S\}, & \text{otherwise.} \end{cases}$$

Then, $Sum(S) \le f(S) < 2Sum(S)$.

Proof: Clearly, $f(S) \ge Sum(S)$. Consider $\mu = 2$ for $f(S)$ and assume $|S| \ge 2$ in the following. We will define a sequence of multi-sets $S^{(0)}, S^{(1)}, \ldots, S^{(j)}$ inductively where $x \ge 2^i$, $\forall x \in S^{(i)}$. Let $S = S^{(0)}$ and $S^{(i+1)}$ be derived from $S^{(i)}$, $0 \le i < j$, as follows. Let the number of integer elements in $S^{(i)} \cap [2^i, 2^{i+1})$ be y and let them be ordered such that $x_0 \le x_1 \le \cdots \le x_{y-1}$. Then, let $2 \cdot \max(x_{2z}, x_{2z+1})$ be an element of $S^{(i+1)}$ for $z = \{0, 1, \ldots, \lfloor \frac{y}{2} \rfloor - 1\}$. If there is an element left unpaired, then simply increase its value to 2^{i+1} and include it in $S^{(i+1)}$. Remaining elements of $S^{(i)}$ are included in $S^{(i+1)}$. For example, if $S^{(3)} = \{8, 9, 11, 13, 14, 17, 21\}$ then $S^{(4)} = \{18, 26, 16, 17, 21\}$. Notice that $Sum(S^{(i+1)}) - Sum(S^{(i)}) \le 2^i$. Furthermore, $f(S^{(i+1)}) \ge f(S^{(i)})$, since for every μ partitioning of $S^{(i+1)}$ there exists a corresponding μ partitioning of $S^{(i)}$ such that $\mu \times \max_{i'} f(S_{i'}^{(i)}) \le \mu \times \max_{i'} f(S_{i'}^{(i+1)})$ where $S_{i'}^{(i)}$'s are μ partitioned sets of $S^{(i)}$. This procedure is iterated until a set $S^{(j)}$ is reached such that $|S^{(j)}| = 2$ and its two elements are in the range $[2^j, 2^{j+1})$. Let these two elements be x and x' with $x' \ge x$. We now show $x' < Sum(S)$. By $Sum(S^{(i+1)}) - Sum(S^{(i)}) \le 2^i$, one can derive $Sum(S^{(j)}) - Sum(S) \le 2^j - 1$, i.e., $x + x' < Sum(S) + 2^j$. This means $x' < Sum(S)$, since $x \ge 2^j$. So, $f(S^{(j)}) = 2x' < 2Sum(S)$. On the other hand, $f(S) = f(S^{(0)}) \le f(S^{(1)}) \le \cdots \le f(S^{(j)})$. Therefore, $f(S) < 2Sum(S)$. ∎

Any SDP algorithm can be employed for each group of GSH's. We only consider Algorithm A1, if a group of GSH's consists of a single GSH. Consequently, Algorithm A6 degenerates to a concurrent application of Algorithm A1 to each GSH, if there is only one GSH per group for every group.

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 based on the grouping of GSH's of an SDP where the number of start-ups is $f(S)$ and the data transfer time is $\frac{\min(S)K}{2(\min(S)-1)} t_c$.*

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 specifies 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 A1, $T(\{\sigma_i+1\},K) = \frac{(\sigma_i+1)K}{2\sigma_i}t_c + (\sigma_i+1)\lceil\frac{K}{2\sigma_i B}\rceil\tau$. 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, $0 \leq i < \mu$, are the μ 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 μ equal subsets, called subsets $0,1,\ldots,\mu-1$. The data of subset i participates in a sequence of SDP_i, $\mathrm{SDP}_{(i+1)\bmod\mu},\ldots,\mathrm{SDP}_{(i-1)\bmod\mu}$, 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 μ subsets are of the same size, the time to realize the SDP with optimum packet size is at most

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

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 11. ∎

Corollary 5 *With n-port communication and optimum packet size, the dimension permutation can be realized by an algorithm based on a partitioning strategy with a complexity of at most*

$$\begin{aligned} T^{A6}_{sdp}(n,\sigma_p,n,K) &\leq \frac{(\sigma_{min}+1)K}{2\sigma_{min}}t_c + (2(\sigma_p+\beta)-1)\tau \\ &\leq \frac{3K}{4}t_c + (3\sigma_p-1)\tau. \end{aligned}$$

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 $\max(\frac{K}{8},\frac{K}{2\sigma_p})$.

