Asymptotically Tight Bounds for Performing BMMC Permutations on Parallel Disk Systems

Thomas H. Cormen*
Thomas Sundquist†
Leonard F. Wisniewski‡

Abstract

This paper presents asymptotically equal lower and upper bounds for the number of parallel I/O operations required to perform bit-matrix-multiply/complement (BMMC) permutations on the Parallel Disk Model proposed by Vitter and Shriver. A BMMC permutation maps a source index to a target index by an affine transformation over $GF(2)$, where the source and target indices are treated as bit vectors. The class of BMMC permutations includes many common permutations, such as matrix transposition (when dimensions are powers of 2), bit-reversal permutations, vector-reversal permutations, hypercube permutations, matrix reblocking, Gray-code permutations, and inverse Gray-code permutations. The upper bound improves upon the asymptotic bound in the previous best known BMMC algorithm and upon the constant factor in the previous best known bit-permute/complement (BPC) permutation algorithm. The algorithm achieving the upper bound uses basic linear-algebra techniques to factor the characteristic matrix for the BMMC permutation into a product of factors, each of which characterizes a permutation that can be performed in one pass over the data.

The factoring uses new subclasses of BMMC permutations: memoryload-dispersal (MLD) permutations and their inverses. These subclasses extend the catalog of one-pass permutations.

Although many BMMC permutations of practical interest fall into subclasses that might be explicitly invoked within the source code, this paper shows how to detect quickly whether a given vector of target addresses specifies a BMMC permutation. Thus, one can determine efficiently at run time whether a permutation to be performed is BMMC and then avoid the general-permutation algorithm and save parallel I/Os by using the BMMC-permutation algorithm herein.

*Dartmouth College Department of Computer Science. Portions of this research were performed while at the MIT Laboratory for Computer Science and appear in [9]; supported in part by the Defense Advanced Research Projects Agency under Grant N00014-91-J-1698 during that time. Other portions of this research were performed while at Dartmouth College and were supported in part by funds from Dartmouth College and in part by the National Science Foundation under Grant CCR-9308667.

†Dartmouth College Department of Mathematics. Supported in part by funds from Dartmouth College.

‡Thinking Machines Corporation. Research performed while at the Dartmouth College Department of Computer Science. Supported in part by INFOSEC Grant 3-56666, in part by the National Science Foundation under Grant CCR-9308667, and in part by a Dartmouth Graduate Fellowship.

An extended abstract of this paper appeared in the Proceedings of the 5th Annual ACM Symposium on Parallel Algorithms and Architectures.

1 Introduction

From both the theoretical and practical points of view, permuting is an interesting and important problem when the data reside on disk. As one of the most basic data-movement operations, permuting is central to the theory of I/O complexity. The problems that we attack with supercomputers are ever-increasing in size, and in several applications matrices and vectors exceed the memory provided by even the largest supercomputers. (Such applications include seismic problems, computational fluid dynamics, and processing large images. For a list of Grand Challenge applications with huge I/O requirements, see the list compiled by del Rosario and Choudhary [14].) One solution is to store large matrices and vectors on parallel disk systems. The high latency of disk accesses makes it essential to minimize the number of disk I/O operations. Permuting the elements of a matrix or vector is a common operation, particularly in the data-parallel style of computing, and good permutation algorithms can provide significant savings in disk-access costs over poor ones when the data reside on parallel disk systems.

This paper examines the class of bit-matrix-multiply/complement (BMMC) permutations for parallel disk systems and derives four important results:

1. a universal lower bound for BMMC permutations,
2. an algorithm for performing BMMC permutations whose I/O complexity asymptotically matches the lower bound, thus making it asymptotically optimal,
3. an efficient method for determining at run time whether a given permutation is BMMC, thus allowing us to use the BMMC algorithm if it is, and
4. two new subclasses of BMMC permutations, memoryload-dispersal (MLD) permutations and their inverses, which we show how to perform in one pass.

Depending on the exact BMMC permutation, our asymptotically optimal bound may be significantly lower than the asymptotically optimal bound proven for general permutations. Moreover, the low constant factor in our algorithm makes it very practical.

Model and previous results

We use the Parallel Disk Model first proposed by Vitter and Shriver [24], who also gave asymptotically optimal algorithms for several problems including sorting and general permutations. In the Parallel Disk Model, N *records* are stored on D *disks* $\mathcal{D}_0, \mathcal{D}_1, \ldots, \mathcal{D}_{D-1}$, with N/D records stored on each disk. The records on each disk are partitioned into *blocks* of B records each. When a disk is read from or written to, an entire block of records is transferred. Disk I/O transfers records between the disks and a *random-access memory* (which we shall refer to simply as "memory") capable of holding M records. Each *parallel I/O operation* transfers up to D blocks between the disks and memory, with at most one block transferred per disk, for a total of up to BD records transferred. We assume *independent I/O*, in which the blocks accessed in a single parallel I/O may be at any locations on their respective disks, as opposed to *striped I/O*, which has the restriction that the blocks accessed in a given operation must be at the same location on each disk.

We measure an algorithm's efficiency by the number of parallel I/O operations it requires. Although this cost model does not account for the variation in disk access times caused by head movement and rotational latency, programmers often have no control over these factors. The

	$\mathcal{D}_0$		$\mathcal{D}_1$		$\mathcal{D}_2$		$\mathcal{D}_3$		$\mathcal{D}_4$		$\mathcal{D}_5$		$\mathcal{D}_6$		$\mathcal{D}_7$	
stripe 0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
stripe 1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
stripe 2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
stripe 3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63

Figure 1: The layout of $N = 64$ records in a parallel disk system with $B = 2$ and $D = 8$. Each box represents one block. The number of stripes is $N/BD = 4$. Numbers indicate record indices.

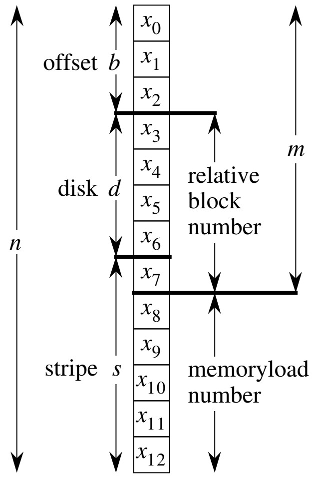

Figure 2: Parsing the address $x = (x_0, x_1, \ldots, x_{n-1})$ of a record on a parallel disk system. Here, $n = 13$, $b = 3$, $d = 4$, $m = 8$, and $s = 6$. The least significant b bits contain the offset of a record within its block, the next d bits contain the disk number, and the most significant s bits contain the stripe number. The most significant $n - m$ bits form the record's memoryload number, and bits $b, b+1, \ldots, m-1$ form the relative block number, used in Section 4.

number of disk accesses, however, can be minimized by carefully designed algorithms. Optimal algorithms have appeared in the literature for fundamental problems such as sorting [3, 6, 21, 22, 24], general permutations [24], and structured permutations [9, 10, 26], as well as higher-level domains such as Fast Fourier transform [24], matrix-matrix multiplication [24], LUP decomposition [27], computational geometry problems [5, 18], graph algorithms [8], and boolean function manipulation [4].

For convenience, we use the following notation extensively:

$$b = \lg B\ , \quad d = \lg D\ , \quad m = \lg M\ , \quad n = \lg N\ .$$

We shall assume that b, d, m, and n are nonnegative integers, which implies that B, D, M, and N are exact powers of 2. In order for the memory to accomodate the records transferred in a parallel I/O operation to all D disks, we require that $BD \leq M$. Also, we assume that $M < N$, since otherwise we can just perform all operations in memory. These two requirements imply that $b + d \leq m < n$.

The Parallel Disk Model lays out data on a parallel disk system as shown in Figure 1. A *stripe*

Permutation	Characteristic matrix	Number of passes
BMMC (bit-matrix-multiply/ complement)	nonsingular matrix A	$2\left\lceil \frac{\lg M - r}{\lg(M/B)} \right\rceil + H(N,M,B)$
BPC (bit-permute/ complement)	permutation matrix A	$2\left\lceil \frac{\rho(A)}{\lg(M/B)} \right\rceil + 1$
MRC (memory-rearrangement/ complement)	$\begin{array}{cc} \begin{array}{cc} m & n-m \end{array} & \\ \left[\begin{array}{c\|c} \text{nonsingular} & \text{arbitrary} \\ \hline 0 & \text{nonsingular} \end{array}\right] & \begin{array}{c} m \\ n-m \end{array} \end{array}$	1

Table 1: Classes of permutations, their characteristic matrices, and upper bounds shown in [10] on the number of passes needed to perform them. A pass consists of reading and writing each record exactly once and therefore uses exactly $2N/BD$ parallel I/Os. For MRC permutations, submatrix dimensions are shown on matrix borders. For BMMC permutations, r is the rank of the leading $\lg M \times \lg M$ submatrix of A, and the function $H(N,M,B)$ is given by equation (1). For BPC permutations, the function $\rho(A)$ is defined in equation (3).

consists of the D blocks at the same location on all D disks. We indicate the *address*, or *index*, of a record as an n-bit vector x with the least significant bit first: $x = (x_0, x_1, \ldots, x_{n-1})$. Record indices vary most rapidly within a block, then among disks, and finally among stripes. As Figure 2 shows, the offset within the block is given by the least significant b bits $x_0, x_1, \ldots, x_{b-1}$, the disk number by the next d bits $x_b, x_{b+1}, \ldots, x_{b+d-1}$, and the stripe number by the $s = n - (b+d)$ most significant bits $x_{b+d}, x_{b+d+1}, \ldots, x_{n-1}$.

Since each parallel I/O operation accesses at most BD records, any algorithm that must access all N records requires $\Omega(N/BD)$ parallel I/Os, and so $O(N/BD)$ parallel I/Os is the analogue of linear time in sequential computing. Vitter and Shriver showed an upper bound of $\Theta\left(\min\left(\frac{N}{D}, \frac{N}{BD}\frac{\lg(N/B)}{\lg(M/B)}\right)\right)$ parallel I/Os for general permutations, that is, for arbitrary mappings $\pi : \{0,1,\ldots,N-1\} \xrightarrow{\text{1-1}} \{0,1,\ldots,N-1\}$. The first term comes into play when the block size B is small, and the second term is the sorting bound $\Theta\left(\frac{N}{BD}\frac{\lg(N/B)}{\lg(M/B)}\right)$, which was shown by Vitter and Shriver for randomized sorting and subsequently by Nodine and Vitter [22] and others [3, 6, 21] for deterministic sorting. These bounds are asymptotically tight, because they match the lower bounds proven earlier by Aggarwal and Vitter [2] using a model with one disk and D independent read/write heads, which is at least as powerful as the Parallel Disk Model.

Specific classes of permutations sometimes require fewer parallel I/Os than general permutations. Vitter and Shriver showed how to transpose an $R \times S$ matrix ($N = RS$) with only $\Theta\left(\frac{N}{BD}\left(1 + \frac{\lg\min(B,R,S,N/B)}{\lg(M/B)}\right)\right)$ parallel I/Os. Subsequently, Cormen [10] studied several classes of bit-defined permutations that include matrix transposition as a special case. Table 1 shows some of the classes of permutations examined and the corresponding upper bounds derived in [10].

BMMC permutations

The most general class considered in [10] is *bit-matrix-multiply/complement*, or *BMMC*, permutations.[1] A BMMC permutation is specified by an $n \times n$ *characteristic matrix* $A = (a_{ij})$ whose entries are drawn from $\{0, 1\}$ and is nonsingular (i.e., invertible) over $GF(2)$.[2] The specification also includes a *complement vector* $c = (c_0, c_1, \ldots, c_{n-1})$ of length n. Treating a *source address* x as an n-bit vector, we perform matrix-vector multiplication over $GF(2)$ and then form the corresponding n-bit *target address* y by complementing some subset of the resulting bits: $y = A\,x \oplus c$, or

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} & a_{02} & \cdots & a_{0,n-1} \\ a_{10} & a_{11} & a_{12} & \cdots & a_{1,n-1} \\ a_{20} & a_{21} & a_{22} & \cdots & a_{2,n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n-1,0} & a_{n-1,1} & a_{n-1,2} & \cdots & a_{n-1,n-1} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{n-1} \end{bmatrix} \oplus \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{n-1} \end{bmatrix} .$$

Because we require the characteristic matrix A to be nonsingular, the mapping of source addresses to target addresses is one-to-one. (This property is a consequence of Lemma 3 in Section 2.)

We shall generally focus on the matrix-multiplication portion of BMMC permutations rather than on the complement vector. The permutation π_A characterized by a matrix A is the permutation for which $\pi_A(x) = A\,x$ for all source addresses x.

The following lemma shows the equivalence of multiplying characteristic matrices and composing permutations when the complement vectors are zero. For permutations π_Y and π_Z, the *composition* $\pi_Z \circ \pi_Y$ is defined by $(\pi_Z \circ \pi_Y)(x) = \pi_Z(\pi_Y(x))$ for all x in the domain of π_Y.

