A Mathematical Model for XOR-Based Application Specific DRAM Address Mapping Schemes

Andrei Rotaru
rotaru@rptu.de
Microelectronic Systems Design Research Group, RPTU Kaiserslautern-Landau
Kaiserslautern, Germany

Oliver Bachtler
o.bachtler@math.rptu.de
Optimization Research Group, RPTU Kaiserslautern-Landau
Kaiserslautern, Germany

Lukas Steiner
lukas.steiner@rptu.de
Microelectronic Systems Design Research Group, RPTU Kaiserslautern-Landau
Kaiserslautern, Germany

Matthias Jung
m.jung@uni-wuerzburg.de
JMU Würzburg / Fraunhofer IESE
Würzburg / Kaiserslautern, Germany

Sven O. Krumke
sven.krumke@math.rptu.de
Optimization Research Group, RPTU Kaiserslautern-Landau
Kaiserslautern, Germany

Norbert Wehn
norbert.wehn@rptu.de
Microelectronic Systems Design Research Group, RPTU Kaiserslautern-Landau
Kaiserslautern, Germany

ABSTRACT

The increasing performance gap between the bandwidth requirements of modern Systems on Chip (SoCs) and the I/O data rate delivered by Dynamic Random Access Memory (DRAM) limits the performance of today's data-intensive applications. A promising direction to mitigate this bottleneck lies in application-specific DRAM address mappings, which exploit the knowledge of an application's memory access behavior to increase the throughput, reduce energy consumption and lower access latency. In this paper, we propose a framework that generalizes the state of the art, by synthesizing address mappings based on Binary Integer Matrices (BIMs) over $GF(2)$ (i.e. the finite field with two elements), constructed from simple logic operations (XOR and AND) in order to address two key challenges simultaneously: minimizing row misses and constraining hardware complexity. We show that the size of the design space can be significantly reduced by reformulating the row-miss minimization problem for XOR-based mappings as a subspace selection problem over $GF(2)^n$ that is determined by the bit flip information of the whole address sequence considered. Furthermore, our approach extends the scope of mapping design from fixed access sequences to probabilistic memory access models, and highlights both the strengths and limitations of the BIM-based approach with respect to row miss minimization.

CCS CONCEPTS

• **Hardware → Dynamic memory; Software tools for EDA**.

KEYWORDS

Application Specific Memory Controller, DRAM, Address Mapping, Optimization, Subspace Selection Problem, Combinatorics, Embedded Systems

1 INTRODUCTION

In the last years, the computational power of modern processors, such as CPUs, GPUs, and specialized accelerators, has advanced rapidly. On the other side, the progression of the memory subsystem performance has been comparatively modest. More precisely, the achievable memory bandwidth and access latency have not kept pace with the growing demands of data-intensive workloads. This problem is referred to as the Memory Wall [25], a well-known performance bottleneck that leads to frequently underutilized processors due to stalled memory access. Many applications, particularly in embedded systems, exhibit deterministic memory-access patterns, either temporal (commands follow a fixed rule) or spatial (the address sequence is known in advance). Despite this regularity, relatively little work has exploited application knowledge to design customized memory controllers. Harnessing these deterministic patterns in application-specific DRAM controllers is a promising direction for mitigating the memory wall and unlocking the full performance potential of contemporary processors [12, 18].

In general, application-level knowledge can be leveraged in two principal ways: (1) through online scheduling techniques, or (2) via offline methods such as optimizing the address mapping of the DRAM controller. Prior research (see Section 3) has demonstrated that the latter approach is lightweight and can significantly enhance performance and energy efficiency, as it enables a global view of the application's memory access behavior. In contrast, online scheduling techniques operate with limited visibility, typically constrained to a short temporal window, thereby restricting their ability to fully exploit optimization opportunities.

In this context, these works have identified two metrics that are strongly related to the performance of the memory subsystem when designing the address mapping: the number of row misses (or page conflicts) and the degree of bank-level parallelism exploited.

This work is based on the publications [12, 18], in which the authors introduced mathematical optimization problems to determine address mappings for given fixed address traces that are provably optimal with respect to row miss minimization. While [12] focused on general bijective address mappings, which were found to be hardware-inefficient, the authors in [18] concentrated on hardware-efficient bit-permutation-based address mappings (see also Section 3 for more details). In this work, we extend the row-miss minimizing address mapping problem introduced in [18] by exploring a broader class of address mappings based on Binary Integer Matrices (BIMs).

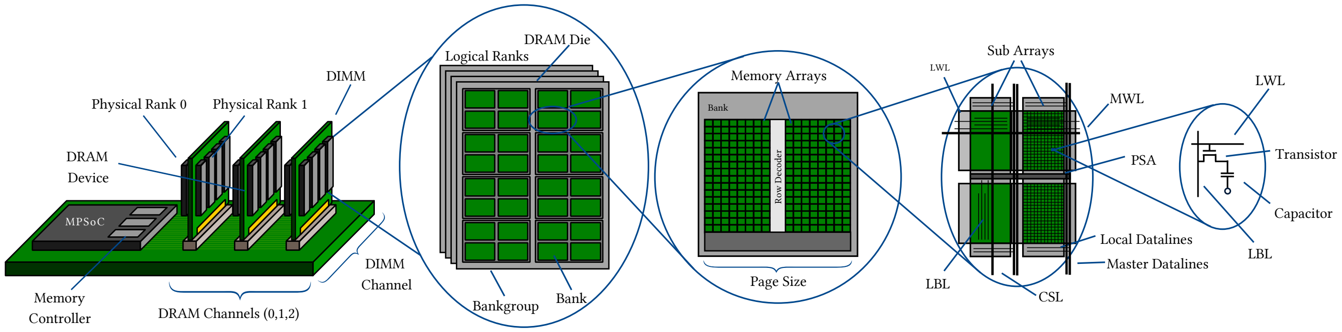


Figure 1: DRAM Device Architecture

While numerous publications have investigated BIM-based address mappings tailored to specific applications, typically based on deterministic access traces, few have actually targeted general binary integer matrices (BIMs) in a rigorous or mathematically grounded manner. The influential work of [2] first highlighted that various address mapping schemes can be represented using full-rank square BIMs over the Galois Field $GF(2)$. However, most subsequent approaches [13, 14, 24] either employed heuristics or focused solely on mappings constrained to bit permutations, which is a strict subclass of general BIMs.

For this restricted case, the authors of [18] presented a method for computing mathematically optimal bit-permutation-based address mappings that minimize row misses for fixed deterministic traces in embedded applications. This currently constitutes the state of the art in exact hardware-efficient offline address mapping optimization.

In contrast, the general case of full-rank BIM-based address mappings has so far remained largely unexplored from a mathematical optimization perspective. In this work, we aim to bridge this gap by extending the state of the art established for bit-permutation-based address mappings [18] by investigating mathematically optimal BIM-based address mappings over $GF(2)$ with respect to row-miss minimization. More specifically, we formulate the problem of finding a row-miss minimizing full-rank BIM as a mathematical optimization problem, thereby extending the models developed in [18], especially since mappings based on bit permutations can be realized as BIMs. Moreover, by leveraging techniques from linear algebra, we show that the problem of finding a row-miss minimizing full-rank BIM over $GF(2)$ can be reduced to an NP-hard subspace selection problem in a vector space over $GF(2)$. This reduction significantly decreases the size of the design space and offers new insights into the structure of the problem. Our approach allows for the first time to rigorously study the tradeoff between implementation complexity and efficiency in terms of row miss minimization. In particular we provide:

- Lower and upper bounds on the row hits achievable by a full-rank BIM with respect to a given trace.
- The possibility of modeling random sequences of physical address vectors that follow a given probability distribution and methods to find address mappings that minimize the number of row misses with respect to a given access trace in expectation.
- Conditions on bank bit mappings in order to achieve good results with respect to minimizing the number of row misses incurred by the mappings considered.

This paper additionally, besides the mathematical model and corresponding insights for finding full-rank row-miss minimizing BIMs, makes the following new contributions:

- We provide an integer linear program formulation that enables the employment of LP-based branch and bound algorithms to solve the subspace selection problem and therefore the row-miss minimization problems for BIMs exactly.
- We show that the bit permutation-based approach is hopelessly bad for certain access patterns and develop a greedy heuristic to find robust solutions to various access patterns.
- We also address the cost of implementing the row-miss minimizing address mapping on chip: the sparsity of a candidate matrix (number of non-zero entries) is strongly correlated with its hardware footprint in fixed-function implementations, and with power consumption in more flexible, programmable realizations of the BIM. More specifically, we provide an algorithm to find a full-rank BIM with fewer non-zero entries that incurs the same number of row misses as a given row-miss minimizing BIM.

The advantage of our approach is twofold. First, the hardware implementation remains small even in the worst case while offering flexibility based on the sparsity of the BIM and robustness with respect to complex access patterns. Second, the linearity of the address mapping allows us to leverage the vector space structure of $GF(2)^n$, enabling the use of classical linear algebra techniques to tackle the problem.

