

Dartmouth College Computer Science Technical Report TR2001-402
Optimizing the Dimensional Method for Performing
Multidimensional, Multiprocessor, Out-of-Core FFTs

Jeremy T. Fineman
Dartmouth College
Department of Computer Science

Advisor: Thomas H. Cormen

June 1, 2001

Abstract

We present an improved version of the Dimensional Method for computing multidimensional Fast Fourier Transforms (FFTs) on a multiprocessor system when the data consist of too many records to fit into memory. Data are spread across parallel disks and processed in sections. We use the Parallel Disk Model for analysis.

The simple Dimensional Method performs the 1-dimensional FFTs for each dimension in turn. Between each dimension, an out-of-core permutation is used to rearrange the data to contiguous locations. The improved Dimensional Method processes multiple dimensions at a time.

We show that determining an optimal sequence and groupings of dimensions is NP-complete. We then analyze the effects of two modifications to the Dimensional Method independently: processing multiple dimensions at one time, and processing single dimensions in a different order.

Finally, we show a lower bound on the I/O complexity of the Dimensional Method and present an algorithm that is approximately asymptotically optimal.

Contents

1	Introduction	2
1.1	Fast Fourier Transforms	2
1.2	The Parallel Disk Model	2
1.3	BMMC Permutations	3
1.4	Some BMMC permutation matrices	4
1.5	Technique for BMMC complexity analysis	5
2	The Dimensional Method	6
2.1	The Basic Dimensional Method	6
2.2	Optimization	7
2.3	NP-Completeness	7
2.4	Grouping Dimensions	8
2.5	Ordering Dimensions	10
2.6	Analysis of Permutations	11
2.7	Determining the Optimal Ordering	19
2.8	Lower Bound on Dimensional Method	21
2.9	Approximation of Optimal Solution	23
2.10	Some Notes on Optimization	25

Chapter 1

Introduction

1.1 Fast Fourier Transforms

Fourier Transforms are based on complex roots of unity. The complex number $\omega_N = e^{2\pi i/N}$, where $i = \sqrt{-1}$ and $e^{iu} = \cos(u) + i \sin(u)$ for any real u , is called the *principal N th root of unity*. Without loss of generality, we assume that N is a power of 2.

Given a vector $a = (a_0, a_1, \dots, a_{N-1})$, the vector $y = (y_0, y_1, \dots, y_{N-1})$ is the *Discrete Fourier Transform* (DFT), where

$$y_k = \sum_{j=0}^{N-1} a_j \omega_N^{kj} .$$

Multidimensional DFTs have a similar definition. We are given k dimensions N_1, N_2, \dots, N_k where $N = N_1 N_2 \dots N_k$. We assume that each dimension is an integer power of 2. We are also given a k -dimensional array $A[0 : N_1 - 1, 0 : N_2 - 1, \dots, 0 : N_k - 1]$. The DFT of A is the k -dimensional array $Y[0 : N_1 - 1, 0 : N_2 - 1, \dots, 0 : N_k - 1]$, where

$$Y[\beta_1, \beta_2, \dots, \beta_k] = \sum_{\alpha_1=0}^{N_1-1} \sum_{\alpha_2=0}^{N_2-1} \dots \sum_{\alpha_k=0}^{N_k-1} \omega_{N_1}^{\beta_1 \alpha_1} \omega_{N_2}^{\beta_2 \alpha_2} \dots \omega_{N_k}^{\beta_k \alpha_k} A[\alpha_1, \alpha_2, \dots, \alpha_k] .$$

The *Fast Fourier Transform*, or FFT, is a method of computing the Discrete Fourier Transform. For more details, see [CLR90].

1.2 The Parallel Disk Model

Throughout this paper, we assume that we are using the Parallel Disk Model (PDM) described by Vitter and Shriver [VS94].

In the PDM, we have the following parameters:

1. N is the total number of *records* on disk.
2. M is the number of records that can fit in the *internal memory*. We assume that all records are the same size.
3. B is the number of records stored in each *block*.
4. D is the number of *disks*, which are denoted by $\mathcal{D}_0, \mathcal{D}_1, \dots, \mathcal{D}_{D-1}$.
5. P is the number of *processors*, which are denoted by $\mathcal{P}_0, \mathcal{P}_1, \dots, \mathcal{P}_{P-1}$.

	\mathcal{P}_1								\mathcal{P}_2							
	\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 1.1: The layout of $N = 64$ records in a parallel disk system with $P = 2$, $B = 2$, and $D = 8$. Each box represents one block. The number of stripes is $N/BD = 4$. Numbers indicate record indices.

We assume that $1 \leq B \leq M/2$, $1 \leq D \leq \lfloor M/B \rfloor$, and $P \leq D$. Furthermore, $M < N$ (or we could read all the data into memory and perform our operations in-core). Each processor \mathcal{P}_i has access to the D/P disks $\mathcal{D}_{iD/P}, \mathcal{D}_{iD/P+1}, \dots, \mathcal{D}_{(i+1)D/P-1}$. Each processor also has internal memory capable of holding M/P records.

The data are stored evenly across the D disks, with N/D records on each disk. Each disk is partitioned into N/BD blocks, with B records per block. The layout is shown in Figure 1.1. We refer to the set of the i th block on all disks as *stripe* i . When data are transferred to or from a disk, an entire block must be read or written. A parallel I/O operation can read or write at most one block from each disk, which is at most BD records in total. We use *independent I/Os*, in which the blocks accessed can be at any location on each of the disks, so we do not have to access an entire stripe at once.

When we refer to a record, we refer to it by *index* or *address*. Each index is an n -bit vector $x = (x_0, x_1, \dots, x_{n-1})$. The least significant b bits, x_0, x_1, \dots, x_{b-1} , represent a record's offset within a block. The next d least significant bits, $x_b, x_{b+1}, \dots, x_{b+d-1}$ represent the disk number. Note that the bits $x_{b+d-1-p}, x_{b+d-p}, \dots, x_{b+d-1}$ indicate which processor can access the record. Finally the remaining $n - (b + d)$ bits, $x_{b+d}, x_{b+d+1}, \dots, x_{n-1}$, indicate the stripe number.

Throughout this paper, we will refer to $n = \lg N$, $m = \lg M$, $b = \lg B$, $d = \lg D$, and $p = \lg P$. We assume that each of n, m, b, d , and p are nonnegative integers, which means that N, M, B, D , and P are powers of 2.

We measure the efficiency of an algorithm by the number of I/O operations required. Since a parallel I/O operation can only access BD records, an algorithm that accesses all N records requires $\Omega(N/BD)$ parallel I/O operations. Thus, $O(N/BD)$ is analogous to linear time in sequential algorithms.

1.3 BMMC Permutations

A BMMC is characterized by a nonsingular (or invertible) $n \times n$ *characteristic matrix* $A = (a_{ij})$, where $a_{ij} \in \{0, 1\}$. The specification also includes a complement vector $c = (c_0, c_1, \dots, c_{n-1})$. Treating a *source address* x as an n -bit vector, we perform matrix-vector multiplication over $GF(2)$ and complement some subset of the resulting bits to form the corresponding n -bit *target index* $y = Ax \oplus c$. For our purposes in this paper, we that assume that the complement vector c is the 0 vector, leaving us with $y = Ax$. Because A is nonsingular, this mapping of source to target addresses is one-to-one.

BMMC permutations are closed under composition. That is to say, if we wish to perform the the BMMC permutations characterized by matrices A_1, A_2, \dots, A_k in that order, we could instead perform the single BMMC permutation characterized by the product $A = A_k A_{k-1} \dots A_1$.

The universal lower bound for BMMC permutations on the PDM is presented in [CSW99]. Any

nonidentity BMMC permutation requires

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

parallel I/Os, where γ is the lower left $\lg(N/B) \times \lg B$ submatrix of the characteristic matrix, and where the rank is computed over $GF(2)$. Note that because γ is a $\lg(N/B) \times \lg B$ matrix, its rank is at most $\lg \min(B, N/B)$.

Also included in [CSW99] is an asymptotically optimal algorithm for performing BMMC permutations. This algorithm requires at most $\frac{2N}{BD}\left(\left\lceil \frac{\text{rank } \phi}{\lg(M/B)} \right\rceil + 1\right)$ parallel I/Os, where ϕ is the lower left $\lg(N/M) \times \lg M$ submatrix of the characteristic matrix. Note that rank ϕ is at most $\lg \min(M, N/M)$.

1.4 Some BMMC permutation matrices

The algorithms for the Dimensional Method use several BMMC permutations. The following are all *bit-permute/complement*, or *BPC*, permutations. A BPC permutation has a characteristic matrix with exactly one 1 in each row and column. Essentially, a BPC permutation forms each target address by applying some fixed permutation to the source address bits.