4.4 Full cube, non-separable dimension permutation algorithms

With *one-port communication*, the dimension permutation is simply obtained by performing β GSH's in sequence. Algorithm A1 should be used for the mixed cycles. For real cycles either Algorithm A1 or A1′ should be used depending on K, t_c, τ and σ_{p_i}. Let β' be the number of real GSH's. The communication complexity is

$$T^{\mathrm{A1}}_{sdp}(1,\sigma_p,n,K) = (\sigma_p+\beta')\frac{K}{2}t_c + (\sigma_p+\beta')\lceil\frac{K}{2B}\rceil\tau,$$

by using Algorithm A1.

With *n-port communication*, one can employ the same algorithm as if there is no virtual dimension, for instance Algorithms A4, A5, or A6. The communication complexity is determined by the number of real dimensions (since the time for local data movement is ignored). The number of virtual dimensions introduced for the exchange sequences is $\sigma_p - m'_s$, where m'_s is the number of virtual dimensions that have a succeeding real dimension in the index set. The number of exchange sequences that can be run concurrently without any extra storage requirement (extra virtual dimensions) is m'_s.

A non-separable SDP for which $\mathcal{R}^b \cap \mathcal{R}^a = \phi$ is an all-to-all personalized communication. In such a case the complexity of Algorithm A2 in Table 3 should be halved, since the table entry assumes $\mathcal{R}^b = \mathcal{R}^a$. The complexity of Algorithm A2 may be preferable for non-separable SDP's.

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

All presented algorithms have a communication complexity of the same order as the lower bound, except in a few cases. The difference in the communication complexities of the algorithms and the lower bound is generally a small constant factor. The control is distributed for all algorithms. The communication complexities are summarized in Table 3. The second last column contains the ratio of data transfer times and the lower bound. The last column contains

Comm.	Alg.	B_{opt}	Communication complexity	t_c factor/$l\,b$	τ factor/$l\,b$
sep.	A1	$\frac{K}{2}$	$\frac{(\sigma_p+\beta)}{2}Kt_c + (\sigma_p+\beta)\lceil\frac{K}{2B}\rceil\tau$	(1, 1.5]	(1, 1.5]
SDP	A1′	K	$\sigma_p K t_c + \sigma_p\lceil\frac{K}{B}\rceil\tau$	2	1
one-port	A2	$\frac{K}{2}$	$\sigma_p K t_c + 2\sigma_p\lceil\frac{K}{2B}\rceil\tau$	2	2
comm.	A3	K	$\lceil\log_2\sigma_{max}\rceil(\sigma_p K t_c + \sigma_p\lceil\frac{K}{B}\rceil\tau)$	$[2, 2\lceil\log_2\sigma_{max}\rceil]$	$[1, \lceil\log_2\sigma_{max}\rceil]$
	A1	$\frac{K}{2\sigma_{min}\beta}$	$\beta\max_i\{\frac{(\sigma_i+1)K}{2\sigma_i\beta}t_c + (\sigma_i+1)\lceil\frac{K}{2\sigma_i\beta B}\rceil\tau\}$	(1, 1.5]	$(1, \frac{(\sigma_p+3)^2}{8}]$
sep.	A1′	$\frac{K}{\sigma_{max}\beta}$	$Kt_c + \beta\sigma_{max}\lceil\frac{K}{\sigma_{max}\beta B}\rceil\tau$	2	$[1, \frac{(\sigma_p+2)^2}{8}]$
SDP	A2	$\frac{K}{2\sigma_p}$	$Kt_c + 2\sigma_p\lceil\frac{K}{2\sigma_p B}\rceil\tau$	2	2
n-port	A3	$\sqrt{\frac{K\tau}{2t_c}}$	$\lceil\log_2\sigma_{max}\rceil(\frac{K}{2B}+1)(\tau+Bt_c)$	$(1, 2\lceil\log_2\sigma_{max}\rceil]$	
comm.	A4	$\frac{K}{\sigma_p+\beta+\sigma_{min}-1}$	$\frac{(\sigma_p+\beta)Kt_c}{\sigma_p+\beta+\sigma_{min}-1} + \lceil\frac{K}{(\sigma_p+\beta+\sigma_{min}-1)B}\rceil(\sigma_p+\beta)\tau$	(1, 2)	(1, 1.5]
	A5	$\frac{K}{2\gamma}$	$\frac{\gamma+1}{2\gamma}Kt_c + (\gamma+1)\lceil\frac{K}{2\gamma B}\rceil\sum_{i=0}^{\beta-1}\lceil\frac{\sigma_i}{\gamma}\rceil\tau$	(1, 1.5]	(1, 2]
	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 3: Summary of the communication complexities for various algorithms. Note that $\sigma_i = \sigma_{p_i}$. The complexity for Algorithm A6 is with $B = B_{opt}$. The complexity with arbitrary B is shown in the proof of Theorem 2. The factor of start-up times for A5 is for $\mu = 2$.