Lemma 1 *Let Z and Y be nonsingular $n \times n$ matrices and let π_Z and π_Y be the permutations characterized by Z and Y, respectively. Then the matrix product ZY characterizes the composition $\pi_Z \circ \pi_Y$.*

Proof: For any source address x, we have

$$\begin{aligned} (\pi_Z \circ \pi_Y)(x) &= \pi_Z(\pi_Y(x)) \\ &= \pi_Z(Y\,x) \\ &= Z(Y\,x) \\ &= (ZY)x \ , \end{aligned}$$

and so the matrix product ZY characterizes the composition $\pi_Z \circ \pi_Y$. ■

When we factor a characteristic matrix A into the product of several nonsingular matrices, each factor characterizes a BMMC permutation. The following corollary describes the order in which we perform these permutations to effect the permutation characterized by A.

Corollary 2 *Let the $n \times n$ characteristic matrix A be factored as $A = A^{(k)}\,A^{(k-1)}\,A^{(k-2)} \cdots A^{(1)}$, where each factor $A^{(i)}$ is a nonsingular $n \times n$ matrix. Then we can perform the BMMC permutation*

[1] Edelman, Heller, and Johnsson [15] call BMMC permutations *affine transformations* or, if there is no complementing, *linear transformations*.

[2] Matrix multiplication over $GF(2)$ is like standard matrix multiplication over the reals but with all arithmetic performed modulo 2. Equivalently, multiplication is replaced by logical-and, and addition is replaced by exclusive-or.

characterized by A by performing, in order, the BMMC permutations characterized by $A^{(1)}, A^{(2)}$, $\ldots, A^{(k)}$. That is, we perform the permutations characterized by the factors of a matrix from right to left.

Proof: The proof is a simple induction, using Lemma 1. ■

The BMMC algorithm in [10] exploits Corollary 2 to factor a characteristic matrix into a product of other characteristic matrices, performing the permutations given by the factors right to left. It uses

$$\frac{2N}{BD}\left(2\left\lceil\frac{\lg M - r}{\lg(M/B)}\right\rceil + H(N,M,B)\right)$$

parallel I/Os, where r is the rank of the leading $\lg M \times \lg M$ submatrix of the characteristic matrix and

$$H(N,M,B) = \begin{cases} 4\left\lceil\dfrac{\lg B}{\lg(M/B)}\right\rceil + 9 & \text{if } M \le \sqrt{N}\ , \\ 4\left\lceil\dfrac{\lg(N/B)}{\lg(M/B)}\right\rceil + 1 & \text{if } \sqrt{N} < M < \sqrt{NB}\ , \\ 5 & \text{if } \sqrt{NB} \le M\ . \end{cases} \tag{1}$$

One can adapt the lower bound proven in this paper to show that $\Omega\left(\frac{N}{BD}\frac{\lg M - r}{\lg(M/B)}\right)$ parallel I/Os are necessary (see Section 2.8 of [9]), but so far it has been unknown whether the $\Theta\left(\frac{N}{BD}\, H(N,M,B)\right)$ term is necessary in all cases. This paper shows that it is not.

BPC permutations

By restricting the characteristic matrix A of a BMMC permutation to be a permutation matrix—having exactly one 1 in each row and each column—we obtain the class of *bit-permute/complement*, or *BPC*, permutations.[3] One can think of a BPC permutation as forming each target address by applying a fixed permutation to the source-address bits and then complementing a subset of the resulting bits. The class of BPC permutations includes many common permutations such as matrix transposition (when dimensions are powers of 2), bit-reversal permutations (used in performing FFTs), vector-reversal permutations, hypercube permutations, and matrix reblocking.

Previous work [10] expressed the I/O complexity of BPC permutations in terms of cross-ranks. For any $n \times n$ permutation matrix A and for any $k = 0, 1, \ldots, n-1$, the *k-cross-rank* of A is

$$\rho_k(A) = \operatorname{rank} A_{k..n-1,0..k-1} = \operatorname{rank} A_{0..k-1,k..n-1}\ , \tag{2}$$

where, for example, $A_{k..n-1,0..k-1}$ denotes the submatrix of A consisting of the intersection of rows $k, k+1, \ldots, n-1$ and columns $0, 1, \ldots, k-1$. The *cross-rank* of A is the maximum of the b- and m-cross-ranks:

$$\rho(A) = \max(\rho_b(A), \rho_m(A))\ . \tag{3}$$

The BPC algorithm in [10] uses at most

$$\frac{2N}{BD}\left(2\left\lceil\frac{\rho(A)}{\lg(M/B)}\right\rceil + 1\right)$$

[3] Johnsson and Ho [19] call BPC permutations *dimension permutations*, and Aggarwal, Chandra, and Snir [1] call BPC permutations without complementing *rational permutations*.

parallel I/Os. One can adapt the lower bound we prove in Section 3 for BMMC permutations to show that this BPC algorithm is asymptotically optimal. The BMMC algorithm in Section 6, however, is asymptotically optimal for all BMMC permutations—including those that are BPC—and it reduces the innermost factor of 2 in the above bound to a factor of 1. Not only is the BPC algorithm in [10] improved upon by the results in this paper, but the notion of cross-rank appears to be obviated as well.

MRC permutations

Memory-rearrangement/complement, or *MRC*, permutations are BMMC permutations with the additional restrictions shown in Table 1: both the leading $m \times m$ and trailing $(n-m) \times (n-m)$ submatrices of the characteristic matrix are nonsingular, the upper right $m \times (n-m)$ submatrix can contain any 0-1 values at all, and the lower left $(n-m) \times m$ submatrix is all 0. Cormen [10] shows that any MRC permutation requires only one pass of N/BD parallel reads and N/BD parallel writes. If we partition the N records into N/M consecutive sets of M records each, we call each set a *memoryload*. Each memoryload consists of M/BD consecutive stripes in which all addresses have the same value in the most significant $n-m$ bits, as Figure 2 shows. Any MRC permutation can be performed by reading in a memoryload, permuting its records in memory, and writing them out to a (possibly different) memoryload number. Because a memoryload may be read and written with striped I/Os, any MRC permutation may be performed with striped reads and striped writes. The class of MRC permutations includes those characterized by unit upper-triangular matrices. As [10] shows, both the standard binary-reflected Gray code and its inverse have characteristic matrices of this form, and so they are MRC permutations.

MLD permutations

We define here a new BMMC permutation subclass, which we shall use in our asymptotically optimal BMMC algorithm. To define this subclass, we first need the standard linear-algebraic notion of a kernel. The *kernel* of any $p \times q$ matrix A is the set of q-vectors that map to 0 when multiplied by A. That is,

$$\ker A = \{x : A\,x = 0\} \ .$$

A *memoryload-dispersal*, or *MLD*, permutation has a characteristic matrix that is nonsingular and of the following form:

$$\begin{array}{cl}
\begin{array}{cc} \quad m \quad & \quad n-m \quad \end{array} & \\
\left[\begin{array}{c|c} \text{arbitrary} & \\ \hline \lambda & \text{arbitrary} \\ \hline \mu & \end{array}\right] & \begin{array}{l} b \\ m-b \\ n-m \end{array}
\end{array} \ ,$$

subject to the *kernel condition*

$$\ker \lambda \subseteq \ker \mu \tag{4}$$

or, equivalently, $\lambda x = 0$ implies $\mu x = 0$.

As we shall see in Section 4, the kernel condition implies that we can perform any MLD permutation in one pass by reading in each source memoryload, permuting its records in memory, and writing these records out to M/BD blocks on each disk. Although the blocks read from each memoryload must come from M/BD consecutive stripes, the blocks written may go to any locations at all, as long as M/BD blocks are written to each disk. That is, MLD permutations use striped reads and independent writes. We shall also see in Section 4 that we can perform the inverse of an MLD permutation in one pass with independent reads and striped writes.

Outline

The remainder of this paper is organized as follows. Section 2 reviews some fundamental linear-algebraic notions and proves some properties that we shall use in later sections. Section 3 states and proves the lower bound for BMMC permutations. Section 4 shows how to perform any MLD permutation in one pass and gives some additional properties of MLD permutations and their inverses. Section 5 previews several of the matrix forms used in Section 6, which presents an algorithm for BMMC permutations whose I/O complexity asymptotically matches the lower bound. Section 7 shows how to detect at run time whether a vector of target addresses describes a BMMC permutation, thus enabling us to determine whether the BMMC algorithm is applicable; this section also presents an easy method for determining whether a nonsingular matrix satisfies the kernel condition (4) and therefore characterizes an MLD permutation. Finally, Section 8 contains some concluding remarks.

The algorithms for MLD and BMMC permutations in Sections 4 and 6 take little computation time and space. (They do, however, require permutations to be performed in memory, and various architectures may differ in how efficiently they do so.) The data structures are vectors of length $\lg N$ or matrices of size at most $\lg N \times \lg N$. Even sequential algorithms for the harder computations (e.g., finding a maximal set of linearly independent columns of a bit matrix) take time polynomial in $\lg N$, in fact $O(\lg^3 N)$.

We shall not concern ourselves with memory issues when manipulating characteristic matrices. That is, we assume throughout this paper that $\lg^2 N \ll M$, since as a practical matter, the size of any characteristic matrix is much smaller than memory. Consider for example a problem with $N = 2^{60}$ records, or about one quintillion. (This problem is much larger than any problem that one is likely to see for a long time. If each record were only one byte long, such a data set would occupy one billion gigabytes.) A 60×60 characteristic matrix for a problem this large would require 3600 bits, or 120 words of 32 bits if each column is packed into two words. This amount is insignificant compared to memory sizes of even modest computer systems. Consequently, we shall think of the M-record memory as holding only records and not the characteristic matrix or complement vector.

We shall use several notational conventions in this paper, as in equation (2). Matrix row and column numbers are indexed from 0 starting from the upper left. Vectors are indexed from 0, too. We index rows and columns by sets to indicate submatrices, using ". ." notation to indicate sets of contiguous numbers. When a matrix is indexed by just one set rather than two, the set indexes column numbers; the submatrix consists of entire columns. When a submatrix index is a singleton set, we shall often omit the enclosing braces. We denote an identity matrix by I and a matrix whose entries are all 0s by 0; the dimensions of such matrices will be clear from their contexts. All matrix and vector elements are drawn from $\{0, 1\}$, and all matrix and vector arithmetic is over $GF(2)$. When convenient, we interpret bit vectors as the integers they represent in binary. Vectors are treated as 1-column matrices in context.

Some readers familiar with linear algebra may notice that a few of the lemmas in this paper are special cases of standard linear-algebra properties restricted to $GF(2)$. We include the proofs here for completeness.

2 Linear-algebraic fundamentals

This section reviews some standard linear-algebraic terms and proves a few simple properties that we shall use later on. It also shows how to find a maximal set of linearly independent columns of a bit matrix.

Ranges and preimages

For a $p \times q$ matrix A with 0-1 entries, we define the *range* of A by

$$\mathcal{R}(A) = \{y : y = Ax \text{ for some } x \in \{0, 1, \ldots, 2^q - 1\}\} ,$$

that is, $\mathcal{R}(A)$ is the set of p-vectors that can be produced by multiplying all q-vectors with 0-1 entries (interpreted as integers in $\{0, 1, \ldots, 2^q - 1\}$) by A over $GF(2)$. We also adopt the notation

$$\mathcal{R}(A) \oplus c = \{z : z = y \oplus c \text{ for some } y \in \mathcal{R}(A)\} ,$$

that is, $\mathcal{R}(A) \oplus c$ is the exclusive-or of the range of A and a fixed vector c.

Lemma 3 *Let A be a $p \times q$ matrix whose entries are drawn from $\{0, 1\}$, let c be any p-vector whose entries are drawn from $\{0, 1\}$, and let $r = \operatorname{rank} A$. Then $|\mathcal{R}(A) \oplus c| = 2^r$.*

Proof: Let S index a maximal set of linearly independent columns of A, so that $S \subseteq \{0, 1, \ldots, q-1\}$, $|S| = r$, the columns of the submatrix A_S are linearly independent, and for any column number $j \notin S$, the column A_j is linearly dependent on the columns of A_S. We claim that $\mathcal{R}(A) = \mathcal{R}(A_S)$. Clearly, $\mathcal{R}(A_S) \subseteq \mathcal{R}(A)$, since $\mathcal{R}(A)$ includes the sum (over $GF(2)$) of each subset of columns of A. To see that $\mathcal{R}(A) \subseteq \mathcal{R}(A_S)$, consider any q-vector $y \in \mathcal{R}(A)$. There is some set T of column indices such that $y = \bigoplus_{j \in T} A_j$. For each column index $j \in T - S$, let $S_j \subseteq S$ index the columns of A_S that A_j depends on: $A_j = \bigoplus_{k \in S_j} A_k$. Then we have

$$\begin{aligned} y &= \bigoplus_{j \in T} A_j \\ &= \left(\bigoplus_{j \in T \cap S} A_j \right) \oplus \left(\bigoplus_{j \in T - S} A_j \right) \\ &= \left(\bigoplus_{j \in T \cap S} A_j \right) \oplus \left(\bigoplus_{j \in T - S} \left(\bigoplus_{k \in S_j} A_k \right) \right) , \end{aligned}$$

and so y is a linear combination of columns of A_S. Thus, $y \in \mathcal{R}(A_S)$, which in turn proves that $\mathcal{R}(A) \subseteq \mathcal{R}(A_S)$ and consequently $\mathcal{R}(A) = \mathcal{R}(A_S)$.

