1-1-1993

Asymptotically Tight Bounds for Performing BMMC Permutations on Parallel Disk Systems

Thomas H. Cormen
Dartmouth College

Leonard F. Wisniewski
Dartmouth College

Asymptotically Tight Bounds for Performing BMMC Permutations on Parallel Disk Systems

(Extended Abstract)

Thomas H. Cormen
Leonard F. Wisniewski
Department of Mathematics and Computer Science
Dartmouth College

Abstract

We give asymptotically equal lower and upper bounds for the number of parallel I/O operations required to perform BMMC permutations (defined by a characteristic matrix that is nonsingular over $GF(2)$) on parallel disk systems. Under the Vitter-Shriver parallel-disk model with N records, D disks, block size B, and M records of RAM, we show a universal lower bound of $\Omega\left(\frac{N}{BD}\left(1+\frac{\mathrm{rank}(\gamma)}{\lg(M/B)}\right)\right)$ parallel I/Os for performing a BMMC permutation, where γ is the lower left $\lg(N/B) \times \lg B$ submatrix of the characteristic matrix. We adapt this lower bound to show that the algorithm for BPC permutations in [Cor93] is asymptotically optimal. We also present an algorithm that uses at most $\frac{2N}{BD}\left(6\left\lceil\frac{\mathrm{rank}(\gamma)}{\lg(M/B)}\right\rceil+5\right)$ parallel I/Os, which asymptotically matches the lower bound and improves upon the BMMC algorithm in [Cor93]. When $\mathrm{rank}(\gamma)$ is low, this method is an improvement over the general-permutation bound of $\Theta\left(\frac{N}{BD}\frac{\lg(N/B)}{\lg(M/B)}\right)$.

Although many BMMC permutations of practical interest fall into subclasses that might be explicitly invoked within the source code, we show how to detect in at most $N/BD+\left\lceil\frac{\lg(N/B)+1}{D}\right\rceil$ parallel I/Os 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 our algorithm.

1 Introduction

The I/O complexity of operations with data on disk is based on data movement. As one of the most basic data-movement operations, permuting forms an important part of the theory of I/O complexity for data on disk.

Portions of this research were performed while Tom Cormen was at the MIT Laboratory for Computer Science and appear in [Cor92]. He was supported in part by the Defense Advanced Research Projects Agency under Grant N00014-91-J-1698. Other portions of this research were performed while at Dartmouth College and were supported in part by funds from Dartmouth College. Len Wisniewski is supported in part by INFOSEC Grant 3-56666.

Moreover, performing permutations efficiently when the data reside on disk is of great practical interest. The problems that we attack with supercomputers are ever-increasing in size, and we find more and more that matrices and vectors exceed the memory provided by even the largest supercomputers. 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 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 three important results:

1. universal lower bounds for BMMC and BPC permutations,
2. an algorithm for performing BMMC permutations whose I/O complexity asymptotically matches the lower bound, thus making it asymptotically optimal, and
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.

Depending on the exact BMMC permutation, this asymptotically optimal bound may be significantly less than the asymptotically optimal bound proven for general permutations.

Model and previous results

We use the parallel-disk model first proposed by Vitter and Shriver [VS90, VS92], who also gave asymptotically optimal algorithms for several problems including sorting and general permutations. In the Vitter-Shriver 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 organized in *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*, or *RAM*, capable of holding M records. Each *parallel I/O operation* transfers up to D blocks between the disks and RAM, with at most one block transferred per disk, for a total of up to BD records transferred. The blocks accessed in a single parallel I/O may be at any locations on

	$\mathcal{D}_0$		$\mathcal{D}_1$		$\mathcal{D}_2$		$\mathcal{D}_3$		$\mathcal{D}_4$		$\mathcal{D}_5$		$\mathcal{D}_6$		$\mathcal{D}_7$	
track 0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
track 1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
track 2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
track 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 tracks needed is $N/BD = 4$. Numbers indicate record indices.

their respective disks; we enforce no requirement on lockstep head movement. 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 number of disk accesses, however, can be minimized by carefully designed algorithms such as those in this paper.

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 $BD \le M$. Also, we assume that $M < N$, since otherwise we can just perform all operations in memory. These two requirements imply that $b + d \le m < n$.

The Vitter-Shriver model lays out data on a parallel disk system as shown in Figure 1. A *track* consists of the D blocks at the same location on all D disks. Record indices vary most rapidly within a block, then among disks, and finally among tracks. We indicate the address of a record as an n-bit vector x with the least significant bit first: $x = (x_0, x_1, \ldots, x_{n-1})$. 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 track number by the remaining $n - (b + d)$ 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. 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 [VS90, VS92] for randomized sorting and by Nodine and Vitter [NV90, NV91, NV92] for deterministic sorting. These bounds are asymptotically tight, for they match the lower bounds proven by Aggarwal and Vitter [AV88] using a model with one disk and D independent read/write heads, which is at least as powerful as the Vitter-Shriver 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 [Cor93] studied several classes of bit-defined permutations that include matrix transposition as a special case. Table 1 shows the classes of permutations examined and the corresponding upper bounds derived in [Cor93].