The paper is organized as follows: Section 2 explains the basic components and structure of a DRAM and Section 3 discusses related work. Our model for the row miss minimization problem for BIMs is presented in Section 4, and the reduction to a subspace selection problem is the topic of Section 5. In Section 6, we treat the single bank case, which we extend to the case of a fixed bank bit mapping in Section 7. In Section 8, we present experimental results and conclude in Section 9.

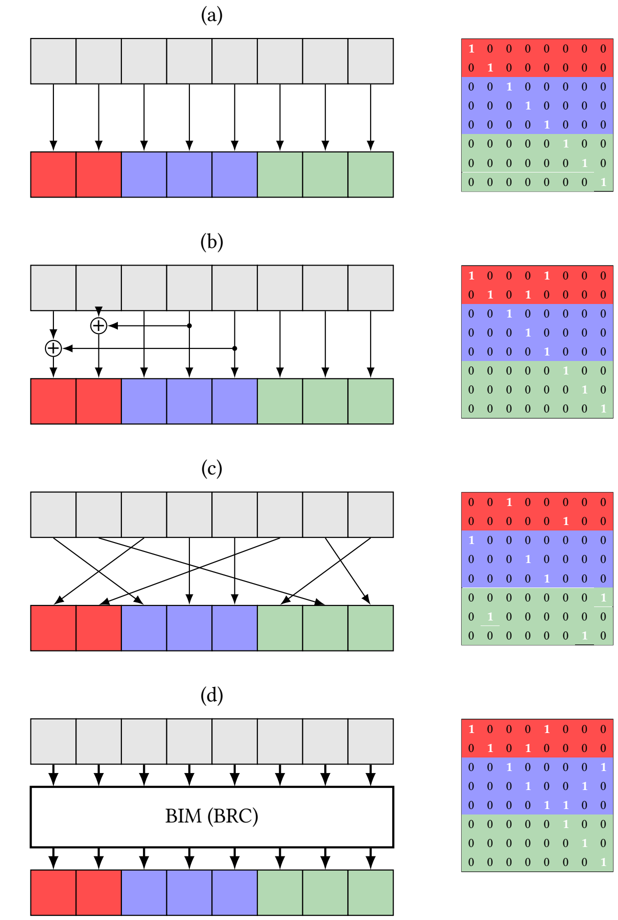


Figure 2: Different address mapping variants and their corresponding BIM realizations (Grey Blocks - Original Address, Red - Bank Bits, Blue - Row Bits, Green - Column Bits): (a) linear address mapping (BRC), (b) Permutation - Based Page Interleaving, (c) Bit Permutation Based Address Mapping and (d) General BIM - Based Address Mapping.

2 DRAM BACKGROUND

DRAM is a memory technology optimized for low cost per bit, meaning memory density must be maximized. To achieve this, a DRAM device internally uses a special hierarchical structure, as shown in Figure 1. Individual memory cells, which consist of a tiny capacitor and an access transistor, are densely packed and organized in so-called columns, rows and banks. Before a column of data can be accessed, the target row within the target bank must be activated first. This connects the memory cells of the row to sense amplifiers, which can be read out or overwritten afterwards. To reduce the routing overhead within the device, in each bank, only one row can be activated at a time. Before accessing the data of another row, the activated row must first be precharged, disconnecting the sense amplifiers from the capacitors and preparing them for the next sensing (activation). As the precharging and activation processes are associated with relatively long latencies and no data can be accessed in the bank during this time, they negatively affect both the bandwidth utilization and the response latency of the memory. To minimize the number of activations and precharges, consecutive accesses to one bank should therefore target columns of the same row (so-called row hits).

Optimizing the row-hit ratio is an important task for the DRAM controller. This can be achieved in two different ways. Firstly, the controller can queue incoming requests and reorder them on the fly using its scheduler. While this does not require prior knowledge of the memory access pattern, it does add additional latency and hardware complexity to the controller. Secondly, if the memory access pattern of an application or a hardware unit is deterministic and known in advance, the row-hit ratio can also be optimized by determining an application-specific mapping from physical (system) addresses to DRAM addresses. For example, if the access pattern is sequential, the lower bits of the physical address should be mapped to bank and column bits because they have a higher toggling rate, while the upper bits should be mapped to the row bits because they have a lower toggling rate.

3 RELATED WORK

There are several publications considering different approaches for optimizing the aforementioned objectives, which are covered in this section.

3.1 Online Scheduling

General-purpose DRAM controllers employ dynamic scheduling strategies to minimize row misses. Typically, these controllers operate in two primary modes: *Open Page Policy* (OPP) and *Closed Page Policy* (CPP). In OPP, the currently accessed row remains open following a `RD` or `WR` command, while CPP automatically precharges the row (Auto-Precharge: `RDA`/`WRA`). CPP is commonly applied in server environments (e.g., web servers), where memory accesses tend to be uniformly random. In contrast, OPP is preferred in desktop and mobile platforms where higher spatial and temporal locality leads to more frequent row hits. To further enhance row buffer hit rates under OPP, dynamic scheduling techniques are utilized. The widely adopted online DRAM scheduling policy known as *First Ready First Come First Served* (FR-FCFS) was introduced by Rixner et al. [20]. FR-FCFS organizes memory requests in a queue to prioritize those targeting the same row, forming groups of row hits (*row-hit-first* strategy). When no row hits are available, the oldest pending request is issued (*oldest-first* strategy). This approach has been refined for general-purpose CPUs in [9] and adapted for multicore and heterogeneous architectures in [16] and [3]. To limit mode-switch overhead between `RD` and `WR`, DRAM controllers maintain separate queues for each and employ an arbiter to alternate between them efficiently. These techniques are implemented in commercial off-the-shelf DRAM controllers.

3.2 Address Mapping

Another approach to improve the performance of the memory subsystem for specific applications is to design an optimized physical DRAM address mapping, which minimizes the number of row misses for a given trace and maximizes the degree of exploited bank-level parallelism. An application-specific address mapping can assist or even outperform an online scheduler in specific cases [18]. Common address mappings take an initial physical address and map it to a new address of the same bit length. Hereby, the bits are partitioned into consecutive blocks of bits (here referred to as bank, row and column bits) that binary encode the bank, row and column index in the physical DRAM (see e.g. Figure 2). A straight forward addresss mapping is the linear address mapping, illustrated in Figure 2 (a), and is rarely used in practice. The most common addressing schemes used, make use of the fact that many accesses to memory are structured as they are e.g. (one-)strided, leading to high toggle rates in the least significant address bits, which are then used to determine the column, the bits with the next highest toggle rates are used for determining the banks and the bits with the lowest toggle rates are used for the rows (see e.g. [1]). Low toggle rates in the row bits increase the likelihood of row hits. An address mapping scheme that does this is the so-called Page Mode or Bank Interleaving [10], and the primary focus of it is to ensure a better bank utilization. Sophisticated memory controller used in high end CPU support the Permutation-Based Page Interleaving technique [26], illustrated in Figure 2 (b). In this technique, accesses to different rows in the same bank are transformed into accesses to different banks by XORing the bank bits with selected row bits, thus making use of bank-level parallelism. The authors in [2] and [14] realized that many of the common address mapping techniques could be generalized by binary integer matrices over $GF(2)$. However, as many other subsequent works, they focused on address mappings based on address bit permutations, as illustrated in Figure 2 (c). To this end, the authors developed heuristic methods based on the toggle rate of individual address bits or window-based entropy measures, that concentrated bit toggle rate information into single values. The primary focus of the authors was on designing permutation matrices in the online mode and changing the address mapping on-the-fly for newly incoming memory requests. This approach involved a large overhead as it required data migration. The authors in [14] considered bit permutation-based (PAE) and general BIM-based (FAE and All) address mappings for GPUs, as illustrated in Figure 2 (d), suitable for handling concurrent accesses to DRAM. They focused on better bank utilization and used window-based entropy measures to determine the entropy information of bits which was used to remap address bits in such a way, that the entropy of the resulting bank and channel bits in the remapped address is high. They reasoned about the choice of the BIM rows heuristically and traversed the design space of all BIMS by randomly generating BIMS and choosing the best performing one in their All scheme. Furthermore, they only achieved slight performance improvements compared to the permutation-based approach (PAE). We will show that, especially for random patterns and patterns stemming from concurrent accesses to DRAM, the general BIM approach can be considerably better than the bit-permutation based approach. The authors in [1] investigated address mappings for fixed address traces and developed a heuristic approach based on entropy and bit-flip based metrics to heuristically determine well-performing bit-permutation-based mappings for specific applications targeting row misses and bank parallelism. The authors in [13] also recognized that most of the common address mappings can be generalized by binary integer matrices and suggested that the entropy- and bit-flip ratio based measures for determining permutation-based mappings were not sufficient and too simplistic. They proposed an approach based on policy gradient-based reinforcement learning for determining efficient address mapping for deterministic sequences of addresses as well as for the online case. However, their approach is limited to permutations, as the action space of their reinforcement learning approach for considering general BIMs would otherwise be too large and infeasible for realisitc scenarios. Furthermore, in the offline case, i.e. for a fixed memory trace, they didn't compare the bit-permutation-based address mapping computed by their algorithm to the one computed by the approach described in [18], which, for the first time, yielded an exact mathematically optimal row-miss minimizing permutation-based address mapping. It has been shown in several works that the row misses incurred by an address mapping are strongly correlated with the overall performance of the memory system.