We have $|\mathcal{R}(A_S)| = 2^{|S|} = 2^r$, since each vector in $\mathcal{R}(A_S)$ is the sum of a unique subset of the columns of S and each column index in S may or may not be included in a sum of the columns. Thus, $|\mathcal{R}(A)| = 2^r$. Exclusive-oring the result of the matrix multiplication by a constant p-vector does not change the cardinality of the range. Therefore, $|\mathcal{R}(A) \oplus c| = |\mathcal{R}(A)| = 2^r$. ∎

For a $p \times q$ matrix A and a p-vector $y \in \mathcal{R}(A)$, we define the *preimage* of y under A by

$$\operatorname{Pre}(A, y) = \{x : Ax = y\} .$$

That is, $\operatorname{Pre}(A, y)$ is the set of q-vectors x that map to y when multiplied by A.

Lemma 4 *Let A be a $p \times q$ matrix whose entries are drawn from $\{0,1\}$, let y be any p-vector in $\mathcal{R}(A)$, and let $r = \operatorname{rank} A$. Then $|\operatorname{Pre}(A,y)| = 2^{q-r}$.*

Proof: Let S index a maximal set of linearly independent columns of A, so that $S \subseteq \{0,1,\ldots,q-1\}$, $|S| = r$, the columns of the submatrix A_S are linearly independent, and for any column number $j \notin S$, the column A_j is linearly dependent on the columns of A_S. Let $S' = \{0,1,\ldots,q-1\} - S$.

We claim that for any value $i \in \{0,1,\ldots,2^{q-r}-1\}$, there is a unique q-vector $x^{(i)}$ for which $x^{(i)}_{S'}$ is the binary representation of i and $y = A\,x^{(i)}$. Why? We have $y = A_S\,x^{(i)}_S \oplus A_{S'}\,x^{(i)}_{S'}$ or, equivalently,

$$y \oplus A_{S'} x^{(i)}_{S'} = A_S\,x^{(i)}_S \ . \tag{5}$$

The columns of A_S span $\mathcal{R}(A)$, which implies that for all $z \in \mathcal{R}(A)$, there is a unique r-vector w such that $z = A_S\,w$. Letting $z = y \oplus A_{S'}\,x^{(i)}_{S'}$, we see that there is a unique r-vector $x^{(i)}_S$ that satisfies equation (5), which proves the claim.

Thus, we have shown that $|\operatorname{Pre}(A,y)| \geq 2^{q-r}$. If we had $|\operatorname{Pre}(A,y)| > 2^{q-r}$, then because y is arbitrarily chosen from $\mathcal{R}(A)$, we would have that $\sum_{y' \in \mathcal{R}(A)} |\operatorname{Pre}(A,y')| > 2^q$. But this inequality contradicts there being only 2^q possible preimage vectors. We conclude that $|\operatorname{Pre}(A,y)| = 2^{q-r}$. ∎

Row spaces

The *row space* of a matrix A, written $\operatorname{row} A$, is the span of the rows of A. We prove the following lemma about the relationship between kernels and row spaces, which we shall use later to prove properties resulting from the kernel condition of MLD permutations and to check that the kernel condition holds.

Lemma 5 *Let K and L be q-column matrices. Then* $\ker K \subseteq \ker L$ *if and only if* $\operatorname{row} L \subseteq \operatorname{row} K$*.*

Proof: For any vector space X, the *orthogonal space* of X, written $X^\perp$, is the set of vectors Y such that for all $x \in X$ and all $y \in Y$, the inner product $x \cdot y$ is 0. We use the following well-known facts from linear algebra (see Strang [23, pp. 138–139] for example):

1. The row space and the kernel are orthogonal spaces of each other. Thus, $(\operatorname{row} K)^\perp = \ker K$ and $(\operatorname{row} L)^\perp = \ker L$.

2. For any vector spaces X and Y, $X \subseteq Y$ implies $Y^\perp \subseteq X^\perp$.

3. For any vector space X, $(X^\perp)^\perp = X$.[4]

The latter two properties imply that if $Y^\perp \subseteq X^\perp$, then $X \subseteq Y$. Thus we have

$$\begin{aligned} \ker K \subseteq \ker L \quad & \text{iff} \quad (\operatorname{row} K)^\perp \subseteq (\operatorname{row} L)^\perp \\ & \text{iff} \quad \operatorname{row} L \subseteq \operatorname{row} K \ , \end{aligned}$$

which proves the lemma. ∎

[4]The proof that this property holds over $GF(2)$ is not as straightforward as the conventional proof that it holds over $\mathbf{R}^n$. Lang [20, p. 131] contains a proof for $GF(2)$.

Finding a maximal set of linearly independent columns

We conclude this section with a simple sequential algorithm to find a maximal set S of linearly independent columns of a $p \times q$ matrix K. We shall use this technique several times in this paper.

We use the following pseudocode:

```
1  S ← ∅
2  for each row index i ← 0 to p − 1 do
3     if there exists some column index j for which K_ij = 1
4        then for each column index j′ such that K_ij′ = 1
5                 add column j to column j′
6                 S ← S ∪ {j}
```

At the completion of this algorithm, the set S contains the indices for a maximal set of linearly independent columns of K. Lines 4–5 zero out any column in the set. Each iteration of the outer loop zeros out the next row. By the end of the algorithm, every column gets zeroed out as a column in S or by the addition of some subset of columns in S.

This algorithm takes $O(p^2 q)$ time on a sequential machine. In our applications of this algorithm, p and q are at most $\lg N$, and so the sequential time will always be $O(\lg^3 N)$.

3 A universal lower bound for BMMC permutations

In this section, we state and prove the lower bound for BMMC permutations. After stating the lower bound, we briefly discuss its significance before presenting the full proof. The lower bound is given by the following theorem.

Theorem 6 *Any algorithm that performs a non-identity BMMC permutation with characteristic matrix A requires*

$$\Omega\left(\frac{N}{BD}\left(1+\frac{\operatorname{rank}\gamma}{\lg(M/B)}\right)\right)$$

parallel I/Os, where γ is the submatrix $A_{b..n-1,0..b-1}$ of size $\lg(N/B) \times \lg B$. ∎

This lower bound is *universal* in the sense that it applies to all inputs other than the identity permutation, which of course requires no data movement at all. In contrast, lower bounds such as the standard $\Omega(N \lg N)$ lower bound for sorting N items on a sequential machine are *existential*: they apply to worst-case inputs, but for some inputs an algorithm may be able to do better.

Section 6 presents an algorithm that achieves the bound given by Theorem 6, and so this algorithm is asymptotically optimal.

Technique

To prove Theorem 6, we rely heavily on the technique used by Aggarwal and Vitter [2] to prove a lower bound on I/Os for matrix transposition; their proof is based in turn on a method by Floyd [16]. We prove the lower bound for the case in which $D = 1$; the general case follows by dividing by D. We consider only I/Os that are *simple*. An input is simple if each record read is removed from the disk and moved into an empty location in memory. An output is simple if the records are removed from the memory and written to empty locations on the disk. When all I/Os are simple, exactly one copy of each record exists at any time during the execution of an algorithm. The following lemma, proven by Aggarwal and Vitter, allows us to consider only simple I/Os when proving lower bounds.

Lemma 7 *For each computation that implements a permutation of records, there is a corresponding computation strategy involving only simple I/Os such that the total number of I/Os is no greater.* ∎

The basic scheme of the proof of Theorem 6 uses a potential-function argument. *Time* q is the time interval starting when the qth I/O completes and ending just before the $(q+1)$st I/O starts. We define a potential function Φ so that $\Phi(q)$ is the *potential* at time q. This potential measures how close the current record ordering is to the desired permutation order. Higher potentials indicate that the current ordering is closer to the desired permutation. We compute the initial and final potentials and bound the amount that the potential can increase in each I/O operation. The lower bound then follows.

To be more precise, we start with some definitions. For $i = 0, 1, \ldots, N/B - 1$, we define the ith *target group* to be the set of records that belong in block i according to the given BMMC permutation. We denote by $g_{\text{block}}(i, k, q)$ the number of records in the ith target group that are in block k on disk at time q, and $g_{\text{mem}}(i, q)$ denotes the number of records in the ith target group that are in memory at time q. We define the continuous function

$$f(x) = \begin{cases} x \lg x & \text{if } x > 0 , \\ 0 & \text{if } x = 0 , \end{cases}$$

and we define *togetherness functions*

$$G_{\text{block}}(k, q) = \sum_{i=0}^{N/B-1} f(g_{\text{block}}(i, k, q))$$

for each block k at time q and

$$G_{\text{mem}}(q) = \sum_{i=0}^{N/B-1} f(g_{\text{mem}}(i, q))$$

for memory at time q. Finally, we define the *potential* at time q, denoted $\Phi(q)$, as the sum of the togetherness functions:

$$\Phi(q) = G_{\text{mem}}(q) + \sum_{k=0}^{N/B-1} G_{\text{block}}(k, q) .$$

Aggarwal and Vitter embed the following lemmas in their lower-bound argument. The first lemma is based on the observation that the number of parallel I/Os needed is at least the total increase in potential over all parallel I/Os divided by the maximum increase in potential (denoted $\Delta\Phi_{\max}$) in any single parallel I/O.

Lemma 8 *Let $D = 1$, and consider any algorithm that uses t parallel I/Os to perform a permutation. Then $t = \Omega\left(\frac{\Phi(t) - \Phi(0)}{\Delta\Phi_{\max}}\right)$.* ∎

Lemma 9 *Let $D = 1$, and consider any permutation that can be performed with t parallel I/Os. Then $\Phi(t) = N \lg B$ and $\Delta\Phi_{\max} = O(B \lg(M/B))$. Therefore, any algorithm that performs a permutation uses $\Omega\left(\frac{N \lg B - \Phi(0)}{B \lg(M/B)}\right)$ parallel I/Os.* ∎

Observe that these lemmas imply lower bounds that are universal. No matter what permutation is being performed, the initial potential is $\Phi(0)$, the final potential is $\Phi(t)$, the increase in potential per parallel I/O is at most $\Delta\Phi_{\max}$, and so $\Omega\left(\frac{\Phi(t)-\Phi(0)}{\Delta\Phi_{\max}}\right)$ parallel I/Os are required.

We can now show a trivial lower bound for all non-identity BMMC permutations.

Lemma 10 *If $D = 1$, any algorithm that performs a BMMC permutation requires $\Omega(N/B)$ parallel I/Os whenever the permutation is not the identity permutation.*

Proof: Consider a BMMC permutation with characteristic matrix A and complement vector c. It is the identity permutation if and only if $A = I$ and $c = 0$, so we shall assume that either $A \neq I$ or $c \neq 0$.

A *fixed point* of the BMMC permutation is a source address x for which

$$A\,x \oplus c = x \; . \tag{6}$$

If a record's source address is not a fixed point, its source block must be read and its target block must be written. We shall show that for any non-identity BMMC permutation, at least $N/2$ addresses are not fixed points. Even if these records are clustered into as few source blocks as possible, then at least half the source blocks, or $N/2B$, must be read. The lemma then follows.

To show that at least $N/2$ addresses are not fixed points, we shall show that at most $N/2$ addresses are. Rewriting equation (6) as $(A \oplus I)x = c$, we see that we wish to bound the size of $\mathrm{Pre}(A \oplus I, c)$. If $c \notin \mathcal{R}(A \oplus I)$, then this size is 0. Otherwise, by Lemma 4, this size is $2^{n-\mathrm{rank}(A \oplus I)}$. If $A \neq I$, then $\mathrm{rank}(A \oplus I) \geq 1$, which implies that $|\mathrm{Pre}(A \oplus I, c)| \leq 2^{n-1} = N/2$. If $A = I$, then $A \oplus I$ is the 0 matrix, and the only vector in its range is 0. But $A = I$ and $c = 0$ yields the identity permutation, which we specifically disallow. ∎

Proof of Theorem 6

Recall that we shall prove Theorem 6 by proving the lower bound for the case in which $D = 1$; the general case follows by dividing by D. We work with characteristic matrix A and complement vector c. We assume that all I/Os are simple and transfer exactly B records, some possibly empty. Since all records start on disk and I/Os are simple, memory is initially empty.

We need to compute the initial potential in order to apply Lemma 9. The initial potential depends on the number of records that start in the same source block and are in the same target group. A record with source address $x = (x_0, x_1, \ldots, x_{n-1})$ is in source block k if and only if

$$k = x_{b..n-1} \; , \tag{7}$$

interpreting k as an $(n-b)$-bit binary number with the least signficant bit first. This record maps to target block i if and only if

$$\begin{aligned} i &= A_{b..n-1,0..n-1}\, x_{0..n-1} \oplus c_{b..n-1} \\ &= A_{b..n-1,0..b-1}\, x_{0..b-1} \oplus A_{b..n-1,b..n-1}\, x_{b..n-1} \oplus c_{b..n-1} \; , \end{aligned} \tag{8}$$

also interpreting i as an $(n-b)$-bit binary number. The following lemma gives the exact number of records that start in each source block and are in the same target group.

Lemma 11 *Let $r = \operatorname{rank} A_{b..n-1,0..b-1}$, and consider any source block k. There are exactly 2^r distinct target blocks that some record in source block k maps to, and for each such target block, exactly $B/2^r$ records in source block k map to it.*

Proof: For a given source block k, all source addresses fulfill condition (7), and so they map to target block numbers given by condition (8) but with $x_{b..n-1}$ fixed at k. The range of target block numbers is thus $\mathcal{R}(A_{b..n-1,0..b-1}) \oplus (A_{b..n-1,b..n-1}\, k \oplus c_{b..n-1})$ which, by Lemma 3, has cardinality 2^r.