BMMC permutations

The most general class considered in [Cor93] is *bit-matrix-multiply/complement*, or *BMMC*, permutations.[1] In a BMMC permutation, we have 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] and we have 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 of $GF(2)$ and then form the corresponding n-bit target address y by complementing some subset of the resulting bits: $y = Ax \oplus c$.

The BMMC algorithm in [Cor93] 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

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

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 [Cor92] for example), but it has been unknown whether the $\Theta(N/BD + H(N, M, B))$ 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

[1] Edelman, Heller, and Johnsson [EHJ92] 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 XOR.

[3] Johnsson and Ho [JH91] call BPC permutations *dimension permutations*, and Aggarwal, Chandra, and Snir [ACS87] call BPC permutations without complementing *rational permutations*.

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	$2\left\lceil \frac{\rho(A)}{\lg(M/B)} \right\rceil + 1$
Block BMMC	$\begin{array}{c} \begin{array}{cc} b & n-b \end{array} \\ \left[\begin{array}{c|c} \text{nonsingular} & 0 \\ \hline 0 & \text{nonsingular} \end{array}\right] \begin{array}{l} b \\ n-b \end{array} \end{array}$	1
MRC (memory-rearrangement/ complement)	$\begin{array}{c} \begin{array}{cc} m & n-m \end{array} \\ \left[\begin{array}{c|c} \text{nonsingular} & \text{arbitrary} \\ \hline 0 & \text{nonsingular} \end{array}\right] \begin{array}{l} m \\ n-m \end{array} \end{array}$	1

Table 1: Classes of permutations, their characteristic matrices, and upper bounds shown in [Cor93] 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 block BMMC and 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).

class of BPC permutations includes many common permutations such as matrix transposition, bit-reversal permutations (used in performing FFTs), vector-reversal permutations, hypercube permutations, and matrix reblocking.

We express the I/O complexity of BPC permutations in terms of cross-ranks. For any $n \times n$ permutation matrix A and for any $k = 0, 1, \ldots, n-1$, we define the *k-cross-rank* of A as

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

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:

$$\rho(A) = \max(\rho_b(A), \rho_m(A)) \ . \quad (3)$$

The BPC algorithm in [Cor93] uses at most

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

parallel I/Os. As we shall see in Section 2, this algorithm is asymptotically optimal.

One-pass permutations

We shall use the remaining two classes in Table 1 as subroutines in performing BMMC permutations. Each class is a restricted BMMC permutation, described by a characteristic matrix and a complement vector, and is shown in [Cor93] to require only one pass of N/BD parallel reads and N/BD parallel writes.

Block BMMC permutations have the restrictions that both the leading $b \times b$ submatrix and the trailing $(n-b) \times (n-b)$ submatrix are nonsingular and the rest of the characteristic matrix is 0.

Memory-rearrangement/complement, or *MRC*, permutations have the restrictions that both the leading $m \times m$ and trailing $(n-m) \times (n-m)$ submatrices 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. Any MRC permutation can be performed by reading in a *memoryload* (i.e., M records) of M/BD tracks, permuting the records read within memory, and writing them out to M/BD tracks. The class of MRC permutations includes those characterized by unit upper triangular matrices. As [Cor93] shows, both the standard binary-reflected Gray code and its inverse have characteristic matrices of this form, and so they are MRC permutations.

Outline

The remainder of this paper is organized as follows. Section 2 states and proves the lower bounds for BMMC and BPC permutations. In Section 3, we describe an algorithm for BMMC permutations whose I/O complexity asymptotically matches the lower bound. Section 4 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. Finally, Section 5 contains some concluding remarks.

We shall use several notational conventions in this paper. Matrix row and column numbers are indexed from 0 starting from the upper left. As in equation (2), we index rows and columns by sets to indicate submatrices, using ".." notation to indicate sets of contiguous numbers. 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.

2 Lower bounds for BMMC and BPC permutations

In this section, we state and prove the lower bounds for BMMC and BPC permutations. After stating the lower bounds, we briefly discuss their significance before presenting the full proofs. The lower bounds are given by the following theorems.

Theorem 1 *Any algorithm that performs a 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 $(n-b)\times b$. ■

Theorem 2 *Any algorithm that performs a BPC permutation with characteristic matrix A requires*

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

parallel I/Os. ■

These lower bounds are *universal* in the sense that they apply to all inputs. 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 we may be able to do better.