In [12], the authors considered general bijective address mappings. That means, functions which map adresses one-to-one to physical DRAM locations. Furthermore, they showed that minimizing row misses in this context leads to graph-cut problems, which they addressed using integer linear programming and heuristic methods. However, this approach has a significant drawback, which is its reliance on lookup tables to implement the determined mappings. These tables could grow arbitrarily large, potentially consuming an excessive chip area and rendering the solution impractical for real-world deployment.

To overcome this, the authors in [18] investigated a more hardware-efficient subclass of bijective mappings, namely permutations on bits, which offered a better trade-off between solution quality and implementation cost. The authors showed that an optimal row-miss minimizing function in the restricted address mapping space could be determined by solving an instance of the minimum k-union problem on hypergraphs. By employing a branch-and-bound algorithm, they managed, for the first time, to determine an exact solution that admits a compact hardware implementation, requiring only a constant number of multiplexers.

Up to now, the method developed in [18] is the state-of-the-art method for determining realistically employable address mapping that are mathematically optimal within their class of mappings. Note, however that the reduction of the minimum row miss problem to Min-k-Union, derived in [18], used the exact positions of addresses within the sequence, which made it unclear how to extend this approach to non-deterministic, random sequences that obey a certain distribution. Furthermore, we will show that bit-permutation based mappings are not suitable for applications that generate traces with high bit-wise toggling rates, such as applications that involve concurrent or random accesses to memory.

The authors of [11] implemented the Min-k-Union problem on a quantum computing platform and demonstrated that, assuming future quantum computers provide access to a sufficiently large number of qubits, the address mapping problem—under fixed hardware constraints—can be solved efficiently.

3.3 Matrices over $GF(2)$

The authors in [22] developed a general model for the design of XOR-based hash functions with applications to the elimination of cache-miss conflicts. They observed that cache conflicts could be eliminated by choosing a hash function with an appropriate kernel. We leverage a similar approach and show that row misses incurred by BIM-based address mappings can be characterized by the kernel space of a submatrix, or more specifically the so-called row bit mapping. Similar approaches, characterizing $GF(2)$ matrices by their kernel, have long been utilized in the theory of error correcting codes. For linear codes, the row vectors of any generator matrix form a basis of the kernel (null space) of the corresponding parity-check matrix. The authors in [21] used the theory from [22] to heuristically develop hash functions for banked scratch pad Memories in GPUs in order to reduce bank conflicts. Note, however, that the nature of the row-miss minimization problem studied in this work differs fundamentally from that of minimizing cache or bank conflict misses: in the cache and bank conflict case, the goal is to design a mapping from physical addresses to banks or cache sets that is ideally surjective and spreads addresses evenly to avoid multiple addresses mapping to the same bank or cache set at the same time. In contrast, row-miss minimization seeks a function that, in essence, does the opposite, that is, it ideally maps multiple addresses to the same row to avoid row misses.

4 MATHEMATICAL METHODS FOR ROW MISS MINIMIZATION

In this section, we explain the core of our work, namely our mathematical model for computing optimal address mappings based on binary integer matrices. Our goal in this context is to minimize both the row misses and the size of the hardware required to realize the address mapping. In order to simplify the notation, we will restrict our addressing to banks, rows and columns.

Let us begin by describing our model. We assume that our DRAM consists of 2^b banks, where each bank has 2^r rows that each contain 2^c columns, for some $b, r, c \in \mathbb{Z}_{\geq 0}$ and $n \coloneqq b + r + c$. In particular, we address the DRAM by binary vectors y in $GF(2)^n$, where the first b bits y^B of y address the bank, the next r bits y^R specify the row, and the final c bits y^C address the column.

Let $\mathcal{X} \coloneqq (x_0, \ldots, x_k) \subseteq \{0,1\}^n$ be a sequence of n-bit physical addresses that should be mapped to the bank, row, and column where the corresponding data is allocated in the DRAM. We do not allow arbitrary mappings, as these are typically expensive to implement [18], and restrict ourselves to bijective linear mappings, that is, we want to map the physical addresses x_i by means of a square matrix $A \in GF(2)^{n\times n}$ of full rank to the binary DRAM address vectors

$$y_i \coloneqq A \cdot x_i \text{ for } i \in \{0, \ldots, k\}\,. \tag{1}$$

We observe that we can split the matrix A into submatrices A_B, A_R, A_C and write (1) equivalently as

$$\begin{pmatrix} y_i^B \\ y_i^R \\ y_i^C \end{pmatrix} = \begin{bmatrix} A_B \\ A_R \\ A_C \end{bmatrix} \cdot x_i = \begin{pmatrix} A_B \cdot x_i \\ A_R \cdot x_i \\ A_C \cdot x_i \end{pmatrix} \text{ for } i \in \{0, \ldots, k\} \tag{2}$$

Here, A_B, A_R and A_C are matrices $A_B \in GF(2)^{b\times n}$, $A_R \in GF(2)^{r\times n}$ and $A_C \in GF(2)^{c\times n}$ and $y_i^B = A_B \cdot x_i$, $y_i^R = A_R \cdot x_i$, and $y_i^C = A_C \cdot x_i$ specify the bank, row, and column that x_i is mapped to. Furthermore, since A has full rank, it follows that these submatrices have full row rank.

The request to a memory bank induces a row miss if and only if no row in the corresponding bank has been activated so far (ACT), or if the last activated row in this bank differs from the one currently requested.

Definition 1. Let $A \in GF(2)^{n\times n}$ be a binary matrix of full rank and $\mathcal{X} \coloneqq (x_0, \ldots, x_k) \subseteq GF(2)^n$ a sequence of $k+1$ logical addresses. We define the *bank sequence* $\mathcal{X}_j$ to be the subsequence of $\mathcal{X}$ containing the addresses mapped to the jth bank, for $j = 0, \ldots, 2^b - 1$. For $x_i \in \mathcal{X}_j$, $A_B \cdot x_i$ encodes the jth bank and we say $\mathcal{X}_j$ is x_i's bank sequence.

Then A induces a *row miss* at $i \in \{0, \ldots, k\}$ for $\mathcal{X}$ if and only if

(1) x_i is the first element of its bank sequence or
(2) the predecessor x_p of x_i in its bank sequence is mapped to a different row, that is, $A_R \cdot x_i \neq A_R \cdot x_p$.

To avoid case distinctions, we note that A induces a *row hit* at $i \in \{0, \ldots, k\}$ for $\mathcal{X}$ if and only if it does not incur a row miss. This is the case if and only if the predecessor x_p of x_i in its bank sequence exists and is mapped to the same row, that is, $A_R \cdot x_i = A_R \cdot x_p$.

We denote the set of indices for which A induces row hits by $H_{\mathcal{X}}(A)$ or just $H(A)$, that is,

$$H(A) \coloneqq H_{\mathcal{X}}(A) \coloneqq \{i \in \{0, \ldots, k\} :\ A \text{ induces a row hit for } \mathcal{X} \text{ at } i\}\,.$$

Let us take a look at an example. In many cases, it will be convenient to denote our memory accesses by integers instead of binary vectors, which is why we write $\overrightarrow{z}$ for the binary vector of length n corresponding to the number z.

Example 1. Let $n = 3$, $\mathcal{X} = (\overrightarrow{0}, \overrightarrow{7}, \overrightarrow{2}, \overrightarrow{7}, \overrightarrow{2}, \overrightarrow{5}, \overrightarrow{5}, \overrightarrow{6}, \overrightarrow{4}, \overrightarrow{3}, \overrightarrow{5})$, $b = 0$, $r = 2$, and $c = 1$. We want to allocate the addresses in $\mathcal{X}$ to the DRAM, that is, to one of the four rows and two columns. Consider the matrix

$$A = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \text{ where } A_R = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \text{ and } A_C = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}.$$

Furthermore, let us consider the addresses 0, 7, 2, and 5 whose corresponding address vectors are

$$\overrightarrow{0} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \overrightarrow{7} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \overrightarrow{2} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \text{and } \overrightarrow{5} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}.$$

By computing

$$A_R \cdot \overrightarrow{0} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, A_R \cdot \overrightarrow{7} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, A_R \cdot \overrightarrow{2} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \text{and } A_R \cdot \overrightarrow{5} = \begin{pmatrix} 0 \\ 1 \end{pmatrix},$$

we see that $\overrightarrow{0}$ and $\overrightarrow{7}$ are mapped to the same row, as are $\overrightarrow{2}$ and $\overrightarrow{5}$, giving us row hits at positions 1, 5, and 6, but misses at 0, 2, 3, and 4.