***x*-bit right-rotation:**

We rotate the bits of each index by x bits to the right, wrapping around the rightmost bit. The characteristic matrix has the form

$$\left[\begin{array}{c|c} x & n-x \\ \hline 0 & I \\ \hline I & 0 \end{array} \right] \begin{array}{l} n-x \\ x \end{array} .$$

Stripe-major to processor-major and vice-versa:

The standard PDM layout of the data is in *stripe-major order*. In this layout, each processor only has access to small sets of contiguous data of only BD/P records. It is often easier to deal with the data in *processor-major order*, in which each processor has access to N/P consecutive records. Processor \mathcal{P}_i has access to the N/P records with indices $i(N/P)$ to $(i+1)(N/P) - 1$. The characteristic matrices to reorder from stripe-major to processor major order and vice-versa follow, where $s = b + d$:

$$\left[\begin{array}{c|c|c} s-p & n-s & p \\ \hline I & 0 & 0 \\ \hline 0 & 0 & I \\ \hline 0 & I & 0 \end{array} \right] \begin{array}{l} s-p \\ p \\ n-s \end{array} ,$$

stripe-major to processor-major

$$\left[\begin{array}{c|c|c} s-p & p & n-s \\ \hline I & 0 & 0 \\ \hline 0 & 0 & I \\ \hline 0 & I & 0 \end{array} \right] \begin{array}{l} s-p \\ n-s \\ p \end{array} .$$

processor-major to stripe-major

1.5 Technique for BMCC complexity analysis

When we analyze the complexity of BMCC permutations, we are usually concerned with finding the rank of ϕ , the lower left $\lg(N/M) \times \lg M$ submatrix of the characteristic matrix A . The submatrix ϕ is equivalent to the matrix product $\phi = XAY$, where X and Y have the forms

$$X = \left[\begin{array}{c|c} & \\ \hline 0 & I \end{array} \right]_{n-m} \quad \text{and} \quad Y = \left[\begin{array}{c} \\ \hline I \\ 0 \end{array} \right]_{n-m}^m .$$

We can view X as a row selection matrix and Y as a column selection matrix. Clearly $\phi = XAY$ because XA selects the lower $\lg(N/M)$ rows of A , and AY selects the leftmost $\lg M$ columns of A . Thus computing the rank of XAY is equivalent to computing the rank of ϕ .

If we permute the rows and columns of XAY , its rank remains the same. Thus, if we let Π_1 be any $\lg(N/M) \times \lg(N/M)$ permutation matrix (with exactly one 1 in each row and column) and Π_2 be any $\lg M \times \lg M$ permutation matrix, then the rank of the product $\Pi_1 XAY \Pi_2$ is equivalent to rank ϕ .

Note also that we can perform the matrix products in any order. We use Π_1 and Π_2 to permute the subproducts for easier computation.

Chapter 2

The Dimensional Method

2.1 The Basic Dimensional Method

The Dimensional Method for computing multidimensional out-of-core FFTs as outlined by Lauren Baptist [Bap99] is relatively straightforward. Multidimensional FFTs can be computed by computing the one-dimensional FFTs of each dimension in turn. We assume that we are working with a PDM and that data are stored contiguously on the first dimension. Because the subsequent dimensions are not in contiguous memory locations, we reorder the data using BMMC permutations between FFT computations to minimize I/O costs.

We assume that the one-dimensional FFTs for each dimension fit into the memory of a single processor. That is, for each dimension j , we assume that $N_j \leq M/P$, or $n_j \leq m - p$. On each read, each processor can hold the data of $(M/P)/N_j$ one-dimensional FFTs. We can then compute these FFTs in-core and write the results back to disk. If the current dimension is stored in contiguous addresses in processor-major order, we can do these in-core FFT computations with exactly one pass through the data.

We can use the Cooley-Tukey method for computing one-dimensional FFTs in-core.

The data starts in stripe-major ordering with the first dimension in contiguous memory locations. Thus, before doing the first FFTs, we must first arrange the data into processor-major ordering. After computing the dimension- j FFTs, we want to reorder the data so that dimension $j + 1$ is in contiguous addresses. This reordering entails an n_j -bit right rotation on the data *when in stripe-major order*. After we finish with the computations for the dimension- k FFTs, we need to put the data back in the order given in the beginning.

We need the following BMMC permutation matrices:

- S characterizes the stripe-major to processor-major permutation.
- S^{-1} characterizes the processor-major to stripe-major permutation.
- R_{n_j} characterizes an n_j -bit right rotation permutation.

Our algorithm is as follows:

1. Though dimension 1 is in contiguous addresses, the data are in stripe-major ordering, so we perform the BMMC permutation characterized by S .
2. Between computing the FFTs for dimensions j and $j + 1$, we want to rearrange the data so that dimension $j + 1$ is in contiguous locations. Thus, we perform the BMMC permutation characterized by the matrix product $SR_{n_j}S^{-1}$.

3. After computing the dimension- k FFTs, we must move the data back to the original ordering with the BMMC permutation characterized by the product $R_{n_k} S^{-1}$.

Note that this algorithm ignores the n_j -bit partial bit-reversal permutations presented in the original method. The Cooley-Tukey method begins the FFT computation with a bit-reversal followed by some butterfly-operations. These bit reversals can be performed in-core (since we assume that each dimension fits in the memory of a single processor), and they do not affect the I/O complexity. For simplicity, therefore, we omit them for now.

2.2 Optimization

There is no reason, however, that the one-dimensional FFTs must be computed in the order given. Working with the data in a different order may result in fewer I/O operations. Furthermore, we do not always have to read only one dimension into memory at a time. When we compute the FFTs on a single dimension, we need one pass through the data for each dimension in addition to the number of passes required for our reorderings. If we can read multiple dimensions into the memory of a single processor at one time, we still have to reorder data, but we reduce the number of passes required to do the computations.

This algorithm permutes the data, then performs the one-dimensional FFTs for one or more dimensions in-core, and then repeats until all dimensions are computed. Upon completion, the data must be reordered back to the initial ordering. We will go into more detail later on, but the problem is now to determine the optimal ordering and grouping of the dimensions so as to minimize I/O.

2.3 NP-Completeness

As it turns out, the problem of determining the optimal ordering and grouping of dimensions to minimize I/O operations is NP-complete.

FFT-ORDERING INSTANCE: Nonnegative integers n, m, b, d, p , a positive integer k , a positive integer C , and a sequence of nonnegative integers $\langle n_1, n_2, \dots, n_k \rangle$ representing the size and orderings of dimensions in the data, where $n = n_1 + n_2 + \dots + n_k$.

QUESTION: Is there a sequence and grouping of the dimensions and a sequence of permutations $\langle A_0, A_1, \dots, A_k \rangle$ such that we can perform the multidimensional FFT with no more than C passes through the data using the Dimensional Method with the permutations A_i ? More formally, is there a sequence $\langle G_1, G_2, \dots, G_z \rangle$ where each $G_i = \{g_{i_1}, g_{i_2}, \dots, g_{i_{q_i}}\}$ is a set of dimension numbers, such that our Dimensional Method takes no more than C passes through the data when performing the permutation A_i following the computation of FFTs in G_i ? Note that the modified Dimensional Method will permute the data so that FFTs for dimensions in G_i can be performed with a single pass through the data using A_{i-1} , then perform the FFTs for these dimensions in-core, and then permute for the dimensions in G_{i+1} with A_{i-1} .

Claim 1 *FFT-ORDERING is NP-complete.*

Proof: First, we need to show that FFT-Ordering is in NP. This is fairly obvious. Given a series of groupings $\langle G_1, G_2, \dots, G_z \rangle$ and a series of permutations $\langle A_0, A_1, \dots, A_k \rangle$, we can verify whether the Dimensional Method can be performed in C passes through the data.

Next we need to show that FFT-Ordering is NP-hard. Reduction from Bin Packing follows.

BIN-PACKING INSTANCE: A finite set U of items, a size $s(u) \in \mathbf{Z}^+$ for each $u \in U$, a positive integer V representing bin capacity, and a positive integer K .

QUESTION: Can U be divided into K bins of capacity V ? That is to say, is there a partition of U into U_1, U_2, \dots, U_K such that $\sum_{u \in U_i} s(u) \leq V$ for all $1 \leq i \leq K$?

Given an instance to BIN-PACKING, we can reduce to an instance of the FFT-ORDERING decision problem in polynomial time as follows. Set $p = 0, b = 0, m = V, d = 0, k = |U|, n = \sum_{u \in U} s(u)$. The ordering of $u \in U$ does not matter, so let us assign some canonical form $U = \{u_1, u_2, \dots, u_{|U|}\}$ to our original instance. We set $n_i = s(u_i)$. Finally, set $C = K$. This reduction obviously takes polynomial time.

We need to show that the instance to BIN-PACKING causes a “yes” if and only if the reduction to FFT-ORDERING produces a “yes.”

(\implies) Suppose the instance to BIN-PACKING is solvable. Then there exist U_1, U_2, \dots, U_K such that $\sum_{u \in U_i} s(u) \leq V$. If we make $G_i = \{x : u_x \in U_i\}$ (the ordering does not matter), each G_i represents a group of dimensions that can be read into memory of the single processor at once. How do we know?

$$\begin{aligned} \sum_{u \in U_i} s(u) &\leq V \\ \sum_{x \in G_i} s(u_x) &\leq V \\ \sum_{x \in G_i} n_x &\leq V \\ 2^{\sum_{x \in G_i} s(u_x)} &\leq 2^V \\ \prod_{x \in G_i} N_x &\leq M. \end{aligned}$$

Thus, we can read the data for the multiple dimensions into memory at the same time, and our grouping is valid. By the Dimensional Method, we perform some BMMC permutation to rearrange the data in order to access the dimensions in G_i , and then we perform the FFT computations for these dimensions. In general, a parallel I/O operation reads one block from each disk, or BD records in total. If $B = 1$ and $D = 1$, as it does in this reduction, each I/O operation reads exactly 1 record from disk. Thus, we can load *any* M records into memory with exactly M I/O operations, regardless of how the data are stored on disk. Therefore, we can perform the identity permutation, with an I/O cost of 0, between each group of FFT computations. As for the FFT computations themselves, since we can read any M records into memory with M read operations, we still only need one pass through the data to perform all the FFTs in-core for the dimensions in G_i . Thus, the total I/O cost is $K = C$ passes through the data.