There are in fact algorithms that achieve the bounds given by Theorems 1 and 2, and so these algorithms are asymptotically optimal. Section 3 presents an algorithm for BMMC permutations, and the BPC algorithm of [Cor93] achieves the bound of Theorem 2. The matrix-transposition bound of $\Theta\left(\frac{N}{BD}\left(1+\frac{\lg\min(B,R,S,N/B)}{\lg(M/B)}\right)\right)$ parallel I/Os is consistent with the BPC formulation; see [Cor92] for details.

Proofs of the lower bounds

To prove Theorems 1 and 2, we rely heavily on the technique used by Aggarwal and Vitter [AV88] for a lower bound on I/Os in matrix transposition; their proof is based in turn on a method by Floyd [Flo72]. We prove the lower bound for the case in which $D = 1$, the general case following 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 RAM. An output is simple if the records are removed from the RAM 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 3 *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.* ■

Potential function

The basic scheme of the proof of Theorem 1 uses a potential function argument. We call the time interval starting when the qth I/O completes and ending just before the $(q+1)$st I/O starts *time q*. We define a potential function Φ so that $\Phi(q)$ is the *potential* at time q. 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. (Remember that $D = 1$.) We denote by $g_{\text{block}}(i,k,q)$ the number of records in the ith target group that are in block k at time q, and $g_{\text{mem}}(i,q)$ denotes the number of records in the ith target group that are in RAM 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 RAM 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 one is based on the observation that the number of parallel I/Os is at least the total change in potential over all parallel I/Os divided by the maximum change in potential in any single parallel I/O.

Lemma 4 *Let $D = 1$, and consider any algorithm that performs a permutation. The increase in potential due to any parallel I/O operation is $O(B \lg(M/B))$. Therefore, any algorithm that uses t parallel I/Os to perform a permutation must have $t = \Omega\left(\frac{N}{B} + \frac{\Phi(t)-\Phi(0)}{B\lg(M/B)}\right)$.* ■

Lemma 5 *Let $D = 1$, and consider any permutation that can be performed with t parallel I/Os. Then $\Phi(t) = N \lg B$. Therefore, any algorithm that uses t parallel I/Os to perform a permutation uses $\Omega\left(\frac{N}{B} + \frac{N\lg B-\Phi(0)}{B\lg(M/B)}\right)$ parallel I/Os.* ■

Observe that these lemmas imply lower bounds that are universal. No matter what the input, the initial potential is $\Phi(0)$, the final potential is $\Phi(t)$, the increase in potential per parallel I/O is bounded by $O(B \lg(M/B))$, and so $\Omega\left(\frac{\Phi(t)-\Phi(0)}{B\lg(M/B)}\right)$ parallel I/Os are required. The $\Omega(N/B)$ term is the trivial lower bound due to reading the entire input.

Ranges and domains

To prove the lower bounds, we shall examine ranges of matrices and domains of vectors under matrix multiplication. We omit the proofs of two simple lemmas.

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 values 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

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

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

Lemma 6 *Let A be a $p \times q$ matrix whose entries are drawn from $\{0, 1\}$, let x be any p-vector whose entries are drawn from $\{0, 1\}$, and let $r = \text{rank}(A)$. Then $|\mathcal{R}(A) \oplus x| = 2^r$.* ■

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

$$\text{Dom}(A, y) = \{x : Ax = y\} .$$

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

Lemma 7 *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 $r = \text{rank}(A)$. Then $|\text{Dom}(A, y)| = 2^{q-r}$.* ■

Proof of Theorem 1

To prove Theorem 1, we prove the lower bound for the case in which $D = 1$, the general case following by dividing by D. 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, RAM is initially empty.

We need to compute the initial potential in order to apply Lemma 5. 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

$$k = x_{b..n-1} , \tag{4}$$

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

$$\begin{aligned} i &= A_{b..n-1,0..n-1}\, x_{0..n-1} \\ &= A_{b..n-1,0..b-1}\, x_{0..b-1} \oplus A_{b..n-1,b..n-1}\, x_{b..n-1} , \end{aligned} \tag{5}$$

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 8 *Let $r = \text{rank}(A_{b..n-1,0..b-1})$, 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 (4), and so they map to target block numbers given by condition (5) but with $x_{b..n-1}$ fixed at k. The range of target block numbers is thus $\mathcal{R}(A_{b..n-1,0..b-1}) \oplus A_{b..n-1,b..n-1}\, k$ which, by Lemma 6, 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 $\mathcal{R}(A_{b..n-1,0..b-1}) \oplus A_{b..n-1,b..n-1}\, k$. Again fixing $x_{b..n-1} = k$ in condition (5) and exclusive-oring both sides by $A_{b..n-1,b..n-1}\, k$, we see that this set is precisely $\text{Dom}(A_{b..n-1,0..b-1}, i \oplus A_{b..n-1,b..n-1}\, k)$. By Lemma 7, this set has cardinality exactly 2^{b-r}, which equals $B/2^r$. ■