Now we determine the set of source addresses in source block k that map to a particular target block i in $\mathcal{R}(A_{b..n-1,0..b-1}) \oplus (A_{b..n-1,b..n-1}\, k \oplus c_{b..n-1})$. Again fixing $x_{b..n-1} = k$ in condition (8) and exclusive-oring both sides by $A_{b..n-1,b..n-1}\, k \oplus c_{b..n-1}$, we see that this set is precisely $\operatorname{Pre}(A_{b..n-1,0..b-1}, i \oplus A_{b..n-1,b..n-1}\, k \oplus c_{b..n-1})$. By Lemma 4, this set has cardinality exactly 2^{b-r}, which equals $B/2^r$. ∎

We can interpret Lemma 11 as follows. Let $r = \operatorname{rank} A_{b..n-1,0..b-1}$, and consider a particular source block k. Then there are exactly 2^r target blocks i for which $g_{\text{block}}(i,k,0)$ is nonzero, and for each such nonzero target block, we have $g_{\text{block}}(i,k,0) = B/2^r$.

Now we can compute $\Phi(0)$. Since memory is initially empty, $g_{\text{mem}}(i,0) = 0$ for all blocks i, which implies that $G_{\text{mem}}(0) = 0$. We have

$$\begin{aligned}
\Phi(0) &= G_{\text{mem}}(0) + \sum_{k=0}^{N/B-1} G_{\text{block}}(k,0) \\
&= 0 + \sum_{k=0}^{N/B-1} \sum_{i=0}^{N/B-1} f(g_{\text{block}}(i,k,0)) \\
&= \sum_{k=0}^{N/B-1} 2^r \frac{B}{2^r} \lg \frac{B}{2^r} \qquad \text{(by Lemma 11)} \\
&= \frac{N}{B} B \lg \frac{B}{2^r} \\
&= N(\lg B - r)\ . \qquad (9)
\end{aligned}$$

Combining Lemmas 9 and 10 with equation (9), we get a lower bound of

$$\Omega\left(\frac{N}{B} + \frac{N \lg B - N(\lg B - r)}{B \lg(M/B)}\right) = \Omega\left(\frac{N}{B}\left(1 + \frac{\operatorname{rank} A_{b..n-1,0..b-1}}{\lg(M/B)}\right)\right)$$

parallel I/Os. Dividing through by D yields a lower bound of

$$\Omega\left(\frac{N}{BD}\left(1 + \frac{\operatorname{rank} A_{b..n-1,0..b-1}}{\lg(M/B)}\right)\right)\ ,$$

which completes the proof of Theorem 6.

4 MLD permutations

In this section, we describe how to perform any MLD permutation in only one pass. This section also discusses additional properties of MLD and MRC permutations and concludes with a discussion of MLD^{-1} permutations, which are permutations whose inverses are MLD permutations. Section 7 shows how to determine whether a given matrix characterizes an MLD permutation.

How the kernel condition implies a one-pass permutation

We shall show in three steps that the kernel condition implies that, for a given source memoryload, the source records are permuted into full target blocks spread evenly across the disks. To do so, we first need to define the notion of relative block number, as shown in Figure 2. For a given n-bit record address $x_{0..n-1}$, the *relative block number* of x is the $m - b$ bits $x_{b..m-1}$. The relative block number ranges from 0 to $M/B - 1$ and determines the number of a block within its memoryload. Recall that the memoryload number is the $n - m$ bits $x_{m..n-1}$. We shall prove that *for a given source memoryload*, the following properties hold:

1. Its records map to all M/B relative block numbers, and each relative block number has exactly B records mapping to it.

2. Records that map to the same relative block number map to the same target memoryload number as well.

The first two properties imply that the records of each source memoryload map to exactly M/B target blocks and that each such target block is full.

3. These M/B target blocks are distributed evenly among the disks, with M/BD mapping to each disk.

Given these properties, we can perform an MLD permutation in one pass. Like the other one-pass permutations described in [10], we allow the permutation to map records from one set of N/BD stripes (the "source portion" of the parallel disk system) to a different set of N/BD stripes (the "target portion"). One can think of addresses as relative to the beginning of the appropriate portion. In this way, we need not be concerned with overwriting source records before we get a chance to read them. Note that when we chain passes together, as in the BMMC algorithm of Section 6 and the BPC algorithm of [10], we can avoid allocating a new target portion in each pass by reversing the roles of the source and target portions between passes, and so the total disk space used is $2N$ records.

We perform an MLD permutation by processing source memoryload numbers from 0 to $N/M-1$. For each source memoryload, we first read into memory its M/BD consecutive stripes from the source portion. We then permute its records in memory, clustering them into M/B full target blocks that are distributed evenly among the disks. We then write out these target blocks using M/BD independent writes to the target portion. After processing all N/M source memoryloads, we have read each record from the source portion and written it to where it belongs in the target portion. Thus, we have performed the MLD permutation in one pass.

The following lemma gives an important consequence of the kernel condition.

Lemma 12 *If the matrix A characterizes an MLD permutation, then the submatrix λ has rank $m - b$.*

Proof: We shall prove that all rows of the leading $m \times m$ submatrix of A are linearly indepedent. The lemma then follows because λ is a subset of these rows.

Because A is nonsingular, the rank of its leftmost m columns (i.e., the submatrix $A_{0..n-1,0..m-1}$) is m. The row rank of any matrix equals the column rank, and so there are m linearly independent rows in $A_{0..n-1,0..m-1}$.

Since $\ker\lambda \subseteq \ker\mu$, Lemma 5 implies that $\text{row}\,\mu \subseteq \text{row}\,\lambda$. Thus, every row of μ is linearly dependent on some rows of λ and hence on some rows of the leading $m \times m$ submatrix of A. Since there are m linearly independent rows in $A_{0..n-1,0..m-1}$, all rows of the leading $m \times m$ submatrix must be linearly independent. ■

We now prove property 1.

Lemma 13 *The records of each source memoryload in an MLD permutation map to exactly M/B relative block numbers. Moreover, for a given source memoryload, each relative block number has exactly B records mapping to it.*

Proof: Let A characterize an MLD permutation with complement vector c. By Lemma 12, $\text{rank}\,A_{b..m-1,0..m-1} = m - b$. The target relative block number $y_{b..m-1}$ corresponding to a source address x is given by the equation

$$y_{b..m-1} = A_{b..m-1,0..m-1}\, x_{0..m-1} \oplus A_{b..m-1,m..n-1}\, x_{m..n-1} \oplus c_{b..m-1}\ . \tag{10}$$

The value of $x_{m..n-1}$ is fixed for a given source memoryload, and so the $(m-b)$-vector $A_{b..m-1,m..n-1}\, x_{m..n-1} \oplus c_{b..m-1}$ has the same value for all records. By Lemma 3, $y_{b..m-1}$ takes on $2^{\text{rank}\,A_{b..m-1,0..m-1}} = 2^{m-b} = M/B$ different values for the M different values of $x_{0..m-1}$. That is, the records of each source memoryload map to exactly M/B different relative block numbers.

Now consider some relative block number $y_{b..m-1}$ that some source address in a memoryload maps to. Using equation (10), the number of source addresses $x_{0..m-1}$ within that memoryload that map to $y_{b..m-1}$ is equal to $|\text{Pre}(A_{b..m-1,0..m-1}, y_{b..m-1} \oplus A_{b..m-1,m..n-1}\, x_{m..n-1} \oplus c_{b..m-1})|$. By Lemma 4, this number is equal to $2^{m-\text{rank}\,A_{b..m-1,0..m-1}} = 2^{m-(m-b)} = B$. ■

Property 2 follows directly from the kernel condition. Although we use kernel notation for its simplicity of expression, the following lemma shows that the kernel condition is equivalent to requiring that, for a given source memoryload, every record destined for a particular relative block number must also be destined for the same target memoryload.

Lemma 14 *Let K and L be matrices with q columns. Then $\ker K \subseteq \ker L$ if and only if for all q-vectors x and y, $Kx = Ky$ implies $Lx = Ly$.*

Proof: Suppose that $\ker K \subseteq \ker L$ and $Kx = Ky$. Then $K(x \oplus y) = 0$, which implies that $L(x \oplus y) = 0$, which in turn implies $Lx = Ly$.

Conversely, suppose that $Kx = Ky$ implies $Lx = Ly$ for all q-vectors x and y, and consider any q-vector $z \in \ker K$. We have $Kz = 0 = K \cdot 0$, which implies $Lz = L \cdot 0 = 0$. Thus, $z \in \ker L$. ■

For an MLD permutation, since $\ker\lambda \subseteq \ker\mu$, we apply Lemma 14 with $K = \lambda$ and $L = \mu$. Thus, any two source records x and y from the same source memoryload that are mapped to relative block number $\lambda\, x_{0..m-1}$ are also mapped to the same target memoryload $\mu\, x_{0..m-1}$.

Property 3 follows from property 1. Each source memoryload maps to relative block numbers $0, 1, \ldots, M/B - 1$. As Figure 2 shows, the number of the disk that a block resides on is encoded in the least significant d bits of its relative block number. The M/B relative block numbers, therefore, are evenly distributed among the D disks, with M/BD residing on each disk.

Thus, we have the following theorem.

Theorem 15 *Any MLD permutation can be performed in one pass with striped reads and independent writes.*

Proof: The above argument demonstrates that we can perform any MLD permutation in one pass with independent writes. Because a memoryload can be read with M/BD striped reads and the above method for performing MLD permutations reads full memoryloads, it uses striped reads. ∎

Additional properties of MLD and MRC permutations

We now examine some additional properties of MLD permutations. We shall use these properties primarily to combine matrix factors in the BMMC algorithm, thus reducing the number of passes. The first property bounds the rank of the submatrix μ as another consequence of the kernel condition.

Lemma 16 *In the characteristic matrix for an MLD permutation, the submatrix μ has rank at most $m - b$.*

Proof: By Lemma 5 and the kernel condition, $\operatorname{row}\mu \subseteq \operatorname{row}\lambda$, which in turn implies that $\dim(\operatorname{row}\mu) \le \dim(\operatorname{row}\lambda)$, where the dimension of a vector space is the size of any basis for it. Applying Lemma 12, we have that $\operatorname{rank}\mu \le \operatorname{rank}\lambda = m - b$. ∎

Thus, if the lower left $(n-m) \times m$ submatrix of a characteristic matrix has rank more than $m - b$, the matrix cannot characterize an MLD permutation.

Theorem 17 *Let the matrix Y characterize an MLD permutation, and let the matrix X characterize an MRC permutation. Then the matrix product $Y\,X$ characterizes an MLD permutation.*

Proof: Write the nonsingular matrix Y as

$$Y = \begin{array}{cc} \begin{array}{cc} m & n-m \end{array} & \\ \left[\begin{array}{c|c} \alpha & \beta \\ \hline \gamma & \delta \end{array}\right] & \begin{array}{l} m \\ n-m \end{array} \end{array} ,$$

where

$$\ker \alpha_{b..m-1,0..m-1} \subseteq \ker \gamma \ . \tag{11}$$

Write the nonsingular matrix X as

$$X = \begin{array}{cc} \begin{array}{cc} m & n-m \end{array} & \\ \left[\begin{array}{c|c} \phi & \sigma \\ \hline 0 & \nu \end{array}\right] & \begin{array}{l} m \\ n-m \end{array} \end{array} ,$$

where ϕ and ν are nonsingular. We now show that the product

$$\begin{array}{cc} & \begin{array}{cc} m & n-m \end{array} \\ YX = & \left[\begin{array}{c|c} \alpha\phi & \alpha\sigma \oplus \beta\nu \\ \hline \gamma\phi & \gamma\sigma \oplus \delta\nu \end{array}\right] \begin{array}{l} m \\ n-m \end{array} \end{array} .$$

characterizes an MLD permutation. Observe that the product $Y X$ is nonsingular because Y and X are each nonsingular.

We must also prove that the kernel condition (4) holds for the product, i.e., that $\ker(\alpha\phi)_{b..m-1,0..m-1} \subseteq \ker(\gamma\phi)$. For an $m \times m$ matrix τ, note that $\tau_{b..m-1,0..m-1} = I_{b..m-1,0..m-1}\,\tau$, where I is the usual $m \times m$ identity matrix. We have that $x \in \ker(\alpha\phi)_{b..m-1,0..m-1}$ implies $(I_{b..m-1,0..m-1}\alpha\phi)x = 0$ (taking $\alpha\phi$ as τ), which in turn implies that $\phi x \in \ker(I_{b..m-1,0..m-1}\alpha) = \ker \alpha_{b..m-1,0..m-1} \subseteq \ker\gamma$, by property (11). Thus, $\gamma\phi\, x = 0$, and so $x \in \ker(\gamma\phi)$. We conclude that $\ker(\alpha\phi)_{b..m-1,0..m-1} \subseteq \ker(\gamma\phi)$, which completes the proof. ■

Theorem 17 shows that the composition of an MLD permutation with an MRC permutation is an MLD permutation. Since we have seen how to perform MLD permutations, we can gain an intuitive understanding of why Theorem 17 holds. An MRC permutation permutes memoryload numbers, with records that start together within a source memoryload remaining together in a target memoryload. We perform an MLD permutation by reading in entire memoryloads. Thus, to perform the composition as an MLD permutation, we only have to remap the source memoryload numbers and adjust the in-memory permutations accordingly. Furthermore, as the following lemma shows, the composition of two MRC permutations is merely the composition of their memoryload mappings with the in-memory permutations adjusted accordingly.