(\impliedby) Now suppose we can solve our instance of the FFT-ORDERING. That means that there is a grouping/ordering (G_1, G_2, \dots, G_z) such that the modified Dimensional Method takes no more than C passes through the data. We create bins U_1, U_2, \dots, U_z where $U_i = \{u_x : x \in G_i\}$. By the same relationship used in (\implies), it must be true that $\sum_{u \in U_i} s(u) \leq V$. Thus we have a valid packing into z bins. Since we have to bin pack into K bins, we just need to verify that $z \leq K$. We need z passes through the data to compute the FFTs for the dimensions in each of the groups, and we need at least 0 passes to permute between the groupings. Thus $C \geq z$. Since $C = K$ by the reduction, $z \leq K$. We conclude that the instance of Bin Packing also has the solution $U_i = \{u_x : x \in G_i\}$. ■

2.4 Grouping Dimensions

Now, let us look at the problem of grouping dimensions without doing any reordering. Suppose we are given dimensions $1, 2, \dots, k$ of sizes N_1, N_2, \dots, N_k . If there exist i and j , $i < j$, such that $\sum_{x=i}^j n_x \leq m - p$, then we can read data for multiple one-dimensional FFTs into the memory of

a single processor and perform the computations in-core. Thus, we reduce the number of passes through the data. The question arises whether we can determine the optimal way of grouping dimensions.

If we use the simple method — without grouping dimensions at all — we perform an n_i -bit right-rotation permutation after working with dimension i to get dimension $i+1$ into the contiguous memory locations. If, however, we read dimensions $i, i+1, \dots, j$ into memory at the same time, we can perform an $(\sum_{x=i}^j n_x)$ -bit right-rotation permutation to get to the next dimension. Since we can only read multiple dimensions into memory at the same time if $\sum_{x=i}^j n_x \leq m-p$, the cost analysis is identical to that in Baptist's thesis .

Let us assume that we are trying to group dimensions into the sequence $\langle G_1, G_2, \dots, G_z \rangle$ where each G_i is a set of consecutive dimensions. We can then do the following operations:

1. Before we do any FFT computation, perform the BMMC permutation S .
2. After performing FFT computations for dimensions in $G_i = \{g_{i_1}, g_{i_2}, \dots, g_{i_y}\}$, perform the BMMC permutation characterized by the matrix product $SR_{g_{i_y}} R_{g_{i_{y-1}}} \dots R_{g_{i_1}} S^{-1}$.
3. After doing FFT computations on $G_z = \{g_{z_1}, g_{z_2}, \dots, g_{z_y}\}$, perform the BMMC permutation characterized by the matrix product $R_{g_{z_y}} R_{g_{z_{y-1}}} \dots R_{g_{z_1}} S^{-1}$.

Since $\sum_{j \in G_i} n_j \leq m-p$, we can apply the results of Baptist's cost analysis. That is to say, the cost of (1), in passes through the data, is

$$\left(\left\lceil \frac{\min(n-m, p)}{m-b} \right\rceil + 1 \right) .$$

The cost of (2), each time, is

$$\left(\left\lceil \frac{\min\left(n-m, \sum_{x \in G_i} n_x\right)}{m-b} \right\rceil + 1 \right) .$$

The cost of (3) is

$$\left(\left\lceil \frac{\min\left(n-m, \sum_{x \in G_z} n_x + p\right)}{m-b} \right\rceil + 1 \right) .$$

The problem exhibits optimal substructure. Let us suppose we are given an optimal grouping $\langle G_1, G_2, \dots, G_z \rangle$ for dimensions $1, 2, \dots, k$. Dimension k belongs to the group of dimensions $G_z = \{j, j+1, \dots, k\}$. It must be the case that $\langle G_1, G_2, \dots, G_{z-1} \rangle$ is an optimal solution to the problem of grouping dimensions $1, \dots, j-1$. Suppose there is some solution $\langle S_1, S_2, \dots, S_y \rangle$ that uses fewer I/O operations than $\langle G_1, G_2, \dots, G_{z-1} \rangle$. Then we would be able to create the groupings $\langle S_1, S_2, \dots, S_y, G_z \rangle$ to take fewer I/O operations than our optimal solution $\langle G_1, \dots, G_z \rangle$, which yields a contradiction.

The total cost of the Dimensional Method algorithm consists of both the costs of (1), (2) and (3) to permute that data between FFT computations, and the cost of the FFT computations themselves. For each group G_i , we require one pass through the data to compute the FFTs.

Next, we can develop a recursive definition for the cost C_i of an optimal solution for dimensions $1, \dots, i$. This definition is relatively straightforward. The cost always includes the cost of (1) (assuming we have at least one dimension), and so we set C_0 to the cost of (1). If $1 \leq i < k$, then C_i can be calculated by going through all possible groupings for the last group and adding the resultant cost of (2) plus one pass (for the FFT computations of this group) to the best way of doing all the

other groupings. We use similar logic if $i = k$ except we use the cost of (3) instead of (2). We get the following as our cost function:

$$C_i = \begin{cases} \left\lceil \frac{\min(n-m,p)}{m-b} \right\rceil + 1 & \text{if } i = 0, \\ \min_{\substack{j \text{ s.t. } 1 \leq j \leq i \\ \text{and } \sum_{x=j}^i n_x \leq m-p}} \left\{ \left\lceil \frac{\min\left(n-m, \sum_{x=j}^i n_x\right)}{m-b} \right\rceil + 2 + C_{j-1} \right\} & \text{if } 1 < i < k, \\ \min_{\substack{j \text{ s.t. } 1 \leq j \leq i \\ \text{and } \sum_{x=j}^i n_x \leq m-p}} \left\{ \left\lceil \frac{\min\left(n-m, \sum_{x=j}^i n_x + p\right)}{m-b} \right\rceil + 2 + C_{j-1} \right\} & \text{if } i = k. \end{cases}$$

Pseudocode for the algorithm may look like:

```

1  $C[0] \leftarrow \left\lceil \frac{\min(n-m,p)}{m-b} \right\rceil + 1$ 
2 for  $i \leftarrow 1$  to  $k$  do
3    $C[i] \leftarrow \infty$ 
4   for  $i \leftarrow 1$  to  $k-1$  do
5      $j \leftarrow i$ 
6      $s \leftarrow n_j$ 
7     while  $s \leq m-p$  and  $j \geq 1$  do
8        $C[i] \leftarrow \min\left(C[i], \left\lceil \frac{\min(n-m,s)}{m-b} \right\rceil + 2 + C[j-1]\right)$ 
9        $j \leftarrow j-1$ 
10       $s \leftarrow s + n_j$ 
11   $j \leftarrow k$ 
12   $s \leftarrow n_k$ 
13  while  $s \leq m-p$  and  $j \geq 1$  do
14     $C[k] \leftarrow \min\left(C[k], \left\lceil \frac{\min(n-m,s+p)}{m-b} \right\rceil + 2 + C[j-1]\right)$ 
15     $j \leftarrow j-1$ 
16     $s \leftarrow s + n_j$ 

```

The nested loop runs in $O(k^2)$ time, and so the whole algorithm has a run time of $O(k^2)$.

2.5 Ordering Dimensions

There is also no reason why we have to compute the FFTs in the order given. In many cases, it takes fewer passes through the data if we compute the dimensions in a different order. For example, take the simple case with $M = 2^{11}$, $D = 2^5$, $P = 2^4$, $B = 2^5$, $k = 6$, $N = 2^{20}$, and the dimensions are of the following sizes: $N_1 = 2^2$, $N_2 = 2^7$, $N_3 = 2^2$, $N_4 = 2^3$, $N_5 = 2^3$, $N_6 = 2^3$. Our general assumptions about the problem are true here, so this example is valid. Doing the dimensions in order according to the simple Dimensional Method yields the cost of 21 passes through the data. If, however, we compute the FFTs in the order of dimensions 1, 2, 6, 4, 5, 3, then we only need 19 passes through the data. Between dimensions 2 and 6, we do a 15-bit right-rotation. Between dimensions 6 and 4, we do a 14-bit right-rotation. Between dimensions 5 and 3, we do a 15-bit right rotation. We finish with an 11-bit right-rotation to get the data back to the original order. The rest is pretty obvious.

Let us assume for now that we are only going to read one dimension into memory at a time, but that we can do the one-dimensional FFT computations in any order. If we wish to do the dimensions in the order $\langle x_1, x_2, \dots, x_k \rangle$, we can use the following modified algorithm:

1. Before doing any FFT computations, permute to the starting dimension and convert from stripe-major to processor-major order with the BMMC permutation characterized by SR_{1,x_1} .