We can interpret Lemma 8 as follows. Let $r = \text{rank}(A_{b..n-1,0..b-1})$, 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 RAM is initially empty, $g_{\text{mem}}(i, 0) = 0$ for all blocks i, which implies that $G_{\text{mem}}(0) = 0$. We have

$$\begin{aligned} \Phi(0) &= G_{\text{mem}}(0) + \sum_{k=0}^{N/B-1} G_{\text{block}}(k, 0) \\ &= 0 + \sum_{k=0}^{N/B-1} \sum_{i=0}^{N/B-1} f(g_{\text{block}}(i, k, 0)) \\ &= \sum_{k=0}^{N/B-1} 2^r \frac{B}{2^r} \lg \frac{B}{2^r} \qquad \text{(by Lemma 8)} \\ &= \frac{N}{B} B \lg \frac{B}{2^r} \\ &= N(\lg B - r) . \end{aligned} \tag{6}$$

Combining Lemma 5 and equation (6), we get a lower bound of

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

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

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

which completes the proof of Theorem 1.

Proof of Theorem 2

We use Theorem 1 and the following lemma to prove Theorem 2.

Lemma 9 *For any permutation matrix A, we have*

$$\rho_b(A) - \lg(M/B) \le \rho_m(A) \le \rho_b(A) + \lg(M/B) .$$

Proof: Since the rank of any submatrix of a permutation matrix is equal to the number of 1s in the submatrix, we

have that

$$\begin{aligned}\rho_m(A) &= \operatorname{rank}(A_{m..n-1,0..m-1}) \\ &= \operatorname{rank}(A_{b..n-1,0..b-1}) - \operatorname{rank}(A_{b..m-1,0..b-1}) \\ &\quad + \operatorname{rank}(A_{m..n-1,b..m-1}) \\ &= \rho_b(A) - \operatorname{rank}(A_{b..m-1,0..b-1}) \\ &\quad + \operatorname{rank}(A_{m..n-1,b..m-1}) \ . \qquad (7)\end{aligned}$$

The rank of a submatrix is nonnegative and is at most the smaller of the number of rows and the number of columns, which implies that

$$\begin{aligned}0 \le \operatorname{rank}(A_{b..m-1,0..b-1}) &\le m-b = \lg(M/B) \ , \\ 0 \le \operatorname{rank}(A_{m..n-1,b..m-1}) &\le m-b = \lg(M/B) \ .\end{aligned}$$

Combining the above inequalities with equation (7) yields

$$\begin{aligned}\rho_b(A) - \lg(M/B) &\le \rho_b(A) - \operatorname{rank}(A_{b..m-1,0..b-1}) \\ &\le \rho_m(A) \\ &\le \rho_b(A) + \operatorname{rank}(A_{m..n-1,b..m-1}) \\ &\le \rho_b(A) + \lg(M/B) \ ,\end{aligned}$$

which completes the proof. ■

To prove Theorem 2, we note that Lemma 9 implies that

$$\frac{\operatorname{rank}(A_{b..n-1,0..b-1})}{\lg(M/B)} = \frac{\rho_b(A)}{\lg(M/B)} = \Theta\left(\frac{\rho(A)}{\lg(M/B)}\right) \ .$$

Combined with Theorem 1, we get a lower bound of

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

parallel I/Os for BPC permutations.

3 An asymptotically optimal BMMC algorithm

In this section, we present an algorithm to perform BMMC permutations by factoring them into simpler permutations. As usual, we assume that the BMMC permutation is given by an $n \times n$ characteristic matrix A and a complement vector c of length n. The number of parallel I/Os is at most $\frac{2N}{BD}\left(6\left\lceil\frac{\operatorname{rank}(\gamma)}{\lg(M/B)}\right\rceil + 5\right)$ parallel I/Os, where γ is the submatrix $A_{b..n-1,0..b-1}$.

Our strategy is to factor the matrix A into a product of matrices, each of which is the characteristic matrix of one of four types of permutations: MRC, block BMMC, BPC, and a fourth type described in Appendix A. For now, we ignore the complement vector c. We read the factors right to left to determine the order in which to perform these permutations. For example, if we factor $A = VW$, then we perform A by first permuting according to characteristic matrix W and then permuting according to characteristic matrix V. The reason for this right-to-left order is that if $y = Ax$, then we first multiply Wx, giving y', and then multiply Vy', giving y. Factoring the matrix A in this way will make it easy to count how many passes the BMMC permutation takes.