Theorem 18 *The class of MRC permutations is closed under composition and inversion. That is, if a matrix A characterizes an MRC permutation, then so does the matrix A^{-1}, and if $A^{(1)}$ and $A^{(2)}$ characterize MRC permutations, then so does the product $A^{(1)}A^{(2)}$.*

Proof: We first show that MRC permutations are closed under inverse. Let the matrix

$$\begin{array}{cc} & \begin{array}{cc} m & n-m \end{array} \\ A = & \left[\begin{array}{c|c} \alpha & \beta \\ \hline 0 & \delta \end{array}\right] \begin{array}{l} m \\ n-m \end{array} \end{array}$$

characterize an MRC permutation, so that the leading submatrix α and trailing submatrix δ are nonsingular. The inverse of this matrix is

$$\begin{array}{cc} & \begin{array}{cc} m & n-m \end{array} \\ A^{-1} = & \left[\begin{array}{c|c} \alpha^{-1} & \alpha^{-1}\beta\delta^{-1} \\ \hline 0 & \delta^{-1} \end{array}\right] \begin{array}{l} m \\ n-m \end{array} \end{array} ,$$

where the leading $m \times m$ submatrix α^{-1} and trailing $(n-m)\times(n-m)$ submatrix δ^{-1} are nonsingular. Thus, the matrix A^{-1} characterizes an MRC permutation.

We now show that MRC permutations are closed under composition. Consider MRC characteristic matrices

$$\begin{array}{cc} & \begin{array}{cc} m & n-m \end{array} \\ A^{(1)} = & \left[\begin{array}{c|c} \alpha^{(1)} & \beta^{(1)} \\ \hline 0 & \delta^{(1)} \end{array}\right] \begin{array}{l} m \\ n-m \end{array} \end{array} ,$$

$$
\begin{array}{ccc}
 & \begin{array}{cc} m & n-m \end{array} & \\
A^{(2)} = & \left[\begin{array}{c|c} \alpha^{(2)} & \beta^{(2)} \\ \hline 0 & \delta^{(2)} \end{array}\right] & \begin{array}{l} m \\ n-m \end{array}
\end{array} ,
$$

where the submatrices $\alpha^{(1)}$, $\alpha^{(2)}$, $\delta^{(1)}$, and $\delta^{(2)}$ are nonsingular. Then their product is

$$
\begin{array}{ccc}
 & \begin{array}{cc} m & n-m \end{array} & \\
A^{(1)} A^{(2)} = & \left[\begin{array}{c|c} \alpha^{(1)} \alpha^{(2)} & \alpha^{(1)} \beta^{(2)} \oplus \beta^{(1)} \delta^{(2)} \\ \hline 0 & \delta^{(1)} \delta^{(2)} \end{array}\right] & \begin{array}{l} m \\ n-m \end{array}
\end{array} .
$$

Because $\alpha^{(1)}$ and $\alpha^{(2)}$ are nonsingular, so is their product $\alpha^{(1)} \alpha^{(2)}$. Similarly, the product $\delta^{(1)} \delta^{(2)}$ is nonsingular. The product $A^{(1)} A^{(2)}$, therefore, characterizes an MRC permutation. ∎

On the other hand, the composition of two MLD permutations is not necessarily an MLD permutation. We can see this fact in two ways. First, since we perform an MLD permutation by reading in entire memoryloads but writing blocks independently, it may not be possible to remap the source memoryload numbers. Second, consider the product of two matrices, each of which characterizes an MLD permutation. Although the rank of the lower left $(n-m) \times m$ submatrix of each factor is at most $m-b$, it may be the case that the rank of the lower left $(n-m) \times m$ submatrix of the product exceeds $m-b$. If so, then by Lemma 16, the product cannot characterize an MLD permutation.

Moreover, the composition of an MRC permutation with an MLD permutation (that is, reversing the order of the factors in Theorem 17) is not necessarily an MLD permutation. A simple example is the product

$$
\begin{array}{c}
\begin{array}{ccc} b & m-b & n-m \end{array} \\
\left[\begin{array}{c|c|c} 0 & I & 0 \\ \hline I & 0 & 0 \\ \hline 0 & 0 & I \end{array}\right] \\
\text{MRC}
\end{array}
\begin{array}{c}
\begin{array}{ccc} b & m-b & n-m \end{array} \\
\left[\begin{array}{c|c|c} I & 0 & 0 \\ \hline 0 & I & 0 \\ \hline 0 & I & I \end{array}\right] \\
\text{MLD}
\end{array}
=
\begin{array}{c}
\begin{array}{ccc} b & m-b & n-m \end{array} \\
\left[\begin{array}{c|c|c} 0 & I & 0 \\ \hline I & 0 & 0 \\ \hline 0 & I & I \end{array}\right] \\
\text{not MLD}
\end{array}
\begin{array}{l} b \\ m-b \\ n-m \end{array} ,
$$

with $b = m-b = n-m$. This product is not MLD since an m-vector with 0s in the first b positions and 1s in the last $m-b$ positions is a vector in $\ker \lambda$, but it is not a vector in $\ker \mu$.

Finally, we note that any MRC permutation is an MLD permutation. Observe that the lower left $(n-m) \times m$ submatrix of an MRC permutation must be 0, which implies that its kernel is the set of all m-vectors. No matter what $\ker \lambda$ is, it is a subset of this set.

Inverses of MLD permutations

The first BMMC algorithm we shall see works by factoring the BMMC characteristic matrix into matrices that characterize MRC and MLD permutations. In some settings, it may be easier to perform a permutation whose inverse is MLD (we call this class MLD^{-1}) than to perform an MLD permutation. We shall see an alternative way to factor BMMC characteristic matrices—into MRC and MLD^{-1} characteristic matrices—so that the resulting algorithm takes the same number of parallel I/Os as the original factorization into MRC and MLD permutations. In the remainder of this section, we examine the properties of MLD^{-1} permutations that enable this alternative factorization.

Striped writes may be valuable when redundant data is maintained on a parallel disk system to reduce the chance of data loss due to a failed device. Many common parallel-disk organizations fall under the heading of RAID (Redundant Array of Inexpensive Disks) [7, 17], which is organized into "levels" of redundancy. In RAID levels 3 and 4, an additional disk is added to the disk array to store redundancy. Each block of this *parity disk* contains the bitwise exclusive-or of the contents of the corresponding blocks of the other D data disks. If any one data disk fails, its contents are easily reconstructed from the contents of the $D-1$ remaining data disks and the parity disk. If an entire stripe is written to the disk array, it is easy to compute the corresponding parity information at the same time and write it to the parity disk in parallel with the data being written to the data disks. On the other hand, when less than a full stripe of data is being written to a given stripe of the disk array, updating the parity disk is harder. For each partial stripe being written, the old data and parity information must be read and the new data and parity information must be written. If k different stripes are being written, accessing the parity disk may become a severe bottleneck since k different blocks of the parity disk must be read and rewritten. Because an independent write may update individual blocks in several different stripes, in a RAID 3 or 4 organization, algorithms that use striped writes are preferable to those that use independent writes.

With this motivation, we begin our investigation of MLD^{-1} permutations with a property that pertains to all one-pass permutations.

Lemma 19 *If a permutation Π is a one-pass permutation, then its inverse permutation Π^{-1} is also a one-pass permutation. Moreover, if we perform Π using striped reads (respectively, writes), then we can perform Π^{-1} using striped writes (respectively, reads).*

Proof: Consider a one-pass algorithm to perform the permutation Π. It repeatedly reads a set of blocks, permutes their records in memory, and writes the records as full blocks. The one-pass algorithm reads and writes each record once. To perform the inverse permutation Π^{-1}, we invert each read-permute-write step in the algorithm for Π. In each step, we read the blocks that were written in the corresponding step for Π, we perform the inverse in-memory permutation, and we write the blocks that were read in the corresponding step for Π. Each record is still read and written once, and thus Π^{-1} is also a one-pass permutation. Note that if a read (respectively, write) for Π is striped, then the corresponding write (respectively, read) for Π^{-1} is also striped. ■

The following corollary follows directly from Theorem 15 and Lemma 19.

Corollary 20 *Any MLD^{-1} permutation can be performed in one pass with independent reads and striped writes.* ■

Our final property of MLD^{-1} permutations is analogous to Theorem 17.

Lemma 21 *Let the matrix X characterize an MRC permutation, and let the matrix Y characterize an MLD^{-1} permutation. Then the matrix product XY characterizes an MLD^{-1} permutation.*

Proof: Let $Z = XY$, so that $Z^{-1} = Y^{-1}X^{-1}$. Since the matrix Y characterizes an MLD^{-1} permutation, the matrix Y^{-1} characterizes an MLD permutation. By Theorem 18, the matrix X^{-1} characterizes an MRC permutation. By Theorem 17, therefore, the matrix Z^{-1} characterizes an MLD permutation. We conclude that the matrix Z characterizes an MLD^{-1} permutation. ■

5 Matrix-column operations

In this section, we classify forms of matrices which, as factors, have the effect of adding columns of other matrices to yield a product. We shall use matrices of this form in Section 6 to transform the characteristic matrix for any BMMC permutation into a characteristic matrix for an MRC permutation. This section shows the structure and useful properties of specific characteristic matrix forms we shall use.

Column additions

We define a *column-addition* matrix as a matrix Q such that the product $A' = AQ$ is a modified form of A in which specified columns of A have been added into others. Denoting the kth column of A by A_k, we define the matrix $Q = (q_{ij})$ by

$$q_{ij} = \begin{cases} 1 & \text{if } i = j \ , \\ 1 & \text{if column } A_i \text{ is added into column } A_j \ , \\ 0 & \text{otherwise .} \end{cases}$$

For example,

$$\underbrace{\begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}}_{Q} = \left[\begin{array}{c|c|c|c} A_0 & A_0 \oplus A_1 \oplus A_3 & A_0 \oplus A_2 & A_3 \end{array}\right] = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{A'} .$$

Column-addition matrices are also subject to a *dependency restriction* that if column i is added into column j, then column j cannot be added into any other column. That is, if $q_{ij} = 1$, then $q_{jk} = 0$ for all $k \neq j$. The following lemma shows that any column-addition matrix is the product of two nonsingular matrices, and so any column-addition matrix is also nonsingular.

Lemma 22 *Any column-addition matrix is nonsingular.*

Proof: We shall prove by induction on the matrix size n that any column-addition matrix Q is the product of two nonsingular matrices L and U. Thus, the matrix Q is also nonsingular.

For the basis, when $n = 2$, the only column-addition matrices are $\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$, $\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$, and $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$. Since each of these matrices is nonsingular, each of them is the product of itself and the identity matrix.

For the inductive step, we assume that every $(n-1) \times (n-1)$ column-addition matrix is the product of two nonsingular matrices. We partition an arbitrary $n \times n$ column-addition matrix Q as

$$Q = \begin{array}{cc} \begin{array}{cc} 1 & n-1 \end{array} & \\ \left[\begin{array}{c|c} 1 & \psi \\ \hline \theta & \chi \end{array}\right] & \begin{array}{l} 1 \\ n-1 \end{array} \end{array} .$$

The trailing $(n-1) \times (n-1)$ submatrix χ is a column-addition matrix because all of its diagonal elements are 1s and, as a submatrix of Q, it obeys the dependency restriction. By our inductive hypothesis, therefore, the submatrix χ is the product of two $(n-1) \times (n-1)$ nonsingular matrices, say ν and η. By the dependency restriction, if there are any 1s in θ, then there cannot be any 1s in ψ. Therefore, either ψ or θ is a zero submatrix, and consequently we can factor Q as

$$Q = \underbrace{\begin{array}{c} \begin{array}{cc} 1 & n-1 \end{array} \\ \left[\begin{array}{c|c} 1 & 0 \\ \hline \theta & \nu \end{array}\right] \end{array}}_{L} \; \underbrace{\begin{array}{c} \begin{array}{cc} 1 & n-1 \end{array} \\ \left[\begin{array}{c|c} 1 & \psi \\ \hline 0 & \eta \end{array}\right] \end{array}}_{U} \begin{array}{l} 1 \\ n-1 \end{array} .$$

The rightmost $n-1$ columns of L are linearly independent since the submatrix ν is nonsingular and the upper right $1 \times (n-1)$ submatrix is 0. The leftmost column is linearly independent of the rightmost $n-1$ columns since its top entry is 1 and the top entry of each of the rightmost $n-1$ columns is 0. Thus, L is nonsingular. Similarly, because the submatrix η is nonsingular and the lower left $(n-1) \times 1$ submatrix of U is 0, the matrix U is nonsingular. Thus, any column-addition matrix is the product of two nonsingular matrices, and therefore is also nonsingular. ∎

In fact, the factors L and U in the proof of Lemma 22 are unit lower-triangular and unit upper-triangular matrices, respectively. Thus, we can factor the example above as

$$Q = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} = L\,U .$$

Partitioning the matrix

In Section 6, we shall factor nonsingular matrices into column-addition matrices and matrices that characterize MRC permutations. These matrices will be of various block forms, and to classify these forms, we use the following block representation. We partition a matrix into three sections: left, middle, and right. The *left section* includes the leftmost b columns, the *middle section* includes the middle $m-b$ columns and the *right section* includes the rightmost $n-m$ columns. When the form of a particular submatrix is known, we label that block accordingly. Otherwise, we place an asterisk (*) in blocks whose contents are not of any particular form.