As a way to illustrate our model so far and to relate it to [12], we want to briefly revisit the notion of an access graph for the sequence $\mathcal{X}$, see [12], which is simply an undirected graph $G_{\mathcal{X}}$ with

$$V(G_{\mathcal{X}}) \coloneqq \{x_i :\ i \in \{0, \ldots, k\}\} \text{ and}$$
$$E(G_{\mathcal{X}}) \coloneqq \{e_i \coloneqq (x_{i-1}, x_i) :\ i \in \{1, \ldots, k\}\}\,.$$

For the sequence $\mathcal{X}$ in Example 1, the access graph $G_{\mathcal{X}}$ can be seen in Figure 3. The graph simply visualizes the accesses between subsequent addresses of the sequence.

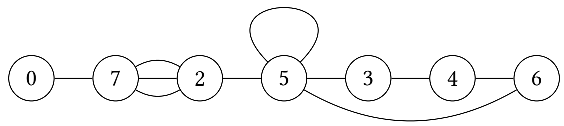


Figure 3: Access graph for the sequence $\mathcal{X}$ from Example 1

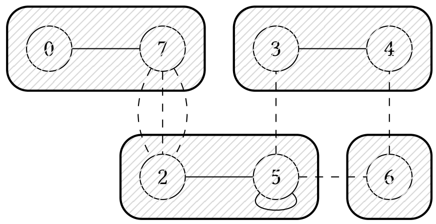


Figure 4: The partition of the access graph of the sequence $\mathcal{X}$ and mapping A from Example 1. Dashed edges between parts correspond to row misses.

In [12], the row miss minimization problem for general bijective address mappings for a single bank led to graph-cut problems on the access graph induced by the given sequence. More specifically, since each vertex corresponds to exactly one unique address from the sequence, every partition of the graph into at most 2^r components that each contains at most 2^c nodes can be translated in to a feasible bijective mapping. This general unconstrained case had many degrees of freedom (placing individual nodes into the parts of the partition), leading to an Integer Linear Programming (ILP) formulation with a huge number of variables and the resulting address mappings were typically complex to implement. To remedy this, the authors in [18] restricted the class of feasible mappings to permutations of bits and obtained simpler (and thus lower quality) mappings, but ones that had efficient hardware implementations.

Our model now lies between these two approaches: Linear address mappings might not cover all bijective maps but promise to yield ones that have hardware implementations of reasonable size. On the other hand, they are more general than the permutation-based approach since permutations can be represented by permutation matrices in our model.

Returning to Example 1, the matrix A leads to the partition of the access graph shown in Figure 4, where edges between vertices of a part of the partition correspond to row hits and edges between parts to row misses.

As we just noted, the main reason we want to restrict access mappings is that general ones are too complex to implement. But even amongst all linear mappings, not all are equally simple. We note that the number of non-zero entries of the matrix A, denoted by the ℓ_0-“norm” $\|A\|_0$ of A, corresponds to the size of the hardware realizing A on the final chip for a fixed implementation and is a measure for the power consumption of A in a flexible implementation. Thus, we also want to minimize $\|A\|_0$ if possible.

We are now ready to formulate the problem we are trying to solve, which searches for a binary matrix A with full rank that maximizes the number of row hits $H(A)$ and also minimizes the number of non-zero entries $\|A\|_0$ of A. We write the problem with two objectives, where we typically prioritize the first.

Problem 1.

$$\begin{array}{ll} \max & |H(A)| \\ \min & \|A\|_0 \\ \text{s.t.} & \operatorname{rank}(A) = n \\ & A \in GF(2)^{n\times n} \end{array}$$

The optimal solution to Problem 1 has the potential to achieve better solutions than those obtained through bit-permutations, which Examples 2 and 5 illustrate. In Example 2, we see that mappings based on permutations have less than 2 % row hits, while our more general model has a solution that yields a row hit for half the accesses and in Example 5 we see that the permutation matrices have the worst performance amongst all linear maps. But, in order to present these examples properly, it helps to analyze the model a bit more.

So let us start by making a few observations. The column-bit-matrix A_C has no effect on the number of row hits, that is, it does not affect the first objective. Thus, for fixed bank- and row-bit-matrices A_B and A_R, we can find a column-bit-matrix A_C such that the matrix $A = \begin{bmatrix} A_B \\ A_R \\ A_C \end{bmatrix}$ has full rank and A_C contains only c many one entries (by iteratively adding unit vectors not contained in the span of the rows of A_B and A_R). This number is optimal with respect to the second objective, since the full row rank condition necessitates at least one non-zero entry per row.

Observation 1. For a given bank- and row-bit-matrix A_B and A_R, we can find an optimal column-bit-matrix A_C efficiently.

Next, we take a closer look at the number of row hits. These depend on both A_B and A_R.

- The matrix A_B determines the subsequences $\mathcal{X}_j$ and these affect the row hits. Two addresses x_i and x_p are mapped to the same bank (and end up in the same subsequence) if $A_B \cdot x_p = A_B \cdot x_i$, or equivalently, if $A_B \cdot (x_i - x_p) = 0$.
- Similarly, we get a row hit each time to subsequent addresses x_p and x_i in a sequence $\mathcal{X}_j$ are mapped to the same row, that is, if $A_R \cdot x_p = A_R \cdot x_i$, or equivalently, if $A_R \cdot (x_i - x_p) = 0$.

We note that, since we are working over $GF(2)$, these difference vectors $x_i - x_p$ can also be written as $x_i + x_p$ and we shall use the notation $x_i \oplus x_p$ to highlight the fact that this is the addition over $GF(2)$ and not the addition of two binary vectors.

All in all, we have seen that, the predecessor x_p of x_i in the sequence $\mathcal{X}_j$ depends solely on the bank-bit-matrix A_B, while the row hits only depend on the vectors $x_i \oplus x_p$, and we want to choose A_R such that the number of these vectors in the kernel of A_R, that is, in $\operatorname{Ker}(A_R) = \{d \in GF(2)^n\colon\ A_R \cdot d = 0\}$, is maximal. This motivates the following terminology.

Definition 2. For $j \in \{0, \ldots, 2^b - 1\}$, let $\mathcal{X}_j = (x_{i_1}, \ldots, x_{i_l})$. We define $\mathcal{D}_j = (x_{i_2} \oplus x_{i_1}, \ldots, x_{i_l} \oplus x_{i_{l-1}})$ to be the sequence of difference vectors of $\mathcal{X}_j$. Moreover, we let $\mathcal{D}$ be the concatenation of $\mathcal{D}_0, \ldots, \mathcal{D}_{2^b-1}$.

Observation 2. The number of row hits $H(A)$ is equal to the number of elements in $\mathcal{D} = (d_1, \ldots, d_{k'})$ that are in the kernel of A_R, that is,

$$H(A) = \left|\left\{i \in \left\{1, \ldots, k'\right\} : d_i \in \operatorname{Ker}(A_R)\right\}\right|. \tag{3}$$

For the sequence $\mathcal{X}$ and matrix A from Example 1, we see that each edge of the access graph $G_{\mathcal{X}}$ shown in Figures 3 and 4 corresponds to a difference vector in $\mathcal{D}$. Furthermore, we note that for the parts obtained by the matrix A, the edges between parts represent the difference vectors not contained in the kernel of A_R (so the row misses) and those between vertices of the same part correspond to difference vectors in the kernel (and row hits). We also observe that the same difference vector can occur multiple times. Therefore, while the sequences can be arbitrarily long, at most 2^n unique difference vectors can occur. Hence, we can compress the information about the difference vectors of $\mathcal{X}$ by using a weight function.

Definition 3. For an access sequence $\mathcal{X}$ with difference vectors $\mathcal{D} = (d_1, \ldots, d_{k'})$, we define

$$w_{\mathcal{D}} : GF(2)^n \to \mathbb{Z}_{\geq 0}, d \mapsto \left|\left\{i \in \left\{1, \ldots, k'\right\} : d_i = d\right\}\right|.$$

For the sequence $\mathcal{X}$ and the matrix A from Example 1, we get $w_{\mathcal{D}}(\overrightarrow{7} \oplus \overrightarrow{2}) = 3$. This lets us rewrite (3) as follows:

$$|H(A)| = \sum_{d \in \operatorname{Ker}(A_R)} w_{\mathcal{D}}(d). \tag{4}$$

The following behavior of the bank- and row-bit-matrices is good to keep in mind, which combines our earlier observations with our new notation.

Observation 3. The bank-bit-matrix A_B determines the weights $w_{\mathcal{D}}$ of the binary vectors d and, in particular, if $d \notin \operatorname{Ker}(A_B)$, then $w_{\mathcal{D}}(d) = 0$. On the other hand, the row-bit-matrix A_R determines which vectors are mapped to the same row and thus yield a row hit.

We are now sufficiently prepared to present the first example we referenced earlier, but before we do, let us see how our notation relates to the typical bit flip measures used in the literature to determine good permutations. These measures condense the bit flip information of the sequence $\mathcal{X} = (x_0, \ldots, x_k)$ into single values. We want to illustrate this on the measures mentioned in [1, 2, 6, 14] for individual bits. Let the following measures be defined with respect to the jth address bit for $j \in \{1, \ldots, n\}$ and let $\mathcal{D} = (d_1, \ldots, d_{k'})$ be the sequence of difference vectors of $\mathcal{X}$, for a single bank.