Permuting columns to create a nonsingular trailing submatrix

We start by permuting the columns of A so that the trailing $(n-b) \times (n-b)$ submatrix is nonsingular. That is, we divide A between the lower b rows and columns and the upper $n-b$ rows and columns and factor it as $A = \widehat{A}\,\Pi$, where

$$A = \left[\begin{array}{c|c} \alpha & \beta \\ \hline \gamma & \delta \end{array}\right] \begin{matrix} b \\ n-b \end{matrix} \quad \begin{matrix} b & n-b \end{matrix} \ ,$$

$$\widehat{A} = \left[\begin{array}{c|c} \widehat{\alpha} & \widehat{\beta} \\ \hline \widehat{\gamma} & \widehat{\delta} \end{array}\right] \begin{matrix} b \\ n-b \end{matrix} \quad \begin{matrix} b & n-b \end{matrix} \ ,$$

Π is a permutation matrix, and $\widehat{\delta} = \widehat{A}_{b..n-1,b..n-1}$ is a nonsingular submatrix. We would like to permute the columns of A so that the number of I/Os required to perform Π is minimum. We call such a permutation a *minimum-impact* permutation. The size of the largest set S of columns of δ that are linearly independent is equal to $\operatorname{rank}(\delta)$. We can come up with a minimum-impact permutation by choosing a set T of $n-b-\operatorname{rank}(\delta)$ columns from the leftmost b columns of A that, along with S, provide the needed set of $n-b$ linearly independent columns. To find these sets S and T, we can use Gaussian elimination, as described in [Cor92]. Because the permutation characterized by Π exchanges $n-b-\operatorname{rank}(\delta)$ columns between the leftmost b and the rightmost $n-b$ columns, one can easily show that

$$\rho(\Pi) = n - b - \operatorname{rank}(\delta) \ . \qquad (8)$$

Moreover, because there are at least $n-b-\operatorname{rank}(\delta)$ linearly independent columns in γ that are also in $\widehat{\delta}$, we have

$$\operatorname{rank}(\gamma) \ge n - b - \operatorname{rank}(\delta) \ . \qquad (9)$$

Combining properties (8) and (9) yields $\rho(\Pi) \le \operatorname{rank}(\gamma)$, and so we can perform the BPC permutation characterized by Π with at most $2\left\lceil\frac{\operatorname{rank}(\gamma)}{\lg(M/B)}\right\rceil + 1$ passes. Note also that because the linearly independent columns in $\widehat{\gamma}$ come from either γ or from the columns of δ that are permuted to $\widehat{\gamma}$, we have that

$$\begin{aligned}\operatorname{rank}(\widehat{\gamma}) &\le \operatorname{rank}(\gamma) + n - b - \operatorname{rank}(\delta) \\ &\le 2\ \operatorname{rank}(\gamma) \qquad (10)\end{aligned}$$

Observe that whenever a nonsingular matrix is expressed as a product of matrix factors, each of the factors must itself be nonsingular. This fact follows because the rank of a matrix product is at most the rank of any factor, so if the product of square matrices has full rank, each factor must have full rank as well.

Factoring the remaining matrix

Our next task is to factor the matrix $\widehat{A}$, which is nonsingular by the above argument. We factor $\widehat{A}$ as

$$\widehat{A} = U\,V\,W \ ,$$

where

$$U = \left[\begin{array}{c|c} \widehat{\alpha} \oplus \widehat{\beta}\,\widehat{\delta}^{-1}\,\widehat{\gamma} & \widehat{\beta}\,\widehat{\delta}^{-1} \\ \hline 0 & I \end{array}\right] \begin{array}{l} b \\ n-b \end{array}\ ,$$

$$V = \left[\begin{array}{c|c} I & 0 \\ \hline \widehat{\gamma} & I \end{array}\right] \begin{array}{l} b \\ n-b \end{array}\ ,$$

$$W = \left[\begin{array}{c|c} I & 0 \\ \hline 0 & \widehat{\delta} \end{array}\right] \begin{array}{l} b \\ n-b \end{array}\ ,$$

and so the factors U, V, and W are nonsingular. Since $\widehat{\delta}$ is nonsingular, W characterizes a block BMMC permutation, and so it requires only one pass. The upper $b \times b$ submatrix $\widehat{\alpha} \oplus \widehat{\beta}\,\widehat{\delta}^{-1}\,\widehat{\gamma}$ of U is nonsingular because if it contained linearly dependent columns, then so would U, contradicting the nonsingularity of U. Thus, U characterizes an MRC permutation, which only requires one pass.

We further divide the matrix V and factor it as

$$V = P\,Q\ ,$$

where

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

and

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

Here, we have divided the submatrix $\widehat{\gamma}$ as

$$\widehat{\gamma} = \left[\begin{array}{c} \sigma \\ \hline \tau \end{array}\right] \begin{array}{l} b \\ m-b \\ n-m \end{array}\ . \tag{11}$$

The upper $m \times m$ submatrix of P is unit lower triangular and, therefore, nonsingular. Thus, P characterizes an MRC permutation, requiring only one pass.