For column-addition operations, the characteristic matrix has the following form. Every entry on the diagonal is 1. We place an asterisk in each submatrix that contains any non-diagonal 1s as defined by the operation. Returning to the example above, if $b = 1$ and $m = 2$, the form of Q is

$$Q = \begin{array}{c} \begin{array}{ccc} b=1 & m-b=1 & n-m=2 \end{array} \\ \left[\begin{array}{c|c|c} I & * & * \\ \hline 0 & I & 0 \\ \hline 0 & * & I \end{array}\right] \end{array} \begin{array}{l} b=1 \\ m-b=1 \\ n-m=2 \end{array} .$$

We define several column-addition operations and MRC permutations by the form of their characteristic matrices. Each of these forms is nonsingular and characterizes a one-pass permutation. We shall show that the inverse of each of these one-pass permutations falls into a specific class of one-pass permutations.

Trailer matrix form

In Section 6, we shall need to transform a nonsingular matrix into one that has a nonsingular trailing $(n-m) \times (n-m)$ submatrix. We shall create the nonsingular trailing submatrix by adding some columns from the left and middle sections to the right section. We define a *trailer matrix* as a column-addition matrix that adds some columns from the left and middle sections into the columns of the right section. The matrix T for this operation is of the form

$$T = \begin{array}{c} \begin{array}{ccc} b & m-b & n-m \end{array} \\ \left[\begin{array}{c|c|c} I & 0 & * \\ \hline 0 & I & * \\ \hline 0 & 0 & I \end{array}\right] \end{array} \begin{array}{l} \\ b \\ m-b \\ n-m \end{array} .$$

The trailer matrix form characterizes an MRC permutation.

Reducer matrix form

Once we have a matrix with a nonsingular trailing submatrix, we need an operation that puts the matrix into "reduced form." (We shall define reduced form precisely in Section 6.) We convert a matrix into reduced form by adding columns from the left and middle sections into other columns in the left and middle sections while respecting the dependency restriction. Thus, a *reducer matrix* R is a column-addition matrix of the form

$$R = \begin{array}{c} \begin{array}{ccc} b & m-b & n-m \end{array} \\ \left[\begin{array}{c|c|c} * & * & 0 \\ \hline * & * & 0 \\ \hline 0 & 0 & I \end{array}\right] \end{array} \begin{array}{l} \\ b \\ m-b \\ n-m \end{array} .$$

Since the dependency restriction is obeyed, the leading $m \times m$ submatrix of R is nonsingular. Thus, the matrix R characterizes an MRC permutation.

We can multiply the forms T and R to create another matrix form that also characterizes a one-pass permutation. The product $T\,R$ results in a matrix of the form

$$P = \begin{array}{c} \begin{array}{ccc} b & m-b & n-m \end{array} \\ \left[\begin{array}{c|c|c} * & * & * \\ \hline * & * & * \\ \hline 0 & 0 & I \end{array}\right] \end{array} \begin{array}{l} \\ b \\ m-b \\ n-m \end{array} .$$

Since both of the matrix forms T and R characterize MRC permutations, by Theorem 18, so does the matrix form P and its inverse.

Swapper matrix form

We shall also need to transform the columns in the lower left and lower middle submatrices into columns of zeros. To do so, we must move the nonzero columns in the lower left submatrix into the lower middle submatrix positions by swapping at most $m-b$ columns at a time from the left section with those in the middle section. Thus, the swap operation is a permutation of the leftmost m columns. A *swapper matrix* is of the form

$$S = \begin{array}{c} \begin{array}{cc} m & n-m \end{array} \\ \left[\begin{array}{c|c} \text{permutation} & 0 \\ \hline 0 & I \end{array}\right] \end{array} \begin{array}{l} \\ m \\ n-m \end{array}$$

so that the leading $m \times m$ submatrix is a permutation matrix, which dictates the permutation of the leftmost m columns. The matrix form S characterizes an MRC permutation and, by Theorem 18, so does its inverse.

Erasure matrix form

The last operation used in Section 6 is an erasure operation to zero out columns in the lower middle submatrix. To perform this operation, we add columns from the right section into columns in the middle section. Thus, an *erasure* matrix form is defined as

$$E = \begin{array}{c} \begin{array}{ccc} b & m-b & n-m \end{array} \\ \left[\begin{array}{c|c|c} I & 0 & 0 \\ \hline 0 & I & 0 \\ \hline 0 & * & I \end{array}\right] \end{array} \begin{array}{l} \\ b \\ m-b \\ n-m \end{array} .$$

This matrix form characterizes an MLD permutation because the kernel of $E_{b..m-1,0..m-1}$ includes only those m-vectors x with $x_{b..m-1} = 0$, and each such vector is also in the kernel of $E_{m..n-1,0..m-1}$. Moreover, observe that any matrix of this form is its own inverse. Consequently, the inverse of such a matrix characterizes an MLD permutation.

6 An asymptotically optimal BMMC algorithm

In this section, we present an algorithm to perform any BMMC permutation by factoring its characteristic matrix into matrices which characterize one-pass permutations. We assume that the BMMC permutation is given by an $n \times n$ characteristic matrix A and a complement vector c of length n. We show that the number of parallel I/Os to perform any BMMC permutation is at most $\frac{2N}{BD}\left(\left\lceil \frac{\operatorname{rank}\gamma}{\lg(M/B)} \right\rceil + 2\right)$ parallel I/Os, where γ is the submatrix $A_{b..n-1,0..b-1}$, which appears in the lower bound given by Theorem 6.

Our strategy is to factor the matrix A into a product of matrices, each of which characterizes an MRC or MLD permutation. For now, we ignore the complement vector c. According to Corollary 2, we read the factors right to left to determine the order in which to perform the permutations.

To obtain the factorization for A, we multiply A by matrices of the forms described in Section 5. By applying these matrix-column operations, we transform the matrix A into a matrix F that characterizes an MRC permutation. Multiplying F by the inverse of each of the matrix-column factors yields the factorization.

Creating a nonsingular trailing submatrix

We start to transform the characteristic matrix A by creating a nonsingular matrix $A^{(1)}$ which has a nonsingular trailing $(n-m)\times(n-m)$ submatrix. We represent the matrix A as

$$A = \begin{array}{cc} \begin{array}{cc} m & n-m \end{array} & \\ \left[\begin{array}{c|c} \alpha & \beta \\ \hline \phi & \delta \end{array}\right] & \begin{array}{l} m \\ n-m \end{array} \end{array} .$$

Our algorithm depends on the structure of ϕ rather than γ. The following lemma allows us to consider $\operatorname{rank}\phi$ instead of $\operatorname{rank}\gamma$ with only a minor difference.

Lemma 23 *For any matrix A,*

$$\operatorname{rank} A_{b..n-1,0..b-1} - \lg(M/B) \le \operatorname{rank} A_{m..n-1,0..m-1} \le \operatorname{rank} A_{b..n-1,0..b-1} + \lg(M/B) .$$

Proof: Because the rank of a submatrix is the maximum number of linearly independent rows or columns, we have

$$\operatorname{rank} A_{m..n-1,0..b-1} \le \operatorname{rank} A_{b..n-1,0..b-1} \le \operatorname{rank} A_{m..n-1,0..b-1} + \lg(M/B) , \tag{12}$$
$$\operatorname{rank} A_{m..n-1,0..b-1} \le \operatorname{rank} A_{m..n-1,0..m-1} \le \operatorname{rank} A_{m..n-1,0..b-1} + \lg(M/B) . \tag{13}$$

Subtracting $\lg(M/B)$ from the right-hand inequality of (12) and combining the result with the left-hand inequality of (13) yields

$$\operatorname{rank} A_{b..n-1,0..b-1} - \lg(M/B) \le \operatorname{rank} A_{m..n-1,0..b-1} \le \operatorname{rank} A_{m..n-1,0..m-1} . \tag{14}$$

Adding $\lg(M/B)$ to the left-hand inequality of (12) and combining the result with the right-hand inequality of (13) yields

$$\operatorname{rank} A_{m..n-1,0..m-1} \le \operatorname{rank} A_{m..n-1,0..b-1} + \lg(M/B) \le \operatorname{rank} A_{b..n-1,0..b-1} + \lg(M/B) . \tag{15}$$

Combining inequalities (14) and (15) proves the lemma. ■

By Lemma 23, therefore,

$$\operatorname{rank}\phi \le \operatorname{rank}\gamma + \lg(M/B) . \tag{16}$$

We shall use this fact later in the analysis of the algorithm to express the bound in terms of $\operatorname{rank}\gamma$.

We make the trailing $(n-m)\times(n-m)$ submatrix nonsingular by adding columns in ϕ into those in δ. Consider δ as a set of $n-m$ columns and ϕ as a set of m columns. Because A is nonsingular, the submatrix of A consisting of the bottom $n-m$ rows (i.e., submatrices ϕ and δ) has rank $n-m$. Hence, there exists a set of $n-m$ linearly independent columns in the bottom $n-m$ rows of A. We use the method described in Section 2 to determine a maximal set V of $\operatorname{rank}\delta$ linearly independent columns in δ and a set W of $n-m-\operatorname{rank}\delta$ columns in ϕ that, along with V, comprise a set of $n-m$ linearly independent columns. Denoting by $\overline{V}$ the $n-m-\operatorname{rank}\delta$ columns of δ not in V, we make the trailing submatrix of A nonsingular by pairing up columns of W with columns of $\overline{V}$ and adding each column in W into its corresponding column in $\overline{V}$. Because V is a maximal set of linearly independent columns in δ, the columns of $\overline{V}$ depend only on columns of V and not on columns of W. Adding a column of W into a column of $\overline{V}$ must produce a column that is linearly independent of those in V. Because each column of $\overline{V}$ has a different column of W added in, the resulting columns are linearly independent of each other, too.

We must express the above transformation as a column-addition operation. Although we focused above on adding columns of ϕ to columns of δ, column-addition operations add entire columns, and so we must also add the corresponding columns of α to the corresponding columns of β. Since we add columns from the leftmost m columns of A to the rightmost $n-m$ columns, the characteristic matrix of this operation has the trailer matrix form T described in Section 5. The matrix product is now

$$A^{(1)} = AT = \begin{array}{cc} \begin{array}{cc} m & n-m \end{array} & \\ \left[\begin{array}{c|c} \alpha & \widehat{\beta} \\ \hline \phi & \widehat{\delta} \end{array}\right] & \begin{array}{l} m \\ n-m \end{array} \end{array} ,$$

where $\widehat{\delta}$ is nonsingular. Since the matrices A and T are nonsingular, the matrix $A^{(1)}$ is nonsingular.

Transforming the matrix into reduced form

The next step is to transform the matrix $A^{(1)}$ into reduced form. For our purposes, a matrix is in *reduced form* when there are $\operatorname{rank}\phi$ linearly independent columns and $m - \operatorname{rank}\phi$ columns of zeros in the lower left $(n-m)\times m$ submatrix and the trailing $(n-m)\times(n-m)$ submatrix is nonsingular. Once again, we use the method of Section 2 to determine a set U that indexes $\operatorname{rank}\phi$ linearly independent columns of ϕ. To perform the reduction, we determine for each linearly dependent column ϕ_j a set of column indices $U_j \subseteq U$ such that $\phi_j = \oplus_{k\in U_j}\phi_k$. Adding the set of columns of ϕ indexed by U_j into ϕ_j zeros it out. We add linearly independent columns from the left and middle sections into the linearly dependent columns of these sections. Since we never add a linearly dependent column into any other column and there are no column additions into the linearly independent columns, we respect the dependency restriction. The matrix R that reduces the matrix $A^{(1)}$ has the reducer matrix form described in Section 5. Thus, the matrix product $T\,R$ is of the form P also described in Section 5, and it characterizes an MRC permutation. We now have the product

$$A^{(2)} = A^{(1)}R = AT\,R = AP = \begin{array}{cc} \begin{array}{cc} m & n-m \end{array} & \\ \left[\begin{array}{c|c} \widehat{\alpha} & \widehat{\beta} \\ \hline \widehat{\phi} & \widehat{\delta} \end{array}\right] & \begin{array}{l} m \\ n-m \end{array} \end{array} ,$$

with a nonsingular trailing submatrix $\widehat{\delta}$ and a lower left submatrix $\widehat{\phi}$ in reduced form. Since A and P are nonsingular, the matrix $A^{(2)}$ is also nonsingular.