Bit flip count [1, 6]	$\sum_{i=1}^{k'} d_{ij}$
Bit flip rate [2]	$\frac{1}{k'} \cdot \sum_{i=1}^{k'} d_{ij}$
Bit value ratio [14]	$\operatorname{BVR}(j) \coloneqq \frac{1}{k} \cdot \sum_{i=0}^{k} x_{ij}$
Bit flip probability [1]	$\min\left(\operatorname{BVR}(j), 1 - \operatorname{BVR}(j)\right)$

Figure 5 visualizes the bit flip count with respect to an example sequence $\mathcal{X}$ and the corresponding difference vectors derived from it. These measures are intuitive when considering mappings based on permutations: in this case, vectors mapped to the same row must coincide in the bits selected to represent the row. Thus, row misses and bit flips at row bits coincide. These measures are, of course, only heuristics since higher flip rates could still be better if they are correlated and the argumentation only applies to the case of a single bank.

x_0	0	1	1	1
x_1	1	0	1	0
x_2	1	1	0	1
x_3	1	1	0	1
x_4	0	1	1	0

$\longrightarrow$

d_1	1	1	0	1
d_2	0	1	1	1
d_3	0	0	0	0
d_4	1	0	1	1

$\downarrow$

Bit flip count	2	2	2	3

Figure 5: Visualization of the bit flip information of a sample sequence $\mathcal{X}$ in the form of difference vectors and bit flip counts (reduced bit flip information).

We complete this section with the first promised example.

Example 2. We assume we have a shared memory that is accessed by two processes. We use our previous variables n, b, r, and c to describe the memory, where we assume that $b = 0$ for simplicity. Thus, our memory has 2^n addresses. Let $s \coloneqq 2^{\frac{n}{2}}$ and n be even.

Process 1 now requests the addresses from 0 to $s - 1$, while process 2 asks for the addresses 0 to $2^n - s$ with stride s, that is, it requests $0, s, 2 \cdot s, \ldots, (s-1) \cdot s$. In vector form, these are the vectors

$$\begin{pmatrix} \overrightarrow{0} \\ \overrightarrow{z} \end{pmatrix} \text{ for } z \in \{0, \ldots, s-1\} \text{ and } \begin{pmatrix} \overrightarrow{z} \\ \overrightarrow{0} \end{pmatrix} \text{ for } z \in \{0, \ldots, s-1\}$$

where $\overrightarrow{z}$ is the binary vector of length $\frac{n}{2}$ corresponding to the number z. Since the memory is shared, we assume the actual request alternate between the two processes. The resulting sequence of difference vectors $\mathcal{D}$ now contains $2s - 1$ entries, which are

$$\begin{pmatrix} \overrightarrow{z} \\ \overrightarrow{z} \end{pmatrix} \text{ for } z \in \{0, \ldots, s-1\} \text{ and } \begin{pmatrix} \overrightarrow{z} \\ \overrightarrow{z+1} \end{pmatrix} \text{ for } z \in \{0, \ldots, s-2\}.$$

First, we show that every matrix A_R that is part of a permutation matrix achieves at most $2^{\frac{c}{2}+1}$ many row hits. Such matrices have exactly one 1-entry in each row. Let K be the columns containing these 1s, then a difference vector d is in the kernel, if and only if $d_i = 0$ for all $i \in K$. We note that $|K| = r = n - c$ and we split K into the bits $K_1 \coloneqq K \cap \left\{1, \ldots, \frac{n}{2}\right\}$ that are used by process 1 and $K_2 \coloneqq K \cap \left\{\frac{n}{2} + 1, \ldots n\right\}$ that process 2 uses.

Note that a vector

$$\begin{pmatrix} * \\ \overrightarrow{z} \end{pmatrix} \qquad \text{or} \qquad \begin{pmatrix} \overrightarrow{z} \\ * \end{pmatrix}$$

is in $\operatorname{Ker}(A_R)$ if and only if $\overrightarrow{z}_j = 1$ implies that $j \notin K_1$ or $\frac{n}{2} + j \notin K_2$, respectively. Since there are only $2^{\frac{n}{2} - |K_1|}$ and $2^{\frac{n}{2} - |K_2|}$ vectors of each type, and each yields at most two difference vectors for the kernel, we may assume $\frac{n}{2} - |K_1| \geq \frac{c}{2}$ and $\frac{n}{2} - |K_2| \geq \frac{c}{2}$. Thus, $|K_1| \leq \frac{n-c}{2}$ and $|K_1| = |K| - |K_2| \geq n - c - (\frac{n-c}{2}) = \frac{n-c}{2}$. But now, we have at most $2 \cdot 2^{\frac{n}{2} - |K_1|} = 2^{\frac{n}{2} - \frac{n-c}{2} + 1} = 2^{\frac{c}{2}+1}$ many row hits, as claimed.

But our more general model can do better. We need to specify r linearly independent rows of A_R. We use the $n-2c$ unit vectors corresponding to the bits $c+1, \dots, \frac{n}{2}, \frac{n}{2}+c, \dots, n$. For the last c rows, we use the sum of the two unit vectors for the bits j and $\frac{n}{2}+j$, for $j \in \{1, \dots, c\}$. These are linearly independent and $\mathrm{Ker}(A_R)$ consists exactly of the vectors

$$\begin{pmatrix} \overrightarrow{z} \\ \overrightarrow{z} \end{pmatrix} \text{ for } z \in \left\{0, \dots, 2^c - 1\right\}.$$

These are all part of $\mathcal{D}$, and we achieve 2^c row hits. In particular, this solution is optimal: the kernel of every feasible row-bit-matrix A_R contains at most 2^c many vectors and all our accesses are unique.

To obtain a numerical example, we can set $n = 24$ and $r = c = 12$. Then, our sequence has length $2s = 8192$ the permutation approach can achieve at most 128 row hits, meaning less than 2 % of all accesses can result in a hit. However, the presented solution for a general bijective linear mapping has 4096 hits, so exactly half the accesses are hits.

5 RESTRICTING THE SEARCH SPACE

We noted before that our problem is more restrictive than finding an arbitrary bijective function. But there are still a lot of binary matrices, even if we restrict to those of full rank. To be specific, there are $\prod_{i=0}^{n-1}(2^n - 2^i)$ many such matrices. To reduce the search space, we note that the number of row hits only depends on the kernel of A_B and A_R, not on the specific matrices themselves. This holds as the sequences $\mathcal{X}_j$ and thus the difference vectors $\mathcal{D}$ depend only on the kernel of A_B and the row hits depend only on the kernel of A_R and the vectors in $\mathcal{D}$. This yields the following observation.

Observation 4. Let $\mathcal{X}$ be an access sequence and A, A' be two binary matrices. If $\mathrm{Ker}(A_B) = \mathrm{Ker}(A'_B)$ and $\mathrm{Ker}(A_R) = \mathrm{Ker}(A'_R)$, then $H_{\mathcal{X}}(A) = H_{\mathcal{X}}(A')$.

Thus, for us, matrices with the same kernel exhibit the same behavior. Since the rank of the matrix $A_{BR} \coloneqq \begin{bmatrix} A_B \\ A_R \end{bmatrix}$ is $b + r$, its kernel has dimension $n - b - r = c$. We are thus looking for a c-dimensional subspace of $GF(2)^n$ that contains maximally many difference vectors in $\mathcal{D}$.

We denote the set of all c-dimensional subspaces of $GF(2)^n$ by $\mathcal{K}_c$. Then, we can define the row hits $|H(K)| = |H_{\mathcal{X}}(K)|$ of a subspace $K \in \mathcal{K}_c$ as the value $|H_{\mathcal{X}}(A)|$ for some matrix A whose kernel of A_{BR} is K. Such a matrix is not hard to find, as the following construction shows.

Construction 1. From the theory of error-correcting codes [8], we get a method to construct every matrix $A_{BR} \in GF(2)^{(b+r)\times n}$ that satisfies $\mathrm{Ker}(A_{BR}) = K$ for a given $K \in \mathcal{K}_c$:

(1) Take a basis $\mathcal{B} \coloneqq \{b_1, \dots, b_c\}$ of K.
(2) Create the matrix $B(K) = (b_1 \dots b_c)^\top \in GF(2)^{c \times n}$ and note that it has full row-rank since the b_i are linearly independent. The rank-nullity theorem yields $\dim(\mathrm{Ker}(B(K))) = n - c = b + r$.
(3) Now, every basis $\{c_1, \dots, c_{b+r}\}$ of $\mathrm{Ker}(B(K))$ yields the rows of a matrix $C(K) = (c_1 \dots c_{b+r})^\top \in GF(2)^{(b+r)\times n}$ for which $\mathrm{Ker}(C(K)) = K$, since $c_j^\top b_i = 0$ for all i and j. On the other hand, every row $r^\top$ of a matrix $C \in GF(2)^{(b+r)\times n}$ with $\mathrm{Ker}(C) = K$ satisfies $r \in \mathrm{Ker}(B(K))$ since $b_i^\top r = r^\top b_i = 0$ for all i, so $B(K) \cdot r = 0$.