2. After performing the FFT computations for dimension x_i , we must perform the BMBC permutation characterized by $SR_{x_i, x_{i+1}}S^{-1}$.
3. After doing the FFT computations for the dimension x_k , perform the BMBC permutation characterized by the matrix product $R_{x_k, 1}S^{-1}$.

where $R_{i,j}$ is defined as follows:

$$R_{i,j} = \begin{cases} \left(\sum_{x=i}^{j-1} n_x \right)\text{-bit right-rotation matrix} & \text{if } i \leq j, \\ \left(n - \sum_{x=i}^{j-1} n_x \right)\text{-bit right-rotation matrix} & \text{if } i > j. \end{cases}$$

2.6 Analysis of Permutations

Before we can determine what the optimal ordering is, we have to determine the I/O complexity of each of the permutation matrices we will need. Any BMBC permutation can be performed in $\frac{2N}{BD} \left(\left\lceil \frac{\text{rank } \phi}{\lg(M/B)} \right\rceil + 1 \right)$ parallel I/O operations, where ϕ is the lower left $(n-m) \times m$ submatrix of the characteristic matrix. The algorithm to solve the one-dimensional FFTs in a different order, without grouping, requires the characteristic matrices represented by the matrix products SR_x , SR_xS^{-1} , and R_xS^{-1} . The analyses of these matrices differs from Lauren Baptist's analyses [Bap99] in that x is now only limited to $0 \leq x \leq n$ instead of $0 \leq x \leq m-p$. We need the following results before we can prove anything regarding the optimal ordering.

Lemma 1 *For the matrix product SR_x , we have*

$$\text{rank } \phi = \min(n-m, m, n-x+p, |x-p|).$$

Proof: Computing $\text{rank } \phi$ is equivalent to computing the rank of the $(n-m) \times m$ matrix product $Z = XSR_xY$. We can group these factors however we choose, so we will compute $E = XS$ and $F = R_xY$, then $Z = EF$. We can represent this grouping as

$$\underbrace{\overbrace{XS}^E \overbrace{R_xY}^F}_Z$$

We begin by finding the subproduct $E = XS$, where the matrices have the forms

$$X = \left[\begin{array}{c|c} m & n-m \\ \hline 0 & I \end{array} \right]_{n-m} \quad \text{and} \quad S = \left[\begin{array}{c|c|c} s-p & n-s & p \\ \hline I & 0 & 0 \\ \hline 0 & 0 & I \\ \hline 0 & I & 0 \end{array} \right]_{\begin{matrix} s-p \\ p \\ n-s \end{matrix}}.$$

In the PDM, we assume that $BD \leq M$, or $s = b + d \leq m$. We can view X as a row selection of S , selecting the bottom $n-m$ rows. Given that $s \leq m$, we know $n-m \leq n-s$, so

$$E = \left[\begin{array}{c|c|c} m-p & n-m & p \\ \hline 0 & I & 0 \end{array} \right]_{n-m}.$$

Next, we need the subproduct $F = R_xY$, where R_x and Y have the forms

$$R_x = \left[\begin{array}{c|c} x & n-x \\ \hline 0 & I \\ \hline I & 0 \end{array} \right]_{\begin{matrix} n-x \\ x \end{matrix}} \quad \text{and} \quad Y = \left[\begin{array}{c} m \\ \hline I \\ \hline 0 \end{array} \right]_{\begin{matrix} m \\ n-m \end{matrix}}.$$

From this point on, we have two cases: $x < m$ and $x \geq m$. We examine each case separately.

1. **Suppose** $x < m$. We can view Y as a column selection of R_x , selecting the leftmost m columns. Since $x < m$, these leftmost m columns includes the leftmost x columns of R_x . So we have that F is

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

Now we can compute the rank of the product $Z = EF$. Note that E is

$$E = \left[\begin{array}{c|c|c} m-p & n-m & p \\ \hline 0 & I & 0 \end{array} \right]_{n-m} .$$

We can view E as a row selection of F . Notice that if $x = p$, then the $n - m$ rows selected are the 0 rows in the middle of F , so the rank is 0. Increasing x up to $m - 1$ slides the band of 0s in F up, so E selects up to $x - p$ rows with 1s from the bottom x rows of F . If, on the other hand, x decreases from p , the band of 0s in F slides down, so E selects up to $p - x$ rows with 1s from the top $m - x$ rows of F . Since we only select $n - m$ rows, the rank can be at most $n - m$. Thus we have $\text{rank } \phi = \min(n - m, |x - p|)$.

Since we assumed that $x < m$, we know that $|x - p| \leq m$, so we can restate this rank as $\text{rank } \phi = \min(n - m, m, |x - p|)$. Furthermore, if $x < m$, then $n - x + p > n - m + p \geq n - m$, so we can end with $\text{rank } \phi = \min(n - m, m, n - x + p, |x - p|)$.

2. **Suppose** $x \geq m$. Recall, we are computing the product $F = R_x Y$. Again, we view Y as a column selection of R_x , selecting the leftmost m columns. In this case, the leftmost m columns fall within the the leftmost x columns. So we have

$$F = R_x Y = \left[\begin{array}{c} m \\ \hline 0 \\ \hline I \\ \hline 0 \end{array} \right] \begin{array}{l} n-x \\ m \\ x-m \end{array} .$$

Now we can look at the rank of the final product $Z = EF$. Let us divide this product further into a few cases.

- (a) **Suppose that** $x \geq m$ **and** $m \leq n - m$. Then we can treat F as a column selection of E .

Our assumption that $m \leq n - m$ implies that $m + p \leq n - m + p$. Notice that if $m + p \leq x \leq n - (m - p)$, then there are at least $m - p$ 0 rows on the top of F , and there are at least p 0 rows on the bottom of F . Thus, the m columns of E selected fall within the $n - m$ columns with 1s, and the rank is m .

When x falls below $m + p$, the band of 0s on the bottom of F decreases below p , so F selects columns with only 0s from E . Thus, the rank decreases. We see there is a linear relationship here. When $x = m + p$, the rank is m . When $x = m$, the rank is $m - p$. Within this range, the rank is $x - p$.

When x increases past $n - (m - p)$, the band of 0s on the top of F decreases to below $m - p$ rows. Thus, F selects some columns of 0s from the left of E . Again, we see a linear relationship here. When $x = n - (m - p)$, we select m 1s. When x is at its maximum of $x = n$, we only select p 1s. We can describe this linear relationship as $n - (x - p)$ 1s being selected.

To sum up, the product $Z = EF$ product gives us

$$\text{rank } \phi = \begin{cases} x - p & \text{if } x \leq m + p, \\ m & \text{if } m + p < x \leq n - (m - p), \\ n - (x - p) & \text{if } x > n - (m - p). \end{cases}$$

In no case are more than m 1s selected, and $m \leq n - m$ by assumption. Thus, we actually have

$$\text{rank } \phi = \begin{cases} \min(m, x - p) & \text{if } x \leq m + p, \\ m & \text{if } m + p < x \leq n - (m - p), \\ \min(m, n - (x - p)) & \text{if } x > n - (m - p). \end{cases}$$

We can reduce this piecewise function into a single expression. Since we assumed that $m \leq n - m$, we have $m \leq n/2$. Therefore, when $x \leq m + p$ it follows that $x - p \leq m \leq n/2$, so $x - p \leq n - (x - p)$. Furthermore, $m + p < x \leq n - (m - p)$ implies $m < x - p$ and $m \leq n - (x - p)$. Finally, when $x > n - m + p$, we know that $x > m + p$ (or $x - p > m$) by our assumption that $n - m \geq m$. Thus, we can combine everything in our piecewise function to get $\text{rank } \phi = \min(m, n - x + p, x - p)$.

Because of our assumption that $m \leq n - m$, we can add an $n - m$ term to this minimum. Furthermore, $x > m$ by our other assumption implies that $x > p$, so $|x - p| = x - p$. Thus, we can restate $\text{rank } \phi = \min(n - m, m, n - x + p, |x - p|)$.

- (b) **Suppose that $x \geq m$ and $m > n - m$.** Recall that we are computing the product $Z = EF$ where E and F have the forms

$$E = \left[\begin{array}{ccc|ccc} m-p & n-m & p & & & \\ 0 & I & 0 & & & \end{array} \right]_{n-m} \quad \text{and} \quad F = \begin{bmatrix} 0 \\ I \\ 0 \end{bmatrix} \begin{matrix} n-x \\ m \\ x-m \end{matrix}.$$

We can view E as a row selection of F .

If $n - m + p \leq x \leq m + p$, then there are fewer than $m - p$ rows of 0s on the top of F and fewer than p rows of 0s on the bottom of F . Thus, the $n - m$ rows selected fall within the $n - m$ rows containing 1s. Thus, when x falls within this range, we have the rank is $n - m$.

When x decreases below $n - m + p$, the number of rows of 0s across the top of F increases above $m - p$. Thus, E begins to select 0s from F . We observe a linear relationship here when x is between m and $n - m + p$. When $x = m$, E only selects $m - p$ 1s. When $x = n - m + p$, E selects $n - m$ 1s from F . Thus, E selects $x - p$ 1s when x falls within this range.

When x increases above $m + p$, the band of 0s across the bottom of F increases to be more than p rows, so E selects rows of 0s from F . Again, when $x = m + p$, E selects $n - m$ 1s, and when $x = n$, E selects p 1s. Thus, we can describe this relationship as $n - (x - p)$.