Zeroing out the lower left submatrix

Our eventual goal is to transform the original matrix A into a matrix F that characterizes an MRC permutation. At this point, the matrix A has been transformed into the nonsingular matrix $A^{(2)}$. Thus, our final task is to multiply $A^{(2)}$ by a series of matrices that transform the $\operatorname{rank}\phi$ nonzero columns in the lower left $(n-m)\times m$ submatrix $\widehat{\phi}$ into columns of zeros. We do so by multiplying

$A^{(2)}$ by matrices of the swapper and erasure matrix forms described in Section 5. Let us further partition the leftmost m columns of $A^{(2)}$ into the leftmost b columns and the middle $m-b$ columns:

$$A^{(2)} = \begin{array}{c} \begin{array}{ccc} b & m-b & n-m \end{array} \\ \left[\begin{array}{c|c|c} \widehat{\alpha}' & \widehat{\alpha}'' & \widehat{\beta} \\ \hline \widehat{\phi}' & \widehat{\phi}'' & \widehat{\delta} \end{array}\right] \end{array} \begin{array}{l} m \\ n-m \end{array} .$$

Our strategy is to repeatedly use swapper matrix forms to move at most $m-b$ columns from the left section into the middle section and then zero out those columns using erasure matrix forms.

We begin by swapping $m-b-\operatorname{rank}\widehat{\phi}''$ columns from $\widehat{\phi}'$ with the zero columns of $\widehat{\phi}''$. Multiplying the matrix $A^{(2)}$ by a nonsingular matrix S_1 of the swapper matrix form described in Section 5, we swap at most $m-b$ columns from the left section with the appropriate columns in the middle section. After performing the swap operation on the matrix $A^{(2)}$, the lower left submatrix has $m-b-\operatorname{rank}\widehat{\phi}''$ additional zero columns and the lower middle submatrix has full rank. The above discussion assumes that $\operatorname{rank}\widehat{\phi}' \geq m-b-\operatorname{rank}\widehat{\phi}''$; if the opposite holds, we swap $\operatorname{rank}\widehat{\phi}'$ columns and the lower left submatrix becomes all zeros.

Our next step is to transform the $m-b$ columns in the lower middle submatrix into columns of zeros. Since the nonsingular trailing $(n-m)\times(n-m)$ submatrix $\widehat{\delta}$ forms a basis for the columns of the lower $n-m$ rows of matrix $A^{(2)}$, we zero out each nonzero column in the lower middle submatrix by adding columns of $\widehat{\delta}$. Since we add columns from the rightmost $n-m$ columns into the middle $m-b$ columns, we perform the matrix-column operation characterized by a nonsingular matrix E_1 of the erasure matrix form described in Section 5. After multiplying by the erasure matrix E_1, the original matrix is transformed into a nonsingular matrix

$$A^{(3)} = A\,P\,S_1\,E_1 ,$$

which has zero columns in the lower middle $(n-m)\times(m-b)$ submatrix and possibly some more nonzero columns in the lower left $(n-m)\times b$ submatrix.

If there are still nonzero columns in the lower left $(n-m)\times b$ submatrix of the matrix $A^{(3)}$, then those columns must also be swapped into the lower middle $(n-m)\times(m-b)$ submatrix by a swapper matrix and transformed into zero columns by an erasure matrix. We repeatedly swap in up to $m-b$ nonzero columns of the lower left submatrix into the lower middle submatrix. Each time we perform a swap operation, we multiply the current product by a matrix S_i of the swapper matrix form. Note that we swap entire columns here, not just the portions in the lower submatrices. After we perform each matrix-column operation S_i, we zero out the lower middle submatrix by multiplying the current product by a matrix E_i of the erasure matrix form.

After repeatedly swapping and erasing each of the nonzero columns in the lower left $(n-m)\times m$ submatrix, the lower left submatrix will contain only zero columns. This matrix is the matrix F mentioned at the beginning of this section. Since the matrix $A^{(2)}$ is in reduced form, there are at most $\operatorname{rank}\phi$ columns in the submatrix $\widehat{\phi}$ that need to be transformed into zero columns. Thus, at most

$$g = \left\lceil \frac{\operatorname{rank}\phi}{m-b} \right\rceil \tag{17}$$

pairs of swap and erasure operations transform all the columns in the lower left $(n-m)\times m$ submatrix into zero columns. Since each matrix-column operation that we performed on the original matrix A to transform it into F is nonsingular, the resulting matrix product

$$F = A\,P\,S_1\,E_1\,S_2\,E_2\cdots S_g\,E_g$$

is a nonsingular matrix that characterizes an MRC permutation. Multiplying both sides by the inverses of the factors yields the desired factorization of A:

$$A = F\, E_g^{-1}\, S_g^{-1}\, E_{g-1}^{-1}\, S_{g-1}^{-1} \cdots E_1^{-1}\, S_1^{-1} P^{-1} \; . \tag{18}$$

Analysis

We now apply several properties that we have gathered to complete the analysis of our BMMC permutation factoring method.

Theorem 24 *We can perform any BMMC permutation with characteristic matrix A and complement vector c in at most*

$$\frac{2N}{BD}\left(\left\lceil \frac{\operatorname{rank}\gamma}{\lg(M/B)} \right\rceil + 2\right)$$

parallel I/Os, where $\gamma = A_{b..n-1,0..b-1}$, using striped reads, independent writes, and $2N$ records of disk space.

Proof: Ignore the complement vector c for the moment. In the factorization (18) of A, both factors S_1^{-1} and P^{-1} characterize MRC permutations. By Theorem 18, therefore, so does the product $S_1^{-1}\, P^{-1}$. As we saw in Section 5, each factor E_i^{-1} characterizes an MLD permutation. Applying Theorem 17, each grouping of factors $E_1^{-1}\, S_1^{-1}\, P^{-1}$ and $E_i^{-1}\, S_i^{-1}$, for $i = 2, 3, \ldots, g$, characterizes an MLD permutation. By Theorem 15, and adding in one more pass for the MRC permutation characterized by F, we can perform A with $g+1$ passes.

If the complement vector c is nonzero, we include it as part of the MRC permutation characterized by the leftmost factor F. See [10] for details.

Regardless of the complement vector, therefore, we can perform the BMMC permutation with $g+1$ passes. Combining equation (17) and inequality (16), we obtain a bound of

$$\begin{aligned} g+1 &= \left\lceil \frac{\operatorname{rank}\phi}{\lg(M/B)} \right\rceil + 1 \\ &\leq \left\lceil \frac{\operatorname{rank}\gamma + \lg(M/B)}{\lg(M/B)} \right\rceil + 1 \\ &= \left\lceil \frac{\operatorname{rank}\gamma}{\lg(M/B)} \right\rceil + 2 \end{aligned}$$

passes for a total of at most

$$\frac{2N}{BD}\left(\left\lceil \frac{\operatorname{rank}\gamma}{\lg(M/B)} \right\rceil + 2\right)$$

parallel I/Os.

Because each factor characterizes either an MRC permutation (performed with striped reads and writes) or an MLD permutation (performed with striped reads and independent writes), and striped I/O is a special case of independent I/O, the method as a whole uses striped reads and independent writes. The method uses $2N$ records of disk space by reversing the roles of the source and target portions between passes. That is, the target portion written to in one pass becomes the source portion read from in the next pass. ■

Performing BMMC permutations with striped writes

Here we describe another way to compose the factors from the product of equation (18) into $g+1$ factors, such that each factor characterizes either an MRC or MLD^{-1} permutation. Both MRC and MLD^{-1} permutations can be performed with striped writes. As mentioned in Section 4, striped writes may have advantages in parallel disk systems organized as RAID levels 3 or 4.

In our alternative factorization, we start by noting that because any erasure matrix is its own inverse, not only does each factor E_i^{-1} in the factorization (18) characterize an MLD permutation, it also characterizes an MLD^{-1} permutation. Instead of grouping the factors $E_1^{-1}\, S_1^{-1}\, P^{-1}$ and $E_i^{-1}\, S_i^{-1}$ for $i = 2, 3, \ldots, g$, here we group by $F\, E_g^{-1}$ and $S_i^{-1}\, E_{i-1}^{-1}$, for $i = 2, 3, \ldots, g$. By Lemma 21, each such grouping of factors characterizes an MLD^{-1} permutation. Thus, the resulting factorization of A has g MLD^{-1} factors and the MRC product $S_1^{-1}\, P^{-1}$. Thus, this alternative factorization of A has the same number of one-pass factors as our previous grouping, but it uses only MRC and MLD^{-1} permutations as its factors. Thus, we have proven the following.

Theorem 25 *We can perform any BMMC permutation with characteristic matrix A and complement vector c in at most*

$$\frac{2N}{BD}\left(\left\lceil \frac{\operatorname{rank}\gamma}{\lg(M/B)} \right\rceil + 2\right)$$

parallel I/Os, where $\gamma = A_{b..n-1,0..b-1}$, using independent reads, striped writes, and $2N$ records of disk space. ∎

7 Detecting BMMC permutations at run time

In practice, we wish to run the BMMC algorithm of Section 6 whenever possible to reap the savings over having run the more costly algorithm for general permutations. For that matter, we wish to run even faster algorithms for any of the special cases of BMMC permutations (MRC, MLD, MLD^{-1}, or block BMMC [10]) whenever possible as well. We must know the characteristic matrix A and complement vector c, however, to run any of these algorithms. If A and c are specified in the source code, before running the algorithm we only need to check that A is of the correct form, e.g., that it is nonsingular for a BMMC permutation, of the MLD or MRC form, etc. Later in this section, we show how to check the kernel condition for MLD permutations. If instead the permutation is given by a vector of N target addresses, we can detect at run time whether it is a BMMC permutation by the following procedure:

1. Check that N is a power of 2.

2. Form a candidate characteristic matrix A and complement vector c such that if the permutation is BMMC, then A and c must be the correct characterizations. This section shows how to do so with only $\left\lceil \frac{\lg(N/B)+1}{D} \right\rceil$ parallel reads.

3. Check that the characteristic matrix is of the correct form. That is, check that it is nonsingular, which is easily done by the method of Section 2. If further structure is desired, e.g., MRC or MLD forms, check further for the desired form.

4. Verify that all N target addresses are described by the candidate characteristic matrix and complement vector. If for any source address x and its corresponding target address y we have $y \neq A\,x \oplus c$, the permutation is not BMMC and we can terminate verification. If $y = A\,x \oplus c$ for all N source-target pairs, the permutation is BMMC. Verification uses at most N/BD parallel reads, since we need to read each target address at most once. Source addresses are generated implicitly, and so they do not entail any I/O cost.

The total number of parallel I/Os is at most

$$\frac{N}{BD} + \left\lceil \frac{\lg(N/B) + 1}{D} \right\rceil ,$$

all of which are reads, and it is usually far fewer when the permutation turns out not to be BMMC.

One benefit of run-time BMMC detection is that the programmer might not realize that the desired permutation is BMMC. For example, as noted in Section 1, the standard binary reflected Gray code and its inverse are both MRC permutations. Yet the programmer might not know to call a special MRC or BMMC routine. Even if the system provides an entry point to perform the standard Gray code permutation and this routine invokes the MRC algorithm, variations on the standard Gray code may foil this approach. For example, a standard Gray code with all bits permuted the same (i.e., a characteristic matrix of $\Pi\,G$, where Π is a permutation matrix and G is the MRC matrix that characterizes the standard Gray code) is BMMC but not necessarily MRC. It might not be obvious enough that the permutation characterized by $\Pi\,G$ is BMMC for the programmer to invoke the BMMC algorithm explicitly.

Forming the candidate characteristic matrix and complement vector

The method for forming the candidate characteristic matrix A and candidate complement vector c is based on two observations. First, if the permutation is BMMC, then the complement vector c must be the target address corresponding to source address 0. This relationship holds because $x = 0$ and $y = A\,x \oplus c$ imply that $y = c$.

The second observation is as follows. Consider a source address $x = (x_0, x_1, \ldots, x_{n-1})$, and suppose that bit position k holds a 1, i.e., $x_k = 1$. Let us denote the jth column for matrix A by A_j. Also, let S_k denote the set of bit positions in x other than k that hold a 1: $S_k = \{j : j \neq k \text{ and } x_j = 1\}$. If $y = A\,x \oplus c$, then we have

$$y = \left(\bigoplus_{j \in S_k} A_j \right) \oplus A_k \oplus c , \tag{19}$$

since only the bit positions j for which $x_j = 1$ contribute a column of A to the sum of columns that forms the matrix-vector product. If we know the target address y, the complement vector c and the columns A_j for all $j \neq k$, we can rewrite equation (19) to yield the kth column of A:

$$A_k = y \oplus \left(\bigoplus_{j \in S_k} A_j \right) \oplus c . \tag{20}$$

We shall compute the complement vector c first and then the columns of the characteristic matrix A one at a time, from A_0 up to A_{n-1}. When computing A_k, we will have already computed

$A_0, A_1, \ldots, A_{k-1}$, and these will be the only columns we need in order to apply equation (20). In other words, $S_k \subseteq \{0, 1, \ldots, k-1\}$. Recall that as Figure 2 shows, the lower b bits of a record's address give the record's offset within its block, the middle d bits give the disk number, and the upper $s = n - (b + d)$ bits give the stripe number.

From equation (20), it would be easy to compute A_k if S_k were empty. The set S_k is empty if the source address is a unit vector, with its only 1 in position k. If we look at these addresses, however, we find that the target addresses for a disproportionate number—all but d of them—reside on disk $\mathcal{D}_0$. The block whose disk and stripe fields are all zero contains b such addresses, so they can be fetched in one disk read. A problem arises for the s source addresses with one 1 in the stripe field: their target addresses all reside on different blocks of disk $\mathcal{D}_0$. If we use this method, each of these blocks must be fetched in a separate read. The total number of parallel reads to fetch all the target addresses corresponding to all unit-vector source addresses is $s + 1 = \lg(N/BD) + 1$.