While the row hits are unaffected by the specific choices of A_B and A_R, the number of non-zero entries most certainly is not. Since we now know what all possible matrices A_{BR} look like that satisfy $\mathrm{Ker}(A_{BR}) = K$, we can choose the one that minimizes $\|A_{BR}\|_0$.

Lemma 1. Let $K \in \mathcal{K}_c$. We can find a sparsest matrix $C \in GF(2)^{(b+r)\times n}$ such that $\mathrm{Ker}(C) = K$ in $O((b + r)2^{b+r})$ time.

Proof. By Construction 1, any such matrix C consists of rows that form a basis of $\mathrm{Ker}(B(K))$. Hence, finding a sparsest matrix with kernel K is equivalent to finding a sparsest basis of $\mathrm{Ker}(B(K))$. This subspace has dimension $b+r$, so it contains 2^{b+r} many vectors we can compute. We can now simply start with the empty set and iteratively add the vector with the fewest non-zero entries that is linearly independent of the vectors chosen so far until we get a basis. That this results in a sparsest basis of $\mathrm{Ker}(B(K))$ results from the matroid structure of the problem, which makes the Greedy algorithm optimal [5].

Computationally, we would have to sort the 2^{b+r} vectors of the kernel and then discard those that are linearly dependent. The sorting takes the specified time and the test for linear independence is easy to do by keeping track of the vectors that can be obtained by a linear combination of the chosen ones. Thus, the 2^{b+r} tests for linear independence can be done in linear time. $\square$

Note that, given a subspace $K \in \mathcal{K}_c$, the computational effort for finding a sparsest matrix $A_{BR} \in GF(2)^{(b+r)\times n}$ that satisfies $\mathrm{Ker}(A_{BR}) = K$ is comparable to the one needed to determine $|H_{\mathcal{X}}(K)|$, which is in $O(2^c)$.

With all this information at hand, we can discard our second objective and arrive at the following reformulation of Problem 1.

Problem 2.

$$\begin{aligned} \max \quad & |H(K)| \\ \text{s.t.} \quad & K \in \mathcal{K}_c \end{aligned}$$

This has reduced the search space drastically, since we no longer need to regard multiple matrices with the same kernel, like those that originate from permuting the rows. To be precise, there are

$$\prod_{i=0}^{c-1} \frac{2^{n-i} - 1}{2^{c-i} - 1}$$

subspaces of $GF(2)^n$ of dimension c [22], which is considerably smaller than the number $\prod_{i=0}^{b+r-1}(2^n - 2^i)$ of matrices A_{BR}. Since the different subspaces exhibit different behaviors, we cannot reduce the number of feasible solutions further (without making additional assumptions at least). Thus, we can use it to compare the number of feasible solutions in our general model to what can be achieved by permutations: The kernel of a permutation matrix sets $b + r$ bits to 0, so we get one kernel per choice of these bits and there are $\binom{n}{b+r} = \binom{n}{c}$ many.

Example 3. Let $n = 6$ and $c = 3$. In this case

$$\frac{\binom{n}{c}}{\prod_{i=0}^{c-1} \frac{2^{n-i}-1}{2^{c-i}-1}} \approx 0.01434$$

Therefore, permutation matrices make up only around 1, 4% of all feasible solution to Problem 2 for this small dimension. For bigger n and c, the fraction is even smaller, for example, for $n = 12$ and $c = 6$ only about $4\mathrm{e}^{-7}$ % of all solutions originate from permutations.

We also note that this behavior is even more pronounced for our search space reduction. For $n = 6$ and $c = 3$, the number of subspaces only amounts to about 0.5 % of the possible matrices of full rank. For $n = 12$ and $c = 6$, it is only around $5\mathrm{e}^{-9}$ %.

We end this section with a final remark. Note that we could have allowed subspaces of dimension less than c in Problem 2, since the function H satisfies $H(K) \supseteq H(K')$ for every subspace K' of K. From this point of view, the problem turns into a so-called height-constrained subspace selection problem, which is a well-studied generalization of many classical subspace selection problems like the principal component analysis and the sparse dictionary selection problem, as described in [17]. The authors there showed that, if the function $K \mapsto |H(K)|$ is DR-submodular with additive gap δ, then the greedy algorithm is guaranteed to find a $(1 - e^{-1}, \delta\,(1 - e^{-1})\,c)$-approximate solution. Unfortunately, our objective does not have this property and we get no approximation guarantee. Nonetheless, we shall look at a greedy algorithm in Section 6.2.

6 THE SINGLE BANK CASE

We now take a closer look at the case in which we have a single bank and assume, until the end of this section, that $b = 0$. In this case, A_B is the unique matrix in $GF(2)^{0\times n}$ and we are just trying to find an optimal choice for A_R, or using the terminology of Problem 2, an optimal subspace $K \in \mathcal{K}_c$. Since the difference vectors in $\mathcal{D}$ do not depend on A_R, the function $w_{\mathcal{D}}$ is fixed and, by (4), the objective function $|H(K)|$ is given by

$$|H(K)| = \sum_{d\in K} w_{\mathcal{D}}(d). \tag{5}$$

Since, in many cases, lots of the values $w_{\mathcal{D}}(d)$ will be zero, we define the set of difference vectors $\mathcal{D}' \coloneqq \{d \in GF(2)^n\colon\ w_{\mathcal{D}}(d) > 0\}$. Using this notation, we can adjust our formulation one final time and obtain the problem below.

Problem 3.

$$\begin{aligned} \max \quad & \sum_{d\in\mathcal{D}'\colon d\in K} w_{\mathcal{D}}(d) \\ \text{s.t.} \quad & K \in \mathcal{K}_c \end{aligned}$$

We assume the input of the problem are the vectors $d \in \mathcal{D}'$ and their weights $w_{\mathcal{D}}(d)$ and the subspace K is encoded by a basis. In this case, we can show that the problem is NP-hard (and that the row miss minimization variant is hard to approximate by a factor better than 2, at least under certain assumptions).

Theorem 1. Problem 3 is NP-hard.

The proof of this theorem can be found in Appendix A.

Next, we want to show that we can determine a lower and an upper bound on the number of row hits achievable by an address mapping based on a binary matrix of full rank.

Observation 5. Let $\mathcal{X}$ be an access sequence with corresponding set of difference vectors $\mathcal{D}' = \{d_1, \ldots, d_l\}$. We assume that $w_{\mathcal{D}}(d_1) \geq \ldots \geq w_{\mathcal{D}}(d_l)$ and let $l^* \in \{1, \ldots, l\}$ be the maximal index that satisfies $K^* \coloneqq \mathrm{span}(\{d_i\colon\ i \leq l^*\}) \in \mathcal{K}_c$. Since this is a feasible solution, $\underline{z} \coloneqq |H(K^*)|$ is a lower bound for the number of row hits. On the other hand, since every $K \in \mathcal{K}_c$ contains 2^c elements, $\overline{z} \coloneqq \sum_{i=1}^{\max\{l,2^c\}} w_{\mathcal{D}}(d_i)$ is an upper bound.

As a result, every feasible subspace K satisfies $\underline{z} \leq |H(K)| \leq \overline{z}$. Moreover, the running time for computing these bounds is dominated by the required sorting, which can be done in $O(l \log l)$ time.

One further advantage of our model in the single bank case is that it can be extended to probabilistic traces. Note that Problem 3 has the linear objective we determined in (5). If we are given a probabilistic access sequence $\mathcal{X}_{\text{prob}} \coloneqq (X_0, \ldots, X_k)$, where the X_i are random variables with values in $GF(2)^n$, we get a random sequence of difference vectors $\mathcal{D}_{\text{prob}} = (D_1, \ldots, D_k) = (X_1 \oplus X_0, \ldots X_k \oplus X_{k-1})$. We would now like to maximize the expected number of row hits.

In contrast to the deterministic case, the weights $w_{\mathcal{D}_{\text{prob}}}(d)$ for $d \in GF(2)^n$ are now also random variables: we simply replace the vector d_i in Definition 3 by the random variable D_i. Maximizing the row hits in expectation now simply means that we want to maximize

$$K \mapsto \mathbb{E}\left[\sum_{d\in K} w_{\mathcal{D}_{\text{prob}}}(d)\right] = \sum_{d\in K} \mathbb{E}\left[w_{\mathcal{D}_{\text{prob}}}(d)\right]$$

where the equality follows from the linearity of the expectation.

Therefore, maximizing the number of row hits in expectation is equivalent to solving an instance of Problem 3 where the weight $w_{\mathcal{D}}(d)$ of the difference vector d is determined by its expected number of occurrences $\mathbb{E}\left[w_{\mathcal{D}_{\text{prob}}}(d)\right]$. Let us illustrate this with a simple example.