Since E never selects more than $n - m$ rows from F , we have

$$\text{rank } \phi = \begin{cases} \min(n - m, n - (x - p)) & \text{if } x > m + p, \\ n - m & \text{if } n - m + p < x \leq m + p, \\ \min(n - m, x - p) & \text{if } x \leq n - m + p. \end{cases}$$

It is again useful to combine this piecewise function. When $x > m + p$, $x - p > m$. Our assumption that $m > n - m$ implies that $n - m < n/2$. Therefore $x \leq n - m + p$ implies $x - p \leq n - m < n/2$, so $n - (x - p) \geq n/2 > n - m$. Finally, $n - m + p < x \leq m + p$

implies $n - m < x - p$ and $n - x \geq n - (m + p)$ which means $n - x + p \geq n - m$. Thus, we have $\text{rank } \phi = \min(n - m, n - x + p, x - p)$

By assumption, $n - m > m$. Furthermore, $x \geq m$ implies $x \geq p$, so $x - p = |x - p|$. Thus, we can rewrite $\text{rank } \phi = \min(n - m, m, n - x + p, |x - p|)$.

All cases yielded the same result, so we have $\text{rank } \phi = \min(n - m, m, n - x + p, |x - p|)$. ■

Lemma 2 For the matrix product SR_xS^{-1} , we have

$$\text{rank } \phi = \min(n - m, m, x, n - x) .$$

Proof: Computing $\text{rank } \phi$ is equivalent to computing the rank of the $(n - m) \times m$ matrix product $Z = XSR_xS^{-1}Y\Pi_1\Pi_2$. Again, we can set Π_1 and Π_2 later to make calculations easier. We can group these factors however we choose, so we will compute $E = XS$ (which we already have above) and $G = S^{-1}Y$. We can then compute $H = G\Pi_1$ followed by $J = R_xH$ and $K = J\Pi_2$. Finally, $Z = EK$. We can represent this grouping as

$$\underbrace{\underbrace{\underbrace{\underbrace{\underbrace{XS}_{E} R_x}_{G} S^{-1}Y}_{H} \Pi_1}_{J} \Pi_2}_{K}}_{Z} .$$

First, we compute the subproduct $G = S^{-1}Y$ where S and Y have the forms

$$S^{-1} = \left[\begin{array}{c|c|c} s-p & p & n-s \\ \hline I & 0 & 0 \\ \hline 0 & 0 & I \\ \hline 0 & I & 0 \end{array} \right] \begin{array}{l} s-p \\ n-s \\ p \end{array} \quad \text{and} \quad Y = \left[\begin{array}{c} m \\ \hline I \\ \hline 0 \end{array} \right] \begin{array}{l} m \\ n-m \end{array} .$$

The work here is identical to Lauren Baptist's. We can think of Y as a column selection of S^{-1} , selecting the leftmost m columns of S^{-1} . Since we know that $s \leq m$, the $n - m$ columns not selected will fall within the rightmost $n - s$ columns. Thus, we have the matrix G of the form

$$G = \left[\begin{array}{c|c|c} s-p & p & m-s \\ \hline I & 0 & 0 \\ \hline 0 & 0 & I \\ \hline 0 & 0 & 0 \\ \hline 0 & I & 0 \end{array} \right] \begin{array}{l} s-p \\ m-s \\ n-m \\ p \end{array} .$$

We can next look at the matrix subproduct $H = G\Pi_1$. Π_1 can be any $m \times m$ permutation matrix. To make later products easier to compute, we want to move the rightmost $m - s$ columns to follow the leftmost $s - p$ columns. If we set Π_1 to be the column permutation matrix

$$\Pi_1 = \left[\begin{array}{c|c|c} s-p & m-s & p \\ \hline I & 0 & 0 \\ \hline 0 & 0 & I \\ \hline 0 & I & 0 \end{array} \right] \begin{array}{l} s-p \\ p \\ m-s \end{array} ,$$

then the product $G\Pi_1$ is

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

The next subproduct is $J = R_x H$, where R_x looks like

$$R_x = \left[\begin{array}{c|c} x & n-x \\ \hline 0 & I \\ \hline I & 0 \\ \hline x & \end{array} \right] \begin{array}{l} n-x \\ x \end{array} .$$

From here on, it is easier to look at three separate cases: $x < m - p$, $m - p \leq x < n - p$, and $x \geq n - p$.

1. **Suppose** $x < m - p$. There is no need to continue. Baptist's work made this assumption and shows that $\text{rank } \phi = \min(n - m, x)$. The only difference is that we dropped the n_j -bit partial bit-reversal permutation, but the rank is unaffected by this change to the product. Since $x < m - p$, $x < m$, so we can rewrite the rank as $\text{rank } \phi = \min(n - m, m, x, n - x)$.
2. **Suppose** $m - p \leq x < n - p$. We can view H as a column selection of the leftmost $m - p$ and rightmost p columns of R_x . Since $x \geq m - p$, the leftmost $m - p$ columns falls within the lower left identity matrix. Since $x \leq n - p$, we have $n - x \geq p$, so the rightmost p columns fall within the upper-right identity matrix of R_x . Thus, we have the subproduct

$$J = \left[\begin{array}{c|c} m-p & p \\ \hline 0 & 0 \\ \hline 0 & I \\ \hline I & 0 \\ \hline 0 & 0 \\ \hline p & \end{array} \right] \begin{array}{l} n-x-p \\ p \\ m-p \\ x-(m-p) \end{array} .$$

We can reorder the two columns with the matrix product $K = J\Pi_2$, so let us combine the to identity submatrices by setting Π_2 to be

$$\Pi_2 = \left[\begin{array}{c|c} p & m-p \\ \hline 0 & I \\ \hline I & 0 \\ \hline p & \end{array} \right] \begin{array}{l} m-p \\ p \end{array} .$$

The resulting matrix product is $K = J\Pi_2$

$$K = \left[\begin{array}{c} m \\ \hline 0 \\ \hline I \\ \hline 0 \end{array} \right] \begin{array}{l} n-p-x \\ m \\ x-(m-p) \end{array} .$$

Finally, $\text{rank } \phi$ is the rank of the matrix $Z = EK$. We already have E to be

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

- (a) **Suppose that** $m - p \leq x < n - p$ **and** $m \leq n - m$. Then we can treat K as a column select of E .

If $m \leq x \leq n - m$, then there are at least $m - p$ rows of 0s across the top and p rows of 0s across the bottom of K . Thus, m columns that K selects fall within the $n - m$ columns with 1s in E . When x is within this range, the rank is m .

When x increases past $n - m$, the number of 0 rows across the top of K decreases below $m - p$. Thus, K selects some 0 rows from the left of E . We notice a linear relationship here between $x = n - m$ selecting m 1s and $x = n - p$ selecting p 1s. We can denote this relationship by $n - x$ 1s being selected.

When x decreases below m , the number of 0s across the bottom of K decreases below p , so K selects some 0s from the right of E . When $x = m$, K selects m 1s from E . When $x = m - p$, K selects $m - p$ 1s from E . Again, there is a linear relationship when x falls within this range, and we can write the rank as x .

In any case, K only selects m columns, so the rank can be at most m . This gives us the piecewise function

$$\text{rank } \phi = \begin{cases} \min(m, x) & \text{if } x \leq m, \\ m & \text{if } m < x \leq n - m, \\ \min(m, n - x) & \text{if } x > n - m. \end{cases}$$

We assumed that $m \leq n - m$. Therefore, $x \leq m$ implies $x \leq n - m \leq n - x$. Similarly, $x > n - m$ implies $x > m$. When $m < x \leq n - m$, we have $x > m$ and $m \leq n - x$. Thus, we can reduce the piecewise function to $\text{rank } \phi = \min(n - m, m, x, n - x)$.

- (b) **Suppose instead that** $m - p \leq x < n - p$ **and that** $m > n - m$. We can treat E as a row select of K .

If $n - m \leq x \leq m$, we see that K has fewer than p rows of 0s on the bottom and fewer than $m - p$ rows of 0s on the top. Thus, E selects $n - m$ rows with 1s from K .

When x increases past m , the number of 0 rows on the bottom of K increases past p , so E selects some rows of 0s from the bottom K . Notice that when $x = m$, E selects $n - m$ 1s from K , and when $x = n - p$, E selects p 1s from K . Thus, when x falls within this range, the rank is $n - x$.

Decreasing x below $n - m$ increases the number of 0 rows across the top of K past $m - p$. Therefore, E selects some 0s from the top of K . When $x = n - m$, E selects $n - m$ 1s. When $x = m - p$, E selects $m - p$ 1s from K . Thus, E selects x 1s from K .

Because E only selects $n - m$ rows from K , so far have the relationship

$$\text{rank } \phi = \begin{cases} \min(n - m, x) & \text{if } x \leq n - m, \\ n - m & \text{if } n - m < x \leq m, \\ \min(n - m, n - x) & \text{if } x > m. \end{cases}$$

Again, note that if $x \leq n - m$, then $m \leq n - x$. Since we assume that $m > n - m$, we have $n - m < n - x$. Similarly, $x > m$ implies $x > n - m$. Finally, $n - m \leq x < m$ implies that $n - x > n - m$ and $n - m \leq x$. Thus, we have $\text{rank } \phi = \min(n - m, m, x, n - x)$.