the ratio of the start-up times and the lower bound[3]. Algorithm A3 is not optimal if $\log_2\sigma_{max} > O(1)$. Also, Algorithms A1 and A1′ (in general) are not optimum for *n-port communication*, and optimally chosen packet sizes. However, the complexities of Algorithms A1 and A1′ are comparable with those of Algorithms A4, A5 and A6 for a packet size that is very small compared to the data volume. Furthermore, the complexity estimates for Algorithm A2 assume two *all-to-all personalized communications*. For an SDP containing all mixed GSH's of order 2, one such communication may suffice and the estimates are pessimistic by a factor of two.

For *one-port communication* the number of start-ups of the algorithms is at most twice the lower bound, except for Algorithm A3 that may have a factor of $\lceil\log_2\sigma_{max}\rceil$ more start-ups. Algorithm A1′ has a number of start-ups equal to the lower bound. For *n-port communication* Algorithms A2, A4, A5, and A6 all have a number of start-ups that are higher than the lower bound by at most a factor of three, with Algorithm A4 potentially having the fewest start-ups. The data transfer time for Algorithm A1 with *one-port communication* is at most 50% higher than the lower bound, and that of Algorithms A1′ and A2 is twice the lower bound. For *n-port communication* all algorithms (with the exception of Algorithm A3) have a data transfer time that is at most a factor of two higher than the lower bound.

We conclude that the data transfer time of Algorithm A1 is a factor of $\frac{\sigma_p+\beta}{2\sigma_p}$ lower than that of Algorithm A1′ for *one-port communication* and about a factor of $\frac{\sigma_{min}+1}{2\sigma_{min}}$ lower for *n-port communication*. The factor ranges from 0.5 - 0.75 for both *one-port* and *n-port communications*.

[3]For convenience, we use σ_p (instead of $\sigma_p - oJ$, Theorem 1) as the lower bound for comparison.

Algorithm	σ_i's = {3, 6}	σ_i's = {2, 2, 5}	σ_i's = {2, 5, 11}
A4	$\frac{11}{13}Kt_c + 11\tau$	$\frac{12}{13}Kt_c + 12\tau$	$\frac{21}{22}Kt_c + 21\tau$
A5	$\frac{2}{3}Kt_c + 12\tau$, $\gamma = 3$ $\frac{7}{12}Kt_c + 14\tau$, $\gamma = 6$	$\frac{3}{4}Kt_c + 15\tau$, $\gamma = 2$ $\frac{2}{3}Kt_c + 16\tau$, $\gamma = 3$	$\frac{7}{12}Kt_c + 28\tau$, $\gamma = 6$ $\frac{6}{11}Kt_c + 36\tau$, $\gamma = 11$
A6	$\frac{2}{3}Kt_c + 14\tau$	$\frac{3}{4}Kt_c + 12\tau$	$\frac{3}{4}Kt_c + 24\tau$

Table 4: Compleixty of Algorithms A4, A6 and A5 for some example SDP's.

With optimum packet size the number of start-ups of Algorithm A1 is a factor of $1 + \frac{\beta}{\sigma_p}$ higher than that of Algorithm A1′ for *one-port communication* and a factor of $1 + \frac{1}{\sigma_{max}}$ 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 A1′ is relatively more competitive for *n-port communication* than for *one-port communication* with optimum packet size. The break-even point between Algorithms A1 and A1′ is $\tau = (\frac{\sigma_p}{\beta} - 1)\frac{K}{2}t_c$ for *one-port communication*, and $\tau = (1 - \frac{1}{\sigma_{max}})\frac{Kt_c}{2\beta}$ for *n-port communication* and $\tau \geq \frac{Kt_c}{2\sigma_{max}\sigma_{min}}$. (Note that the complexity of Algorithm A1 for *n-port communication* has a maximum value at $\sigma_i = \sigma_{max}$ if $\tau \geq \frac{Kt_c}{2\sigma_{max}\sigma_{min}}$; and a maximum value at $\sigma_i = \sigma_{min}$, otherwise.)