Example 4. We define a random sequence $\mathcal{X}_{\text{prob}} = (X_0, \ldots, X_k)$ as follows. Let $p_1, \ldots, p_n \in [0, 1]$ be given, where p_j is the probability that the jth bit flips between two subsequent accesses (i.e. changes from 0 to 1 or the other way around). The access X_0 is an arbitrary vector, say the zero-vector. For $i \in \{1, \ldots, k\}$ we define the random variable $T(i) \in GF(2)^n$ which is a Bernoulli vector whose jth component is Bernoulli distributed with probability p_j and everything is independent. If we set $X_i \coloneqq X_{i-1} \oplus T(i)$, for $i \in \{1, \ldots, k\}$, then $\mathcal{D}_{\text{prob}} = (T(1), \ldots, T(k))$.

For $d \in GF(2)^n$, we can compute the expectation

$$\mathbb{E}\left[w_{\mathcal{D}_{\text{prob}}}(d)\right] = \sum_{i=1}^{k} \mathbb{P}\left(T(i) = d\right) = k \cdot \prod_{j\colon d_j=1} p_j \cdot \prod_{j\colon d_j=0} (1 - p_j).$$

Thus, the optimal solution (in expectation) for this random sequence can be computed by solving the deterministic Problem 3.

Note that the probabilistic sequences above can be used to generate examples where we expect permutations to perform badly. We already saw that if A_R is part of a permutation matrix, then it maps two vectors x_{i-1} and x_i to the same row if and only if they coincide on certain bits. Let I be the bits that need to coincide, then $|I| = r$. If the flip probability p_j of the bit $j \in I$ is high, then many subsequent vectors will be placed in different rows. Even worse, when the probabilities are independent, these all multiply: Two subsequent vectors are mapped to the same row if and only if none of these bits flip, which occurs with probability $\prod_{j\in I}(1-p_j)$. Thus, if we do not have sufficiently large flip probabilities, most accesses will result in row misses.

We take a look at a very small example.

Example 5. Let $n = 6$, $r = 3$, and $c = 3$. We generate two random sequences $\mathcal{X}_l$ and $\mathcal{X}_h$ as described in Example 4. For $\mathcal{X}_l$ we use low bit flip probabilities of 0.3 for all bits, while for $\mathcal{X}_h$ we use high probabilities of 0.6. For both sequences, we evaluate all possible solutions to Problem 1, so all matrices in $GF(2)^{r\times n}$, in terms of their row misses and number of ones. The corresponding results are shown in Figure 6. Note that we use row misses instead of row hits in the plot for ease of reading (since this makes the problem a minimization problem in both objectives and points are better the closer they are to 0 on both axes).

The plots both depict the objective values of the feasible matrices of Problem 1 and the red dots are those that are "Pareto-optimal", meaning that there is no other solution that is better in one objective without being worse in another. So they cannot be strictly improved.

The graphics clearly show that permutation matrices perform well for low bit flip ratios: For the sequence $\mathcal{X}_l$ where the difference vectors are sparse, a permutation matrix yields an optimal solution to the row misses (or hits). Since these also have a minimum number of ones, two in this case, they are the only Pareto-optimal solution.

On the other hand, the objective values of the permutation matrices for the sequence $\mathcal{X}_h$ must also be on the line with three 1-entries. Thus, in terms of row misses, these solutions are actually amongst the worst ones, and all common techniques (like heuristic-based on bank bit flip ratios, entropy measures and exact approaches) would have failed to determine a satisfactory solution. But with access to all linear functions, we can obtain a significantly better mapping with fewer row misses.

The rest of this section is dedicated to solving Problem 3. More precisely, we give an ILP-formulation for the problem in Section 6.1 and present the Greedy algorithm in Section 6.2.

6.1 An Exact ILP-based Approach

In Theorem 1 we have established the NP-hardness of the linear address mapping problem. In this subsection we provide an ILP-formulation for Problem 3 in order to attempt to solve it exactly via an LP-based branch and bound algorithm. Consider the following integer program for a given sequence $\mathcal{X} = (x_0, \dots, x_k)$ and its set of difference vectors $\mathcal{D}' = \{d_1, \dots, d_l\}$. Let $D \in GF(2)^{n\times l}$ be the matrix $\begin{bmatrix} d_1 & \dots & d_l \end{bmatrix}$.

With this notation, we can formulate Problem 3 as follows:

$$\max \sum_{i=1}^{l} w_{\mathcal{D}}(d_i) \cdot y_i \tag{6}$$

$$\sum_{p=1}^{c} \lambda_i^p (D \cdot x^p) = d_i y_i + 2z_i, \qquad \text{for all } i \in \{1, \dots, l\} \tag{7}$$

$$\sum_{i=1}^{l} x_i^p = 1 \qquad \text{for all } p \in \{1, \dots, c\} \tag{8}$$

$$y,\, x^p,\, \lambda^p \in \{0,1\}^l \qquad \text{for all } p \in \{1, \dots, c\} \tag{9}$$

$$z_i \in \mathbb{Z}^n_{\geq 0} \qquad \text{for all } i \in \{1, \dots, l\} \tag{10}$$

To see that this describes our problem, we use the following interpretation. We want to represent the c-dimensional subspaces by a selection of c many difference vectors. (Again, to reach dimension c exactly, we would have to require linear independence, but allowing lower-dimensional spaces does not make the solution incorrect.) The binary variables x^p specify this selection: By Constraint (8), each x^p contains exactly one 1-entry $i(p)$, meaning we choose $d_{i(p)}$ as the pth vector. Let K be the subspace generated by the vectors we selected, that is $K = \operatorname{span}(d_{i(1)}, \dots, d_{i(c)})$.

The binary variables y_i should only be set to 1 if d_i is in K. This is ensured by Constraint (7): If $y_i = 1$, then there exist coefficients λ_i^p for $p = 1, \dots, c$ such that

$$d_i = \sum_{p=1}^{c} \lambda_i^p (D \cdot x^p) - 2z_i = \sum_{p=1}^{c} \lambda_i^p d_{i(p)} - 2z_i.$$

This is what we wanted (and the $2z_i$ term originates from the problem using addition in the real numbers and not in $GF(2)$). If d_i is not in K, we can always set all values to 0.

In the objective (6) we maximize the sum of weights of the vectors in K, so $|H(K)|$, since these have a 1-entry as their y-variable. We can thus assume that $K \in \mathcal{K}_c$ (since we have already observed that $|H(K)|$ does not decrease when adding to the subspace). In particular, we obtain an optimal solution to Problem 3.

Finally, note that our formulation is not an ILP since the first constraint contains the quadratic terms $\lambda_i^p \cdot x_j^p$. But we can linearize these by introducing a variable $f_{ij}^p = \lambda_i^p \cdot x_j^p \in \{0,1\}$ to represent this term and adding the constraints

$$\begin{aligned} f_{ij}^p &\leq \lambda_i^p, \\ f_{ij}^p &\leq x_j^p, \\ f_{ij}^p &\geq \lambda_i^p + x_j^p - 1, \end{aligned}$$

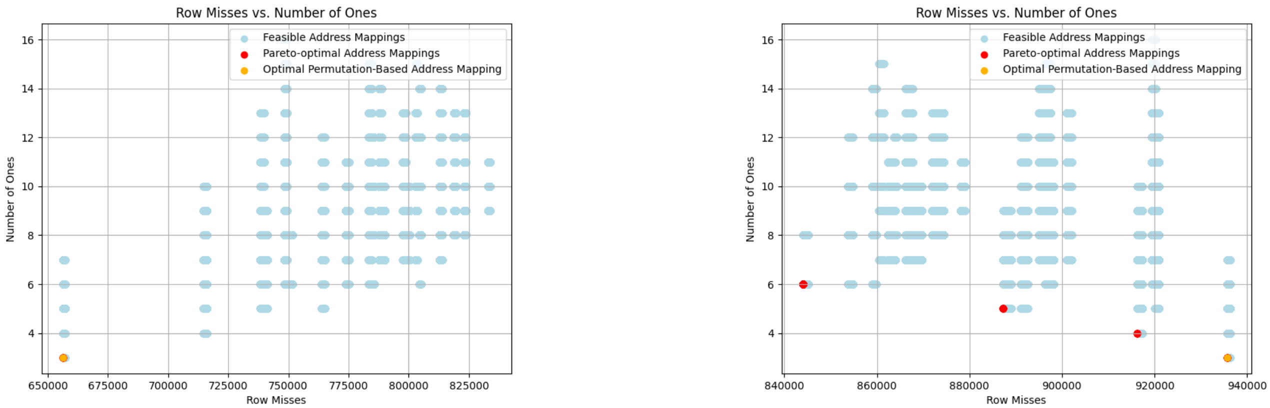
for $i, j \in \{1, \dots, l\}$ and $p \in \{1, \dots, c\}$. The first two constraints ensure that $f_{ij}^p = 0$ if $\lambda_i^p = 0$ or $x_i^p = 0$, while the third ensures that it is 1 if both are, which is the desired behavior.

The above ILP allows to solve Problem 3 exactly. However, a straightforward approach using state-of-the-art solvers such as `gurobi` [7] is currently feasible only for comparatively small traces.