3. **Suppose** $x > n - p$. Recall we are interested in the matrix product $J = R_x H$, where R_x and H have the forms

$$R_x = \left[\begin{array}{c|c} x & n - x \\ \hline 0 & I \\ \hline I & 0 \end{array} \right] \begin{matrix} n - x \\ x \end{matrix} \quad \text{and} \quad H = \left[\begin{array}{c|c} m - p & p \\ \hline I & 0 \\ \hline 0 & 0 \\ \hline 0 & I \end{array} \right] \begin{matrix} m - p \\ n - m \\ p \end{matrix}.$$

We can view H as a column selection of R_x , selecting the rightmost p and the leftmost $m - p$ columns of R_x . Since $x > n - p$, $p > n - x$, so the rightmost p columns includes all the upper right identity submatrix of R_x , and part of the lower left identity submatrix. Furthermore, the $m - p$ leftmost columns fall entirely within the lower left identity submatrix of R_x . So we have the product $J = R_x H$ is

$$J = \left[\begin{array}{c|c|c} m-p & x-(n-p) & n-x \\ \hline 0 & 0 & I \\ \hline I & 0 & 0 \\ \hline 0 & 0 & 0 \\ \hline 0 & I & 0 \end{array} \right] \begin{array}{l} n-x \\ m-p \\ n-m \\ x-(n-p) \end{array} .$$

Again, before we compute the product $K = J\Pi_2$, we can set Π_2 to be any $m \times m$ permutation matrix. Since we would like to combine the two identity submatrices above the 0 rows, we let Π_2 be

$$\Pi_2 = \left[\begin{array}{c|c|c} n-x & m-p & x-(n-p) \\ \hline 0 & I & 0 \\ \hline 0 & 0 & I \\ \hline I & 0 & 0 \end{array} \right] \begin{array}{l} m-p \\ x-(n-p) \\ n-x \end{array} ,$$

which causes $K = J\Pi_2$ to be of the form

$$K = \left[\begin{array}{c|c} m-(x-(n-p)) & x-(n-p) \\ \hline I & 0 \\ \hline 0 & 0 \\ \hline 0 & I \end{array} \right] \begin{array}{l} m-(x-(n-p)) \\ n-m \\ x-(n-p) \end{array} .$$

Finally, we're interested in the product $Z = EK$. Recall that E is of the form

$$E = \left[\begin{array}{c|c|c} m-p & n-m & p \\ 0 & I & 0 \end{array} \right]_{n-m} .$$

We can treat E as a row selection of K . If $x = n - p$, E selects all but the bottom p 0 rows from the $n - m$ 0 rows, so that is $p = n - x$ 1s. Increasing x slides the band of 0s up, adding more 0 rows to the selection. $x \leq n$, so the bottom right identity submatrix of K is at most p rows high. We, therefore, see that the rank of Z is $\text{rank } \phi = \min(n - m, m, n - x)$. We assumed that $x > n - p$, which means that $x > n - m$. Thus, we can write $\text{rank } \phi = \min(n - m, m, n - x, x)$.

Thus, combining all three possibilities ($x \leq m - p$, $m - p < x \leq n - p$, and $n - p < x$) consistently give $\text{rank } \phi = \min(n - m, m, n - x, x)$ for any value of x . \blacksquare

Lemma 3 For the matrix product $R_x S^{-1}$, we have

$$\text{rank } \phi = \min(n - m, m, |n - (x + p)|, x + p) .$$

Proof: Again, we want to find the rank of the product $Z = X R_x S Y \Pi_1 \Pi_2$. Let us start by computing the subproduct $G = S^{-1} Y$, then $H = G \Pi_1$. Next, we can compute $J = R_x H$, then $K = J \Pi_2$.

Finally, we can do the product $Z = XK$. We can represent this grouping as follows.

$$\begin{array}{c}
 X R_x \underbrace{S^{-1} Y}_{G} \Pi_1 \Pi_2 . \\
 \underbrace{\hspace{10em}}_H \\
 \underbrace{\hspace{10em}}_J \\
 \underbrace{\hspace{10em}}_K \\
 \underbrace{\hspace{10em}}_Z
 \end{array}$$

We have to work with three cases again.

1. **Suppose** $x \leq m - p$. Then we can follow exactly Lauren Baptist's work, which gives $\text{rank } \phi = \min(n - m, x + p)$. Note that we assumed that $x \leq m - p$, so $x + p \leq m$. Furthermore, $n - (p + x) \geq n - m \geq 0$. So it is also true that $\text{rank } \phi = \min(n - m, m, |n - (x + p)|, x + p)$.
2. **Suppose** $m - p \leq x \leq n - p$. We can compute K in exactly the same manner we did in Lemma 2. The only work to do here is to compute the product $Z = XK$. Recall that X and K are of the forms

$$X = \left[\begin{array}{c|c} m & n - m \\ 0 & I \end{array} \right]_{n - m} \quad \text{and} \quad K = \left[\begin{array}{c} m \\ \hline 0 \\ I \\ \hline 0 \end{array} \right] \begin{array}{l} n - p - x \\ m \\ x - (m - p) \end{array} .$$

Let us view X as a row selection of K , selecting the bottom $n - m$ rows. We see that if $n - p - x = 0$, or $x = n - p$, the bottom $n - m$ rows are all 0s, so the rank is 0. Decreasing x slides the band of 1s down and increases the rank until all m rows are selected. The rank can be at most $n - m$, so the rank of the resulting $(n - m) \times m$ matrix Z is $\min(n - m, m, n - p - x)$. Note that $x \geq m - p$ in this case, so $x + p \geq m$. Also, $n - p - x \geq 0$, so $n - p - x = |n - p - x|$. Therefore, we can say $\text{rank } \phi = \min(n - m, m, |n - p - x|, x + p)$.

3. **Suppose** $x \geq n - p$. We can again compute K in the same way as in Lemma 2. We are left with the matrix product $Z = XK$ where X and K are of the forms

$$X = \left[\begin{array}{c|c} m & n - m \\ 0 & I \end{array} \right]_{n - m}$$

$$\text{and} \quad K = \left[\begin{array}{c|c} m - (x - (n - p)) & x - (n - p) \\ \hline I & 0 \\ \hline 0 & 0 \\ \hline 0 & I \end{array} \right] \begin{array}{l} m - (x - (n - p)) \\ n - m \\ x - (n - p) \end{array} .$$

Let us again view X as a row selection of K , selecting the bottom $n - m$ rows. We see that X will never select ones from the top left identity submatrix. When $x = n - p$, the bottom $n - m$ rows of K are all 0s. Increasing x moves some rows with 1s up from the bottom, increasing the rank of the resulting $(n - m) \times m$ matrix up to p when $x = n$. Here, we have $\text{rank } \phi = \min(n - m, m, p - (n - x))$. By our assumption, it is always true that $x \geq n - p$, or $x + p \geq n$. Thus, $p - (n - x) = (x + p) - n = |n - (x + p)|$. Thus, we have $\text{rank } \phi = \min(n - m, m, |n - (x + p)|, x + p)$.

In all three cases, we have the same result, so this expression holds for all values of x . ■

Lemma 4 *The costs of performing the BMMC permutations characterized by the matrix products $SR_x S^{-1}$ and $SR_{n-x} S^{-1}$ are the same.*

Proof: From Lemma 2, the cost of performing the permutation characterized by SR_xS^{-1} is

$$\frac{2N}{BD} \left(\left\lceil \frac{\min(n-m, m, x, n-x)}{m-b} \right\rceil + 1 \right)$$

parallel I/O operations. The cost of performing the permutation characterized by $SR_{n-x}S^{-1}$ is

$$\frac{2N}{BD} \left(\left\lceil \frac{\min(n-m, m, n-x, n-(n-x))}{m-b} \right\rceil + 1 \right)$$

These two are obviously equal. ■

2.7 Determining the Optimal Ordering

Given our data, we want to determine what the optimal ordering of dimensions is to have the minimum I/O operations. For now, we will only read one dimension into memory at a time using the algorithm in Section 2.5. We just have to determine what the optimal ordering $X = \langle x_1, x_2, \dots, x_k \rangle$ of dimensions is.