It remains to perform the permutation characterized by the matrix Q, whose lower left $(n-m) \times b$ submatrix τ may be nonzero. Our strategy is to factor Q into $g = \lceil \mathrm{rank}(\tau)/(m-b) \rceil$ matrices $Q^{(1)}, Q^{(2)}, \ldots, Q^{(g)}$, each of which characterizes a permutation that requires only two passes. Note that equation (11) implies that $\mathrm{rank}(\tau) \le \mathrm{rank}(\widehat{\gamma})$ and, using inequality (10), we have $g \le \lceil 2\,\mathrm{rank}(\gamma)/(m-b) \rceil$.

To factor Q, we need a column basis C (a maximal linearly independent set of columns) for τ, which again we can find by Gaussian elimination. Having found C, let τ_C denote the submatrix of τ containing the columns indexed by C. Let $C' = \{0, 1, \ldots, b-1\} - C$ index the columns of τ not in the column basis. We partition C into g sets $C^{(1)}, C^{(2)}, \ldots, C^{(g)}$, where $\left|C^{(i)}\right| \le m-b$ for $i = 1, 2, \ldots, g$. For each index $j \in C'$, the column τ_j is a linear combination of columns in τ_C; we define $C_j^{(i)}$ to be the set of column indices in $C^{(i)}$ that contribute to this sum of columns. That is, for each $j \in C'$, we have

$$\tau_j = \bigoplus_{i=1}^{g} \left(\bigoplus_{k \in C_j^{(i)}} \tau_k \right) .$$

Each factor $Q^{(i)}$ is the matrix

$$Q^{(i)} = \left[\begin{array}{c|c|c} I & 0 & 0 \\ \hline 0 & I & 0 \\ \hline \tau^{(i)} & 0 & I \end{array}\right] \begin{array}{l} b \\ m-b \\ n-m \end{array}\ , \tag{12}$$

where the $(n-m) \times b$ submatrix $\tau^{(i)}$ has columns defined as follows:

- For $j \in C^{(i)}$, we set $\tau_j^{(i)} = \tau_j$.
- For $j \in C - C^{(i)}$, we set $\tau_j^{(i)} = 0$.
- For $j \in C'$, we set $\tau_j^{(i)} = \bigoplus_{k \in C_j^{(i)}} \tau_k$.

The matrices $\tau^{(i)}$ are constructed so that $\mathrm{rank}(\tau^{(i)}) = \left|C^{(i)}\right| \le m-b$ and that $\tau = \tau^{(1)} \oplus \tau^{(2)} \oplus \cdots \oplus \tau^{(g)}$. One can verify that $Q = Q^{(1)}\,Q^{(2)}\,\cdots\,Q^{(g)}$, as required.

To recap, we have factored our original BMMC characteristic matrix A into $A = U\,P\,Q^{(1)}\,Q^{(2)}\,\cdots\,Q^{(g)}\,W\,\Pi$. Both U and P characterize MRC permutations, and so does their product $Z = U\,P$. We can therefore express A with one fewer factor: $A = Z\,Q^{(1)}\,Q^{(2)}\,\cdots\,Q^{(g)}\,W\,\Pi$. Appendix A sketches how to perform each permutation characterized by a factor $Q^{(i)}$ in only two passes. (The procedure is conceptually complex but computationally easy.) The factors Z and W require one pass each, and the BPC permutation given by Π requires at most $2\left\lceil \frac{\mathrm{rank}(\gamma)}{\lg(M/B)} \right\rceil + 1$ passes. The BMMC permutation given by A, therefore, requires at most

$$\begin{aligned} & 2g + 2 + 2\left\lceil \frac{\mathrm{rank}(\gamma)}{\lg(M/B)} \right\rceil + 1 \\ & \qquad \le 2\left(\left\lceil \frac{2\ \mathrm{rank}(\gamma)}{\lg(M/B)} \right\rceil + \left\lceil \frac{\mathrm{rank}(\gamma)}{\lg(M/B)} \right\rceil \right) + 3 \\ & \qquad \le 6\left\lceil \frac{\mathrm{rank}(\gamma)}{\lg(M/B)} \right\rceil + 5 \end{aligned}$$

passes.

Earlier, we deferred discussion of the complement vector c. To include it in the BMMC permutation, we perform it as part of the last MRC permutation, which is characterized by Z. No additional pass is needed to incorporate the complement vector into the BMMC permutation.

4 Detecting BMMC permutations at run time

In practice, we wish to run the BMMC algorithm of Section 3 whenever possible to reap the savings over having run the more costly algorithm for general permutations. For that matter, we wish to run the BPC, MRC, and block BMMC algorithms of [Cor93] 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, a permutation matrix for a BPC permutation, etc. If instead the permutation is given by 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.
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 $y \neq Ax \oplus c$, the permutation is not BMMC and we can terminate verification. If $y = Ax \oplus c$ for all N source-target pairs, the permutation is BMMC. Verification uses at most N/BD parallel reads.