With *n-port communication* Algorithms A4, A5, and A6 are the algorithms of choice, unless the start-up time is negligible, or all cycles are of the same real order. The complexity of Algorithm A5 is bounded from above by $\frac{3}{4}Kt_c + 2\sigma_p\tau$ by choosing $\gamma = 2$. The number of start-ups in the complexity estimate for Algorithm A6 is for the worst case, Lemma 11. It may be as small as $\sigma_p + \beta$. The break-even point between Algorithms A4 and A5 is approximately $\tau = \frac{Kt_c}{4(\sigma_p - \beta)}$. The break-even point between Algorithms A4 and A6 is approximately $\tau = \frac{Kt_c}{4(\sigma_p + \beta - 1)}$. From the complexity estimates, Algorithm A5 is always better than Algorithm A6 due to the worst case estimate of A6. In general, Algorithm A6 may be preferable to Algorithm A5.

Table 4 shows complexities of Algorithms A4, A5 and A6 for three examples. For the first example, both A4 and A5 may be preferable. For the second example, both A5 and A6 may be preferable. For the third example, any of A4, A5 and A6 may be preferable. The choice depends on K, t_c, τ and $\mathcal{J}$ etc. So does the choice of γ in Algorithm A5. Since the coefficient of τ in A5 $(\gamma + 1)\sum_{i=0}^{\beta-1}\lceil\frac{\sigma_i}{\gamma}\rceil \geq \sigma_p + \beta$, A4 has a lowest start-up time. The start-up time of A6 is within a factor two to that of A4. By choosing $\gamma = 2$, the start-up time of A5 is within a factor of two of that of A4. By choosing $\gamma = \sigma_{min}$ in A5, both A5 and A6 have the same data transfer time. However, either of them may have lower start-ups (as shown by the two examples) depending on the distribution of σ_i's. The data transfer time of A4 is within a factor of two to that of A5 and A6. The data transfer time in A5 decreases as γ increases and becomes the same as lower bound, asymptotically. However, minimizing the start-up time yields $2 \leq \gamma \leq \sigma_{max}$.

Algorithm A2 is inferior to Algorithms A4 and A5 for separable, stable SDP's, but may be preferable for non-separable SDP's. The complexity of Algorithm A3 is, in general, an order of $O(\log_2 \sigma_{max})$ higher than the lower bound, but has the lowest complexity of the considered

Algorithm	A1	A1′	A2	A3
Memory	K	$2K$	K	$2K$
$\min\{K\}$	2	1	$2^{\sigma_{max}}$	1

Table 5: The memory required and the minimum size of data volume required for maximum concurrency for different algorithms with *one-port communication*.

Algorithm	A1	A1′	A2	A3	A4	A5	A6
Memory	K	$2K$	K	$2K$	K	K	K
$\min\{K\}$	$2\beta\sigma_{max}$	σ_p	$\beta 2^{\sigma_{max}}$	$2\sigma_{max}$	$2(\sigma_p+\beta+\sigma_{min}-1)$	$\gamma\mu 2^{\mu}$	$\leq 2\sigma_p$

Table 6: The memory required and the minimum size of data volume required for maximum concurrency for different algorithms with *n-port communication*.

algorithms for $\sigma_i \leq 2$, $\forall 0 \leq i < \beta$, and *n-port communication*.