Lemma 5 *The cost of performing the BMMC permutation characterized by the matrix SR_x differs by at most 1 pass through the data from the cost of performing the BMMC permutation SR_xS^{-1} .*

Proof: Recall that the BMMC permutation characterized by SR_x requires

$$\left\lceil \frac{\min(n-m, m, n-x+p, |x-p|)}{m-b} \right\rceil + 1$$

passes through the data, and the BMMC permutation characterized by SR_xS^{-1} requires

$$\left\lceil \frac{\min(n-m, m, n-x, x)}{m-b} \right\rceil + 1$$

passes. Observe that $\min(n-m, m, n-x+p, |x-p|)$ and $\min(n-m, m, n-x, x)$ differ by at most p . By our PDM assumptions, $m \geq d+b \geq p+b$, so $m-b \geq p$. Thus, the permutation SR_x requires at most

$$\left\lceil \frac{\min(n-m, m, n-x, x)}{m-b} + \frac{p}{m-b} \right\rceil + 1 \leq \left\lceil \frac{\min(n-m, m, n-x, x)}{m-b} \right\rceil + 2$$

passes through the data. Similarly for the minimum. ■

Lemma 6 *The I/O costs of the BMMC permutations characterized by R_xS^{-1} and SR_xS^{-1} differ by at most 1 pass through the data.*

Proof: Recall that the BMMC permutation characterized by the matrix product R_xS^{-1} requires

$$\left\lceil \frac{\min(n-m, m, |n-(x+p)|, x+p)}{m-b} \right\rceil + 1$$

passes through the data. Notice that $\min(n-m, m, |n-(x+p)|, x+p)$ and $\min(n-m, m, n-x, x)$ differ by at most p . Recall that $p \leq m-b$. Thus, the permutation R_xS^{-1} requires at most

$$\left\lceil \frac{\min(n-m, m, n-x, x)}{m-b} + \frac{p}{m-b} \right\rceil + 1 \leq \left\lceil \frac{\min(n-m, m, n-x, x)}{m-b} \right\rceil + 2$$

passes. Similar math applies for the minimum. ■

Theorem 7 *The ordering $\langle 1, 2, \dots, k \rangle$ takes at most 4 more passes through the data than an optimal ordering.*

Proof: Recall that the algorithm for computing the FFTs in the order $\langle x_1, x_2, \dots, x_k \rangle$ is

1. Before doing any FFT computations, we perform the BMMC permutation characterized by SR_{1,x_1} .
2. After performing the FFT computations for dimension x_j , where $1 \leq j < k$, perform the BMMC permutation characterized by the matrix product $SR_{x_j, x_{j+1}}S^{-1}$.
3. After performing the FFT computations for dimension x_k , perform the BMMC permutation characterized by the matrix product $R_{x_k,1}S^{-1}$.

By Lemma 5, the cost of (1) differs by at most 1 from the cost of performing the permutation characterized by the matrix product $SR_{1,x_1}S^{-1}$. By Lemma 6, the cost of (3) differs by at most 1 from the cost of performing the permutation characterized by the matrix product $SR_{x_k,1}S^{-1}$.

Let us look at the cost of performing the FFT computations in the order $\langle 1, 2, \dots, k \rangle$. Since we assumed that each $n_j \leq m - p$, Lemma 2 gives us that the cost of performing the BMMC permutation characterized by the matrix product $SR_{n_j, n_{j+1}}S^{-1}$, or $SR_{n_j}S^{-1}$, is

$$\left\lceil \frac{\min(n - m, n_j)}{m - b} \right\rceil + 1$$

passes through the data. Thus, performing the FFT computations in the order $\langle 1, 2, \dots, k \rangle$ takes at most

$$1 + \sum_{j=1}^k \left(\left\lceil \frac{\min(n - m, n_j)}{m - b} \right\rceil + 1 \right) + k + 2$$

passes through the data.

Let us assume that we have an optimal ordering $X = \{x_1, x_2, \dots, x_k\}$. To make the work a little bit clearer, let us define $Y = \{y_0, y_1, \dots, y_k\}$ to be the sequence of rotation amounts. That is to say,

$$y_i = \begin{cases} \min \left(\sum_{j=1}^{x_1} n_j, n - \sum_{j=1}^{x_1} n_j \right) & \text{if } i = 0, \\ \min \left(\sum_{j=\min(x_i, x_{i+1})}^{\max(x_i, x_{i+1})-1} n_j, n - \sum_{j=\min(x_i, x_{i+1})}^{\max(x_i, x_{i+1})-1} n_j \right) & \text{if } 1 \leq i < k, \\ \min \left(\sum_{j=1}^{x_k} n_j, n - \sum_{j=1}^{x_k} n_j \right) & \text{if } i = k. \end{cases}$$

From Lemmas 2 and 4, we know that performing the BMMC permutation characterized by the matrix $SR_{y_j}S^{-1}$ requires

$$\left\lceil \frac{\min(n - m, m, y_j)}{m - b} \right\rceil$$

passes through the data. Therefore, when we apply Lemmas 5 and 6, we get that the cost of performing our algorithm in the ordering X takes at least

$$\sum_{j=0}^k \left(\left\lceil \frac{\min(n - m, m, y_j)}{m - b} \right\rceil + 1 \right) + k - 2.$$

passes through the data. It is fairly obvious that

$$\sum_{j=0}^k \left(\left\lceil \frac{\min(n-m, m, y_j)}{m-b} \right\rceil + 1 \right) + k \geq 1 + \sum_{j=1}^k \left(\left\lceil \frac{\min(n-m, n_j)}{m-b} \right\rceil + 1 \right) + k ,$$

so, computing the one-dimensional FFTs in the order given is at most 4 passes (or $8N/BD$ parallel I/Os) slower than an optimal ordering, giving us a fairly good approximation. ■

2.8 Lower Bound on Dimensional Method

We already showed the FFT-ORDERING decision problem to be NP-complete. We are now interested in finding a good approximation of an optimal solution. We first look at a good lower bound on the I/O complexity of the Dimensional Method.

We rely heavily on the universal lower bound for BMMC permutations in [CSW99]. Any non-identity BMMC permutation requires

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

parallel I/Os, where γ is the lower left $(n-b) \times b$ submatrix of the characteristic matrix.

We are going to need some minimum cost based on a given grouping $G = \langle G_1, \dots, G_z \rangle$, where $G_i = \{g_{i_1}, \dots, g_{i_y}\}$ and each g_{i_j} is a unique dimension number. Before we can establish the minimum I/Os needed to perform the FFTs for all the dimensions, it makes sense to look at the cost to go from group G_i to group G_{i+1} .

The Dimensional Method requires that we perform the FFTs for the dimensions in G_i followed by some BMMC permutation so that the dimensions in G_{i+1} are *accessible*. That is, we want to be able to perform the FFTs for the dimensions in G_{i+1} with only one pass through the data.

Notice that when we perform these FFTs, we have some concept of *FFT groups*. That is to say, certain groups of records are processed together. For example, suppose we have 4 dimensions, each of size 2, and we have $G_1 = \{1, 2\}$, where $g \in G_i$ is a dimension index. We perform the FFTs in FFT groups $\{0, 1, 2, 3\}$, $\{4, 5, 6, 7\}$, $\{8, 9, 10, 11\}$, and $\{12, 13, 14, 15\}$, where each of these integers is a record index. Notice that the concept of FFT groups is independent of the available memory.

In finding a lower bound, let us first look at the structure on disk of a grouping G_i . That is, we want to show some minimum number of specific records in one block. Then, we can show a minimum rank γ to get from G_i to G_{i+1} . Once we have that, we can show a lower bound for performing the full Dimensional Method.

Lemma 8 *If*

$$C = \prod_{g_j \in G_i} N_{g_j} \geq M/B ,$$

and the data are stored so that G_i is accessible, then at least $C/(M/B)$ records in each block must belong to the same FFT group.

Proof: We can read only M/B blocks into memory at a time. These M/B blocks hold a total of M records, which must consist of M/C FFT groups. Suppose there is a block that contains fewer than $C/(M/B)$ records from any FFT group. Since we assume that we use only one pass through the data for FFT computations, we can only work with the M/C FFT groups, so the block holds fewer than $(M/C)(CB/M) = B$ records. But the block must contain exactly B records, so this produces a contradiction. Thus, every block must contain at least $C/(M/B)$ records from some FFT group. ■

Lemma 9 *Suppose we want to perform a BMBC permutation characterized by the matrix A to get G_i to be accessible, and we define C as in Lemma 8 ($C \geq M/B$), and all blocks have just 1 record from the FFT groups. Then $\text{rank } \gamma \geq \lg C - (m - b)$.*

Proof: From Lemma 8, following the BMBC permutation, each block must have at least $C/(M/B)$ records from an FFT group. Thus, each block must draw its records from at least $C/(M/B)$ pre-permutation blocks. It follows that each pre-permutation block must send its records to at least $C/(M/B)$ post-permutation blocks. Suppose this is not the case. That is, suppose some block σ sends its records to fewer than $C/(M/B)$ blocks when performing the BMBC permutation. Then that block must send more than $B/(CB/M) = M/C$ records to some block ω . We know that σ started with only 1 record from any FFT group. So now ω has more than M/C records that do not share any FFT group. So fewer than $B/(M/C)$ in ω share an FFT group, which violates Lemma 8. Thus, we now have that every block must send its records to at least $C/(M/B)$ blocks.

Let us now look at what implications this result has on γ . We perform the BMBC permutation characterized by the matrix

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

and some n -bit complement vector $c = (c_0, c_1, \dots, c_{n-1})$. The permutation maps a source address $x = (x_0, x_1, \dots, x_{n-1})$ onto a target address $y = (y_0, y_1, \dots, y_{n-1})$, where

$$\begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{bmatrix} = \left[\begin{array}{c|c} \alpha & \beta \\ \hline \gamma & \delta \end{array} \right] \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{bmatrix} \oplus \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{n-1} \end{bmatrix}.$$

Now, if we are not concerned with the offset within a block (only with which block a record is mapped to), we can look at the product

$$\begin{bmatrix} y_b \\ y_{b+1} \\ \vdots \\ y_{n-1} \end{bmatrix} = \left[\begin{array}{c|c} \gamma & \delta \end{array} \right] \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{bmatrix} \oplus \begin{bmatrix} c_b \\ c_{b+1} \\ \vdots \\ c_{n-1} \end{bmatrix}.$$

We know that given a source block, its records must map to at least $C/(M/B)$ target blocks. Notice that $y' = (y_b, y_{b+1}, \dots, y_{n-1})$ represents a target block. Since the most significant b bits of x are fixed, it follows that the range of

$$\left[\begin{array}{c} \gamma \end{array} \right] \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{b-1} \end{bmatrix}$$

consists of at least $C/(M/B)$ elements, so $\text{rank } \gamma \geq \lg(C/(M/B)) = \lg C - (m - b)$. ■

We want to show that an optimal solution to BIN-PACKING can give us an optimal solution to FFT-ORDERING. Thus, given an approximation to BIN-PACKING, we have a good approximation of the best way to perform the Dimensional Method. In the remainder of this section, we use bin packing to show a lower bound on the Dimensional Method. In Section 2.9, we show an asymptotically optimal Dimensional Method algorithm which uses an optimal solution to BIN-PACKING.