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 permutation to perform 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 ΠG, 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 ΠG 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 $y = Ax \oplus c$ 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 $y = Ax \oplus c$, then we have

$$y = \left(\bigoplus_{j \in S_k} A_j \right) \oplus A_k \oplus c \ , \tag{13}$$

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 (13) to yield the kth column of A:

$$A_k = y \oplus \left(\bigoplus_{j \in S_k} A_j \right) \oplus c \ . \tag{14}$$

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 (14). In other words, $S_k \subseteq \{0, 1, \ldots, k-1\}$. Recall that in an address, the lower b bits give the record's offset within its block, the middle d bits give the disk number, and the upper $n-(b+d)$ bits give the track number.

From equation (14), 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 track fields are all zero contains b such addresses, so they can be fetched in one disk read. A problem arises for the $n-(b+d)$ source addresses with one 1 in the track field: their target addresses all reside on different blocks of disk $\mathcal{D}_0$. Each 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 $n-(b+d)+1 = \lg(N/BD)+1$.

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 track 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 track 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 (14). For example, let us compute column A_{b+d}, which corresponds to the first bit of the track number. We read track 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 (14) 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 track 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 $D - d - 1$ track-bit columns from the first parallel read.

The remaining parallel reads compute the remaining track-bit columns. We follow the track-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 track 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

$$\begin{aligned} 1 + \left\lceil \frac{n - (b + D - 1)}{D} \right\rceil &= 1 + \left\lceil \frac{\lg(N/B) - D + 1}{D} \right\rceil \\ &= \left\lceil \frac{\lg(N/B) + 1}{D} \right\rceil . \end{aligned}$$

5 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 [Cor92], but by decreasing the upper bound in [Cor93]. The constant factors in the I/O complexity of our algorithm are small, which is especially fortunate in light of the expense of disk accesses.

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.

An important open problem is determining exact, rather than asymptotic, lower bounds. It may yet be possible to "hack the constants" in the upper bound. Without exact lower bounds, however, we do not know when to stop hacking.

Finally, we ask what other permutations can be performed quickly? [Cor92] presents several $O(1)$-pass permutation classes, and this paper has added one more (MLD permutations in Appendix A). What other useful permutation classes can we show to be BMMC? Can we generalize BMMC permutations further to useful classes that can also be performed faster than the general permutation bound?

Acknowledgments

Thanks to Tom Sundquist, C. Esther Jeserum, and Michael Klugerman for their helpful suggestions.

References

[ACS87] Alok Aggarwal, Ashok K. Chandra, and Marc Snir. Hierarchical memory with block transfer. In *Proceedings of the 28th Annual Symposium on Foundations of Computer Science*, pages 204–216, October 1987.

[AV88] Alok Aggarwal and Jeffrey Scott Vitter. The input/output complexity of sorting and related problems. *Communications of the ACM*, 31(9):1116–1127, September 1988.

[Cor92] Thomas 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.

[Cor93] Thomas H. Cormen. Fast permuting in disk arrays. *Journal of Parallel and Distributed Computing*, 17(1–2):41–57, January and February 1993.

[EHJ92] Alan Edelman, Steve Heller, and S. Lennart Johnsson. Index transformation algorithms in a linear algebra framework. Technical Report LBL-31841, Lawrence Berkeley Laboratory, 1992.

[Flo72] Robert W. Floyd. Permuting information in idealized two-level storage. In Raymond E. Miller and James W. Thatcher, editors, *Complexity of Computer Computations*, pages 105–109. Plenum Press, 1972.

[JH91] S. Lennart Johnsson and Ching-Tien Ho. Generalized shuffle permutations on boolean cubes. Technical Report TR-04-91, Harvard University Center for Research in Computing Technology, February 1991.

[NV90] Mark H. Nodine and Jeffrey Scott Vitter. Greed sort: An optimal external sorting algorithm for multiple disks. Technical Report CS-90-04, Department of Computer Science, Brown University, February 1990.

[NV91] Mark H. Nodine and Jeffrey Scott Vitter. Large-scale sorting in parallel memories. In *Proceedings of the 3rd Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 29–39, July 1991.

[NV92] Mark H. Nodine and Jeffrey Scott Vitter. Optimal deterministic sorting on parallel disks. Technical Report CS-92-08, Department of Computer Science, Brown University, 1992.

[VS90] Jeffrey Scott Vitter and Elizabeth A. M. Shriver. Optimal disk I/O with parallel block transfer. In *Proceedings of the Twenty Second Annual ACM Symposium on Theory of Computing*, pages 159–169, May 1990.