To achieve only $\left\lceil \frac{\lg(N/B)+1}{D} \right\rceil$ parallel reads, each read fetches one block from each of the D disks. The first parallel read determines the complement vector, the first $b + d$ columns, and the next $D - d - 1$ columns. Each subsequent read determines another D columns, until all n columns have been determined.

In the first parallel read, we do the same as above for the first $b + d$ bits. That is, we fetch blocks containing target addresses whose corresponding source addresses are unit vectors with one 1 in the first $b + d$ positions. As before, b of them are in the same block on disk $\mathcal{D}_0$. This block also contains address 0, which we need to compute the complement vector. The remaining d are in stripe number 0 of disks $\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_4, \mathcal{D}_8, \ldots, \mathcal{D}_{D/2}$. Having fetched the corresponding target addresses, we have all the information we need to compute the complement vector c and columns $A_0, A_1, \ldots, A_{b+d-1}$.

The columns we have yet to compute correspond to bit positions in the stripe field. If we were to compute these columns in the same fashion as the first $b + d$, we would again encounter the problem that all the blocks we need to read are on disk $\mathcal{D}_0$. In the first parallel read, the only unused disks remaining are those whose numbers are not a power of 2 ($\mathcal{D}_3, \mathcal{D}_5, \mathcal{D}_6, \mathcal{D}_7, \mathcal{D}_9, \ldots$). The key observation is that we have already computed all d columns corresponding to the disk field, and we can thus apply equation (20). For example, let us compute column A_{b+d}, which corresponds to the first bit of the stripe number. We read stripe 1 on disk $\mathcal{D}_3$ and find the first target address y in this block. Disk number 3 corresponds to the first two disk-number columns, A_b and A_{b+1}. Applying equation (20) with $S_{b+d} = \{b, b+1\}$, we compute $A_{b+d} = y \oplus A_b \oplus A_{b+1} \oplus c$. The next column we compute is A_{b+d+1}. Reading the block at stripe 2 on disk $\mathcal{D}_5$, we fetch a target address y and then compute $A_{b+d+1} = y \oplus A_b \oplus A_{b+2} \oplus c$. Continuing on in this fashion, we compute a total of $D - d - 1$ stripe-bit columns from the first parallel read.

The remaining parallel reads compute the remaining stripe-bit columns. We follow the stripe-bit pattern of the first read, but we use all disks, not just those whose disk numbers are not powers of 2. Each block read fetches a target address y, which we exclusive-or with a set of columns from the disk field and with the complement vector to compute a new column from the stripe field. The first parallel read computes $b + D - 1$ columns and all subsequent parallel reads compute D columns. The total number of parallel reads is thus

$$1 + \left\lceil \frac{n - (b + D - 1)}{D} \right\rceil \quad = \quad 1 + \left\lceil \frac{\lg(N/B) - D + 1}{D} \right\rceil$$

$$= \left\lceil \frac{\lg(N/B) + 1}{D} \right\rceil .$$

Checking the kernel condition for MLD permutations

In practice, we would like a simple procedure to verify that a given matrix characterizes an MLD permutation. By the method of Section 2, it is easy to verify that a candidate matrix A is nonsingular. It may not be obvious how to verify that $\ker \lambda \subseteq \ker \mu$ when the matrix is blocked into λ and μ. Instead of determining directly whether $\ker \lambda \subseteq \ker \mu$, we check whether $\operatorname{row} \mu \subseteq \operatorname{row} \lambda$. By Lemma 5, these conditions are equivalent.

Checking whether $\operatorname{row} \mu \subseteq \operatorname{row} \lambda$ is easy. Note that the rows of the submatrix $A_{b..n-1,0..m-1}$ consist of the union of the rows of λ and μ, and observe that $\operatorname{row} \mu \subseteq \operatorname{row} \lambda$ if and only if $\operatorname{row} \lambda = \operatorname{row} A_{b..n-1,0..m-1}$. That is, if including the rows of μ adds no new vectors to the row space of λ, then the row space of μ must be a subset of the row space of λ. The condition $\operatorname{row} \lambda = \operatorname{row} A_{b..n-1,0..m-1}$ is equivalent to $\dim \operatorname{row} \lambda = \dim \operatorname{row} A_{b..n-1,0..m-1}$, which is in turn equivalent to $\operatorname{rank} \lambda = \operatorname{rank} A_{b..n-1,0..m-1}$. We can check this last condition easily by using the method of Section 2.

8 Conclusions

This paper has shown an asymptotically tight bound on the number of parallel I/Os required to perform BMMC permutations on parallel disk systems. It is particularly satisfying that the tight bound was achieved not by raising the lower bound proven here and in [9], but by decreasing the upper bound in [10]. (After all, we would rather perform BMMC permutations with fewer parallel I/Os.) The multiplicative and additive constants in the I/O complexity of our algorithm are small, which is especially fortunate in light of the expense of disk accesses. Our algorithm has been implemented on a DEC 2100 server with 8 disk drives [13]. This implementation uses asynchronous independent reads and asynchronous striped writes, so that when performing each MRC or MLD^{-1} permutation, it overlaps prefetching the next memoryload, writing the previous memoryload, and permuting in memory the current memoryload. A later implementation that runs on either the DEC 2100 server or a network of workstations [11] uses asynchronous striped reads and asynchronous independent writes; it is a key subroutine in an efficient out-of-core Fast Fourier Transform implementation [12].

One can adapt the proof by Aggarwal and Vitter [2] of Lemma 9 to bound $\Delta\Phi_{\max}$ precisely, rather than just asymptotically. In particular, it is a straightforward exercise to derive the bound

$$\Delta\Phi_{\max} \leq B \left(\frac{2}{e \ln 2} + \lg(M/B) \right) .$$

Moreover, the potential change is at most zero for write operations, and so the potential increases only during read operations. If all I/Os are simple, then the total number of blocks read equals the total number of blocks written. Therefore, we can modify the lower bound of Lemma 8 to $2\,\frac{\Phi(t)-\Phi(0)}{\Delta\Phi_{\max}}$, with which we can derive a lower bound of

$$\frac{2N}{BD} \, \frac{\operatorname{rank} \gamma}{\frac{2}{e \ln 2} + \lg(M/B)}$$

parallel I/Os for any BMMC permutation. Since the quantity $2/(e \ln 2)$ is approximately 1.06, this lower bound is quite close to the exact upper bound given by Theorem 24.

We have also shown how to detect BMMC permutations at run time, given a vector of target addresses. Detection is inexpensive and, when successful, permits the execution of our BMMC algorithm or possibly a faster algorithm for a more restricted permutation class.

Wisniewski [25] uses the linear-algebraic technique of performing column additions and row additions to derive BMMC-permutation algorithms on distributed-memory models and main-memory/cache models. On what other memory models can we use this technique to efficiently perform BMMC permutations?

What other permutations can be performed quickly? Several $O(1)$-pass permutation classes appear in [9], and this paper has added two more (MLD and MLD^{-1} permutations in Section 4). We have shown that the inverse of any one-pass permutation is a one-pass permutation. One can also show that the composition of an MLD permutation with an MLD^{-1} permutation is a one-pass permutation. What other useful permutation classes can we show to be BMMC?

Finally, is the lower bound of $\Omega\left(\frac{\Phi(t)-\Phi(0)}{\Delta\Phi_{\max}}\right)$ parallel I/Os universally tight for all permutations, not just those that are BMMC? One possible approach is to design an algorithm that explicitly manages the potential. If each pass increases the potential by $\Theta\left(\frac{N}{BD}(\Delta\Phi_{\max})\right)$, the algorithm's I/O count would match the lower bound.

Acknowledgments

Thanks to C. Esther Jeserum, Michael Klugerman, and the anonymous referees for their helpful suggestions.

References

[1] A. Aggarwal, A. K. Chandra, and M. Snir, *Hierarchical memory with block transfer*, in Proceedings of the 28th Annual Symposium on Foundations of Computer Science, Oct. 1987, pp. 204–216.

[2] A. Aggarwal and J. S. Vitter, *The input/output complexity of sorting and related problems*, Comm. ACM, 31 (1988), pp. 1116–1127.

[3] L. Arge, *The buffer tree: A new technique for optimal I/O-algorithms*, in 4th International Workshop on Algorithms and Data Structures (WADS), Aug. 1995, pp. 334–345.

[4] ——, *The I/O-complexity of ordered binary-decision diagram manipulation*, in Proceedings of the 6th International Symposium on Algorithms and Computations (ISAAC '95), J. Staples, P. Eades, N. Katoh, and A. Moffat, eds., vol. 1004 of Lecture Notes in Computer Science, Springer-Verlag, Dec. 1995, pp. 82–91.

[5] L. Arge, D. E. Vengroff, and J. S. Vitter, *External-memory algorithms for processing line segments in geographic information systems*, in Proceedings of the Third Annual European Symposium on Algorithms (ESA '95), P. Spirakis, ed., vol. 979 of Lecture Notes in Computer Science, Springer-Verlag, Sept. 1995, pp. 295–310.

[6] R. D. Barve, E. F. Grove, and J. S. Vitter, *Simple randomized mergesort for parallel disks*, in Proceedings of the 8th Annual ACM Symposium on Parallel Algorithms and Architectures, June 1996, pp. 109–118.

[7] P. Chen, G. Gibson, R. H. Katz, D. A. Patterson, and M. Schulze, *Two papers on RAIDs*, Tech. Rep. UCB/CSD 88/479, Computer Science Division (EECS), University of California, Berkeley, Dec. 1988.

[8] Y.-J. Chiang, M. T. Goodrich, E. F. Grove, R. Tamassia, D. E. Vengroff, and J. S. Vitter, *External-memory graph algorithms*, in Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, Jan. 1995, pp. 139–149.

[9] T. H. Cormen, *Virtual Memory for Data-Parallel Computing*, PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1992. Available as Technical Report MIT/LCS/TR-559.

[10] ——, *Fast permuting in disk arrays*, Journal of Parallel and Distributed Computing, 17 (1993), pp. 41–57.

[11] T. H. Cormen and M. Hirschl, *Early experiences in evaluating the Parallel Disk Model with the ViC* implementation*, Tech. Rep. PCS-TR96-293, Dartmouth College Department of Computer Science, Aug. 1996. To appear in *Parallel Computing*.

[12] T. H. Cormen and D. M. Nicol, *Performing out-of-core FFTs on parallel disk systems*, Tech. Rep. PCS-TR96-294, Dartmouth College Department of Computer Science, Aug. 1996.

[13] S. R. Cushman, *A multiple discrete pass algorithm on a DEC Alpha 2100*, Tech. Rep. PCS-TR95-259, Dartmouth College Department of Computer Science, June 1995.

[14] J. M. del Rosario and A. Choudhary, *High-performance I/O for massively parallel computers*, IEEE Computer, (1994), pp. 59–67.

[15] A. Edelman, S. Heller, and S. L. Johnsson, *Index transformation algorithms in a linear algebra framework*, IEEE Transactions on Parallel and Distributed Systems, 5 (1994), pp. 1302–1309.

[16] R. W. Floyd, *Permuting information in idealized two-level storage*, in Complexity of Computer Computations, R. E. Miller and J. W. Thatcher, eds., Plenum Press, 1972, pp. 105–109.

[17] G. A. Gibson, *Redundant Disk Arrays: Reliable, Parallel Secondary Storage*, The MIT Press, Cambridge, Massachusetts, 1992. Also available as Technical Report UCB/CSD 91/613, Computer Science Division (EECS), University of California, Berkeley, May 1991.

[18] M. T. Goodrich, J.-J. Tsay, D. E. Vengroff, and J. S. Vitter, *External-memory computational geometry*, in Proceedings of the 34th Annual Symposium on Foundations of Computer Science, Nov. 1993, pp. 714–723.

[19] S. L. Johnsson and C.-T. Ho, *Generalized shuffle permutations on boolean cubes*, Tech. Rep. TR-04-91, Harvard University Center for Research in Computing Technology, Feb. 1991.

[20] S. Lang, *Linear Algebra*, Springer-Verlag, 3rd ed., 1987.

[21] M. H. Nodine and J. S. Vitter, *Deterministic distribution sort in shared and distributed memory multiprocessors*, in Proceedings of the 5th Annual ACM Symposium on Parallel Algorithms and Architectures, June 1993, pp. 120–129.

[22] ——, *Greed sort: Optimal deterministic sorting on parallel disks*, J. Assoc. Comput. Mach., 42 (1995), pp. 919–933.

[23] G. Strang, *Linear Algebra and Its Applications*, Harcourt Brace Jovanovich, San Diego, third ed., 1988.

[24] J. S. Vitter and E. A. M. Shriver, *Algorithms for parallel memory I: Two-level memories*, Algorithmica, 12 (1994), pp. 110–147.

[25] L. F. Wisniewski, *Efficient Design and Implementation of Permutation Algorithms on the Memory Hierarchy*, PhD thesis, Dartmouth College Department of Computer Science, Mar. 1996.

[26] ——, *Structured permuting in place on parallel disk systems*, in Proceedings of the Fourth Annual Workshop on I/O in Parallel and Distributed Systems (IOPADS), May 1996, pp. 128–139.

[27] D. Womble, D. Greenberg, S. Wheat, and R. Riesen, *Beyond core: Making parallel computer I/O practical*, in DAGS '93, June 1993.