Theorem 10 Suppose $b \leq m/2$ (or $B^2 \leq M$). Suppose also that we make an instance to BIN-PACKING with $U = \{u_1, u_2, \dots, u_k\}$, $s(u) = n_u$, and a bin capacity of m . If z is the minimum number of bins into which U can be packed, then the Dimensional Method requires at least $\Omega(zN/BD)$ parallel I/Os.

Proof: This proof is fairly obvious. If z is the minimum number of bins, then it is also the minimum number of groups that we can arrange for the Dimensional Method. Since we need one pass through the data following each BMMC permutation to compute the FFTs, we need a minimum of $2zN/BD$ parallel I/Os. ■

Theorem 11 Suppose $b \geq m/2$ (or $B^2 \geq M$). Suppose also that we make an instance to BIN-PACKING with $U = \{u_1, u_2, \dots, u_k\}$, a bin capacity of $2(m - b)$, and

$$s(u) = \begin{cases} n_u & \text{if } n_u \leq 2(m - b) , \\ 2(m - b) & \text{if } n_u > 2(m - b) . \end{cases}$$

If U_1, U_2, \dots, U_z is the optimal partition, then the Dimensional Method requires at least

$$\Omega \left(\frac{N}{BD} \left(z + \sum_{i \text{ s.t. } n_i > 2(m-b)} \frac{n_i - (m-b)}{m-b} \right) \right)$$

parallel I/Os.

Proof: Let us first examine the grouping generated by an optimal BIN-PACKING solution. It must be the case that we have no more than one group G_i such that $\sum_{g \in G_i} n_g \leq m - b$. If there were 2 such groups, they could easily be combined because our bin capacity was $2(m - b)$. Notice also that the

$$\Omega \left(\frac{N}{BD} \left(\sum_{i \text{ s.t. } n_i > 2(m-b)} \frac{n_i - (m-b)}{m-b} \right) \right)$$

component of the cost comes from the large dimensions. From Lemma 9, if $\sum_{g \in G_{i+1}} n_g \geq m - b$, then we require

$$\Omega \left(\frac{N}{BD} \left(\frac{\sum_{g \in G_{i+1}} n_g - (m-b)}{m-b} \right) \right)$$

parallel I/Os to permute from G_i to G_{i+1} .

In order to reduce the number of groupings, we must increase the sizes of each grouping. Since only one group is smaller than M/B , reducing the number of groups by more than 1 results in a similar increase in cost from Lemma 9 (since $(m - b)/(m - b) = 1$). Thus, we have the minimum cost as stated above. ■

2.9 Approximation of Optimal Solution

Since the lower bound on I/Os for the Dimensional Method relied on BIN-PACKING, our approximation relies on a good approximation for BIN-PACKING.

First, we need a new BMMC permutation matrix. We want A_i to be the characteristic matrix of the permutation we need to get G_i to be accessible. Notice that the data are initially ordered so that bits $0, 1, \dots, n_0 - 1$ indicate dimension 0 (fixing all other bits, you get all the addresses for a vector in dimension 0). Similarly, $n_0, n_0 + 1, \dots, n_0 + n_1 - 1$ is the range of bits for dimension 1, etc. We want to maintain this simple bit ordering. So essentially, let A_i swap the bits representing

dimensions in G_i with the bits representing the dimensions that can fit in memory (i.e., the least significant m bits; we start the swapping with the most significant of these). For example, if we have $m = 5$, $n = 10$, $n_1 = 3$, $n_2 = 2$, $n_3 = 2$, $n_4 = 2$, and $n_5 = 1$, and we want A_1 where $G_1 = \{3, 5\}$, then we would have

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

Suppose $b < m/2$. Then we construct an instance to BIN-PACKING as in Theorem 10. Suppose U_1, U_2, \dots, U_z is the optimal partition. Then we want groups $G_i = \{g_j : u_j \in U_i\}$.

Suppose $b > m/2$. Then we construct an instance to BIN-PACKING as in Theorem 11.

We can now use the following algorithm:

1. Perform the BMMC permutation characterized by the matrix SA_1 .
2. Between computing the FFTs for dimensions in group G_i and G_{i+1} , perform the BMMC permutation characterized by the matrix $SA_{i+1}S^{-1}$.
3. After computing the G_z FFTs, perform the BMMC permutation characterized by the matrix A_0S^{-1} .

We want to show that this algorithm performs as well as the minimum shown in Theorems 10 and 11.

Theorem 12 *Suppose $b < m/2$. This Dimensional Method algorithm takes at most $O(zN/BD)$ parallel I/Os.*

Proof: This proof is fairly obvious. If we have z groups, we need z passes through the data to perform the FFTs. Since $b < m/2$, we can perform any BMMC permutation with 3 passes through the data. Since we must perform only $z + 1$ such permutations, our cost is no more than $\frac{2N}{BD}4(z + 1)$ parallel I/Os, or $O(zN/BD)$ parallel I/Os. ■

Theorem 13 *Suppose $b > m/2$. This Dimensional Method algorithm takes at most*

$$O\left(\frac{N}{BD}\left(z + \sum_{i \text{ s.t. } n_i > 2(m-b)} \frac{n_i - (m-b)}{m-b}\right)\right)$$

parallel I/Os.

Proof: For now, let us assume that we are using a single processor. Then this proof is relatively simple. The rank of the lower left $(n-m) \times n$ submatrix of A_i is $\sum_{g \in G_i} n_g$. So the cost of performing the BMMC permutation characterized by A_i is at most

$$\frac{2N}{BD}\left(\left\lceil \frac{\sum_{g \in G_i} n_g}{m-b} \right\rceil + 1\right)$$

I/O operations. For the large groups (a group with a dimension bigger than $2(m - b)$), the cost becomes

$$\frac{2N}{BD} \left(\frac{n_i - 2(m - b)}{m - b} + 4 \right) .$$

For the other groups (the size is less than $2(m - b)$), the cost becomes

$$\frac{2N}{BD} (3) .$$

If we add the cost of performing the z groups' FFT computations, we get a total cost of less than

$$\frac{2N}{BD} \left(5z + \frac{\sum_{i \text{ s.t. } n_i > 2(m - b)} n_i - (m - b)}{m - b} \right)$$

parallel I/Os, so the complexity is

$$O \left(\frac{2N}{BD} \left(z + \frac{\sum_{i \text{ s.t. } n_i > 2(m - b)} n_i - (m - b)}{m - b} \right) \right) .$$

■

2.10 Some Notes on Optimization

We can trim off some I/O operations. This optimization does not affect the I/O complexity significantly, but it may reduce some constants involved. When we permute the data above in the Dimensional Method, we get $m - (b + d)$ bits for free. The Dimensional Method presented in this paper rearranges the data so that consecutive stripes are read in to perform the FFTs, but this reordering may waste I/O operations. We can read in any $m - (b + d)$ stripes we want at a time. Thus, any movement within bits $b + d, b + d + 1, \dots, m - 1$ is wasted. Since the overriding factor is the number of groups, this optimization only affects the constant.

Bibliography

- [Bap99] Lauren M. Baptist. Two Algorithms for Performing Multidimensional, Multiprocessor, Out-of-Core FFTs. Dartmouth College Computer Science Technical Report PCS-TR99-350.
- [CLR90] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. The MIT Press, Cambridge, Massachusetts, 1990.
- [CN98] Thomas H. Cormen and David M. Nicol. Performing out-of-core FFTs on parallel disk systems. *Parallel Computing*, 24(1):5-20, January 1998.
- [Cor97] Thomas H. Cormen. Determining an Out-ofCore FFT Decomposition Strategy for Parallel Disks by Dynamic Programming. Dartmouth College Computer Science Technical Report PCS-TR97-322, July 1997.
- [CSW99] Thomas H. Cormen, Thomas Sundquist, and Leonard F. Wisniewski. Asymptotically tight bounds for performing BMMC permutations on parallel disk systems. *SIAM Journal on Computing*, 28(1):105-136, 1999.
- [CWN97] Thomas H. Cormen, Jake Wegmann, and David M. Nicol. Multiprocessor Out-ofCore FFTs with Distributed Memory and Parallel Disks. Dartmouth College Computer Science Technical Report PCS-TR97-303, January 1997.
- [GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the theory of NP-Completeness*. W. H. Freeman And Company, New York, 1979.
- [VS94] Jeffrey Scott Vitter and Elizabeth A. M. Shriver. Algorithms for Parallel Memory, I: Two-Level Memories. *Algorithmica*, 12(2/3):110-147, August/September 1994.