Table 5 shows for different *one-port* algorithms, the memory required and the minimum size of the data set K for which the complexity estimates are true. The corresponding estimates for *n-port* algorithms are given in Table 6. All algorithms require a memory which is at most twice the original data volume. Recall in Algorithm A2, we perform all-to-all personalized communication for each σ_p-dimensional subcube. The minimum K for maximum concurrency is 2^{σ_p} for both *one-port* and *n-port communications*. However, in the *one-port communication* case, one can perform the all-to-all personalized communication within each σ_i-dimensional subcube in sequence for different i's and attains the same result with the same complexity. The minimum K required becomes $2^{\sigma_{max}}$ as shown in the table. In the *n-port communication* case, one can partition the data into β parts such that the ith partition participates the two all-to-all personalized communication in each σ_i-dimensional subcube defined by $\mathcal{J}_i$ and all partitions participate concurrently. Since the complexity of all-to-all personalized communication is independent of size of a cube in *n-port communication*, all partitions take the same time. The procedure is repeated β times in a rotated manner such that the ith partition participates in a subcube defined by $\mathcal{J}_i$, $\mathcal{J}_{(i+1) \bmod \beta}$, $\ldots$, $\mathcal{J}_{(i-1) \bmod \beta}$, respectively, in order. The resulting complexity remains the same as the original algorithm, but the maximum K required is reduced to $\beta 2^{\sigma_{max}}$. For Algorithm A6, the maximum K required is $\max_i\{2^{\ell_i}\sigma_i\}$, where ℓ_i is the level of σ_i in the binary tree defined in the proof of Lemma 2. It can be shown that $2\sigma_p \geq 2^{\ell_i}\sigma_i$, so the maximum K required is at most $2\sigma_p$.

We now discuss Algorithms A4, A5 and A6 based on *one-port* version of Algorithm A1′. Recall the complexity of A1′ for a GSH of order σ_i, *one-port communication* and with optimum packet size is

$$\sigma_i K t_c + \sigma_i \tau.$$

Performing permutation for the next GSH before finishing permutation for the current GSH

causes the data per active processor to be permuted for the next GSH doubled. A4 based on A1′ increases the data volume by a factor of two, since there is an unfinished GSH. The complexity is

$$2Kt_c + \sigma_p\tau.$$

A6 based on A1′ has a complexity of

$$Kt_c + (2\sigma_p - 1)\tau.$$

Note that the complexity of A6 based on A1′ is always higher than that of A4 (based on A1). However, the $(2\sigma_p - 1)\tau$ start-up time is the worst case for *any* SDP. The best case is $\sigma_p\tau$ for some SDP's for which all the partitioned groups have the same number of start-ups for all levels of recursion. A5 based on A1′ has the problem that the number of active processors halved for every unfinished GSH. Since there are $\mu - 1$ unfinished *new* GSH's in general, the data transfer time is at least an order of $O(2^\mu)$ higher than the lower bound. We conclude that Algorithm A4 based on Algorithm A1′ has an optimal start-up time (in comparing to σ_p). The data transfer time is a factor of four to the lower bound. A5 and A6 based on Algorithm A1′ are not competitive except for some special cases of SDP's for A6.

5 Summary and conclusions

We have derived lower bounds for *stable* dimension permutations on Boolean cubes both for *one-port communication* and *n-port communication*, and devised several optimal algorithms for *one-port communication* and *n-port communication*. Optimal algorithms for *one-port communication* can be directly derived from optimal algorithms for generalized shuffle permutations [9]. However, devising optimal algorithms for the *n-port communication* case is non-obvious. A naive generalization would lead to either non-optimal start-up time or poor bandwidth utilization. The proposed three new algorithms, A4 - A6, use different strategies for creating concurrent exchange sequences, such as rotation, permutation and recursive grouping of the index set. We have also showed that for full-cube permutations on a subset of the processor dimensions no reduction in complexity is possible for optimal algorithms by using processor dimensions not participating in the permutation. The presentation of the algorithms also illustrates a general methodology for devising communication algorithms for *n-port communication*. The minimum number of start-ups in sequence is equal to the number of processor dimensions in the permutation both for *one-port communication* and *n-port communication*. The optimum data transfer time in the *n-port communication* case is proportional to the size of the data set per processor. For *one-port communication* it increases by a factor approximately equal to the number of processor dimensions in the permutation. 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.

Acknowledgement

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

References

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

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

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

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

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

[8] S. Lennart Johnsson and Ching-Tien Ho. Matrix transposition on Boolean n-cube configured ensemble architectures. *SIAM J. Matrix Anal. Appl.*, 9(3):419–454, July 1988. YALE/DCS/RR-572, September 1987. (Revised edition of YALEU/DCS/RR-494 November 1986.).

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

[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. Revised November 1987, YALEU/DCS/RR-610. 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*, pages 223–231, 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. 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.