[VS92] Jeffrey Scott Vitter and Elizabeth A. M. Shriver. Algorithms for parallel memory I: Two-level memories. Technical Report CS-92-04, Department of Computer Science, Brown University, August 1992. Revised version of Technical Report CS-90-21.

A Performing permutations characterized by $Q^{(i)}$ in two passes

This appendix sketches how to perform each permutation characterized by a matrix $Q^{(i)}$, as defined in equation (12), in only two passes. A complete description will appear in the full paper.

Our method is to factor each matrix $Q^{(i)}$ into two matrices, each of which characterizes a one-pass permutation. One of these matrices in fact characterizes an MRC permutation. The other characterizes a new type of one-pass permutation, which we call a *memoryload-dispersal*, or *MLD*, permutation.

For notational convenience, we assume from here on that we are working with a matrix $Q^{(i)}$ for a given value of i, and we drop the superscripts. Thus, we shall speak of Q, C, and τ with the implicit understanding that we really mean a given $Q^{(i)}$, $C^{(i)}$, and $\tau^{(i)}$.

Before presenting our factoring of a matrix Q, we need to define some further index sets for submatrix rows and columns and some special matrices. Each matrix Q has an associated index set C, where $|C| \le m-b$, indexing a column basis for the lower left $(n-m) \times b$ submatrix τ. We further define the following:

- $r = |C|$, so that $\mathrm{rank}(\tau) = r \le m - b$.
- $\overline{C} = \{0, 1, \ldots, b-1\} - C$.
- R is a set of row indices in τ such that the $r \times r$ submatrix $\tau_{R,C}$ is nonsingular. (In other words, we choose a row basis $\tau_{R,C}$ for the column basis τ_C.)
- I_C is an $r \times b$ matrix formed by expanding an $r \times r$ identity matrix out to b columns with the columns in $\overline{C}$ being 0 and the columns in C being columns of the identity matrix. For example, if $C = \{0, 2, 3\}$ (so that $r = 3$) and $b = 5$, then

$$I_C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} .$$

  Similary, $I_{\overline{C}}$ is a $(b-r) \times b$ matrix formed by expanding a $(b-r) \times (b-r)$ identity matrix out to b columns with the columns in C being 0 and the columns in $\overline{C}$ being columns of the identity matrix.

We factor the nonsingular matrix as $Q = X\,Y$, where

$$X = \left[\begin{array}{c|c|c|c} \tau' & 0 & (I_C)^{\mathrm{T}} & 0 \\ \hline 0 & I & 0 & 0 \\ \hline 0 & 0 & 0 & I \end{array}\right] \begin{array}{l} b \\ m-b \\ n-m \end{array} \quad ,$$

$$Y = \left[\begin{array}{c|c|c} I_{\overline{C}} & 0 & 0 \\ \hline 0 & I & 0 \\ \hline (\tau_{R,C})^{-1}\tau_{R,0..b-1} & 0 & 0 \\ \hline \tau & 0 & I \end{array}\right] \begin{array}{l} b-r \\ m-b \\ r \\ n-m \end{array} \quad ,$$

and the submatrix τ' of X is given by

$$\tau' = (I_C)^{\mathrm{T}} (\tau_{R,C})^{-1} \tau_{R,\overline{C}} \oplus (I_{\overline{C}})^{\mathrm{T}} .$$

(We omit the proof that this factorization is correct.) Because Q is nonsingular, so are X and Y. Observe that the leading $m \times m$ submatrix of X must be nonsingular, for otherwise there are linearly dependent columns among the leftmost m columns of X, and X would then be singular. Therefore, X characterizes an MRC permutation, as we claimed above would be the case.

The matrix Y, however, characterizes a new type of permutation that we call memoryload-dispersal or MLD. We shall not delve into the details of performing MLD permutations here, but we sketch how to perform one. Using Lemma 6, one can show that because $\mathrm{rank}(\tau) = r \le m - b$, each source memoryload maps to exactly $2^r \le M/B$ target memoryloads. In addition, one can use Lemma 7 to show that given a target memoryload, if a particular source memoryload maps any records to this target, then it maps exactly $2^{m-r} \ge B$ such records. We can partition the source and target memoryload numbers into sets of 2^r source memoryloads and 2^r target memoryloads such that each source set has a unique target set that all of its records map to. We can perform the MLD permutation by reading in a source memoryload at a time, permuting it in memory according to the characteristic matrix Y, and writing it out so that the records mapping to each block image in RAM are destined for the same target memoryload. Moreover, the structure of Y ensures that the blocks are spread out evenly across the disks in the target memoryloads. We read the source records a memoryload at a time, we write the target memoryloads a block at a time, and every block of every target memoryload is written exactly once by the time we are finished processing all the source memoryloads. This procedure takes one pass. After performing the MLD permutation characterized by Y, every target memoryload has the records destined for it according to the matrix Q; the MRC permutation characterized by X then puts each record into its correct location within its memoryload.