6.2 A Heuristic Greedy-Based Approach

To solve larger traces, we propose the following greedy approach: It starts with the 0-dimensional subspace $K = \{0\}$ and iteratively extends it until it has dimension c. To choose the vector d with which to extend, we could choose the one that has the highest weight $w_{\mathcal{D}}(d)$. However, by adding d to K, we get all $2^{\dim(K)}$ vectors $d \oplus k$ for $k \in K$, and not just d. Thus, we choose the vector whose addition to the kernel is most profitable in each step, that is, the one that maximizes $|H(\operatorname{span}(K \cup \{a_i\}))|$. This procedure is shown in Algorithm 1.

Furthermore, we can use the result obtained from the Greedy algorithm shown in Algorithm 1 and subsequently apply the sparsity optimization described in Lemma 1. Figure 7 illustrates the sparsity optimization for $\mathcal{X}_h$ from Example 5. We observe that



(a) The solutions for the sequence $\mathcal{X}_l$. **(b) The solutions for the sequence $\mathcal{X}_h$.**

Figure 6: Considering Problem 1 for $n = 6$, $r = 3$, and $c = 3$. The address sequences used for the two instances are realizations of the stochastic process described in Example 4. In the plots contain a point for each solution of Problem 1 that shows its value in both objectives. The red dots are those that cannot be strictly improved.

Algorithm 1 A greedy algorithm for Problem 3.

$K \leftarrow \{0\}$, $V \leftarrow GF(2)^n$
for $i = 1, \ldots, c$ **do**
  Let $a_i \in \arg\max \{|H(\operatorname{span}(K \cup \{a_i\}))| : a \in V\}$
  $K \leftarrow \operatorname{span}(K \cup \{a_i\})$, $V \leftarrow V \setminus K$
end for
return K

the permutation-based matrix and the solution from the sparsity-optimized greedy procedure make up the two lexicographically optimal solutions for the corresponding instance of Problem 1.

Figure 7: Applying sparsity-optimized Greedy algorithm for the sequence $\mathcal{X}_h$ from Example 5.

7 DEALING WITH MULTIPLE BANKS

In the last section, we took a closer look at our model in the case of a single bank. Now, we show that we can reduce the general case with banks to the one without, if we fix the bank-bit-matrix. Thus, we assume that we are given a suitable bank-bit-matrix A_B.

By Observation 3, if the matrix A_B is fixed, then the function $w_{\mathcal{D}}$ is predetermined and the row hits depend on the matrix A_R as described in (4). In a first step, we show that we can restrict A_B to the simplest choice, namely to $\begin{bmatrix} I & 0 \end{bmatrix}$ where I is the $b \times b$ identity matrix. To see that this can be done without loss of generality, note that the rows $a_1, \ldots, a_b$ of A_B are linearly independent. We extend these to a basis $(a_1^\top, \ldots, a_n^\top)$ of $GF(2)^n$ and define

$$T = \begin{bmatrix} a_1 \\ \vdots \\ a_n \end{bmatrix} \text{ to get } A_B = \begin{bmatrix} I & 0 \end{bmatrix} \cdot T.$$

As a result, we have that a vector d is in the kernel A_B if and only if $T \cdot d$ is in the kernel of $\begin{bmatrix} I & 0 \end{bmatrix}$. Thus, by potentially modifying $\mathcal{D}$ or $\mathcal{X}$ by a multiplication with T, we may assume that

$$A_B = \begin{bmatrix} I & 0 \end{bmatrix}.$$

We are now looking for a matrix A_R that maximizes the weights $w_{\mathcal{D}}$ of the vectors in its kernel. We also recall from Observation 3 that if a difference vector d is not in the kernel of A_B, then $w_{\mathcal{D}}(d) = 0$. Thus, we are only interested in the vectors $d \in \operatorname{Ker}(A_R)$ that also satisfy $d \in \operatorname{Ker}(A_B)$, or equivalently, we are looking for the vectors in $\operatorname{Ker}(A_{BR})$. In our simple case, this simply means that the first b many entries $d_1, \ldots, d_b$ of d must all be 0.

We now notice that, for a binary vector d,

$$d \in \operatorname{Ker}(A_{BR}) \Leftrightarrow d \in \operatorname{Ker} \begin{bmatrix} I & 0 \\ * & A_R' \end{bmatrix} \Leftrightarrow d \in \operatorname{Ker} \begin{bmatrix} I & 0 \\ 0 & A_R' \end{bmatrix}.$$

Consequently, we may assume that A_R is of the form

$$A_R = \begin{bmatrix} 0 & A_R' \end{bmatrix}, \text{ where } 0 \in GF(2)^{r \times b} \text{ and } A_R' \in GF(2)^{r \times (n-b)}.$$

Since we know that the first b components need to be 0 anyway for d to be in the kernel of A_B, we can reduce the dimension and focus on finding an optimal matrix A'_R that maximizes the weight of the vectors in its kernel. These weights are simply the weights $w_{\mathcal{D}}(d)$, though we formally need to drop the leading zeros (and discard all the entries which do not have b many leading zeros). But the resulting problem is exactly an instance of Problem 3 (in the dimension $n-b$).

We have now seen that for a fixed bank-bit-matrix, we can solve the problem using an algorithm for the single bank case. At the same time, both our bounds as well as extension to probabilistic access sequences can be directly transferred. However, one last question remains: how do we choose a good bank-bit mapping A_B?

One method could be to find a good representative set of candidate matrices for A_B and compute the corresponding row- and column-bit matrix by solving the corresponding instances of Problem 3. Such a possible set of candidate bank-bit-matrices could be bit-selecting matrices which select b bank bits, as in [18].

That good bank-bit-matrices can make a difference is illustrated in Figure 8, which shows how the bank bit mapping A_B influences the number of achievable row hits for a sequence $\mathcal{X}$. The figure shows the sets $\mathcal{D}'$ for two different bank bit mappings A_B that use 2 bank bits for addressing the bank of the address vectors. In it, we see (or can compute) that the number of row hits achievable for the right histogram is almost twice the number of row hits achievable for the left histogram. Here, the number of achievable row hits refers to the upper bound determined in Observation 5. This is the case as an even distribution of the difference vectors means that we incur a lot of misses on the vectors that are not in the kernel, while few expensive vectors that end up in the kernel can create a lot of hits.

8 EXPERIMENTAL RESULTS

So far, we have hinted at the theoretical advantages of the general BIM-based approach over the permutation-based approach, as discussed in Examples 2 and 5. Additionally, we proposed an exact ILP-based method for solving Problem 3, which, in its current form, is unfortunately limited to relatively small instances.

To address this, we introduced a heuristic greedy algorithm for the same problem. In what follows, we demonstrate that this greedy approach already yields surprisingly good results for real-world applications. Moreover, we highlight its potential to improve row hit ratios in access patterns derived from concurrent DRAM accesses that, for example, frequently arise in GPU-related applications.

The remainder of this section is structured as follows: In Section 8.1, we show how our greedy algorithm for the single-bank case can be used to derive BIMs suitable for handling concurrent memory accesses. In Section 8.2, we further demonstrate the practical applicability of the generalized greedy algorithm for the multi-bank case, using some common access patterns from embedded systems applications.

8.1 Concurrent Memory Access Traces

In this section, we evaluate the address mapping computed by Algorithm 1 using synthetic workloads with interleaved, single-word memory accesses from multiple initiators, as already described in Example 2 for the case of 2 initiators. We assume a simplified DRAM setting consisting of a single memory bank with 24-bit addresses, split evenly into 12 row bits and 12 column bits. This setup is intentionally minimal and serves to illustrate the behavior of our mapping under structured stress conditions.

Each initiator follows a distinct striding pattern: let $i = 0, \dots, k-1$ denote the i-th initiator out of k total. The access stride of initiator i is defined as $2^{i \cdot \frac{n}{k}}$, where n is the number of address bits (here, $n = 24$). We generate workloads with a sequence length of 1 million addresses. This type of workload models a relevant scenario in shared-memory systems where multiple initiators (e.g., CPU cores or hardware threads) issue interleaved memory requests, with varying stride lengths.

This mixed access pattern challenges traditional, permutation-based address mappings as these traces, although structured, exhibit high toggling rates for many address bits. In Table 1 we applied Algorithm 1 as well as the algorithm described in [18], to which we will refer to as ConGen2, and performed computations for interleaved traces with 2, 3, and 4 initiators. We compare the results with a baseline, which we chose to be an RBC address mapping scheme, which is a permutation-based address mapping scheme. ConGen2 is an exact algorithm which yields an optimal row-hit maximizing permutaion-based address mapping scheme. Therefore, it always achieves a better or equal row hit ratio compared to RBC. However, the row hit ratios achieved by ConGen2 and RBC are relatively small. On the other hand, our greedy approach yields an non-permutation-based address mapping which is able to achieve 1.4 times the row hit ratio, for the case of 4 initiators, and even up to 32 times the row hit ratio achieved by ConGen2 for 2 initiators. Our evaluation demonstrates that the mapping computed by Algorithm 1 may perform effectively under concurrent accesses to a shared memory, showing its potential to handle diverse and interleaved access patterns. Furthermore, the upper bound computed for the trace generated by two initiators is 500287, showing that greedy is likely optimal for this case. On the other hand, the upper bound computed for the remaining 2 traces is 999999, which is a trivial bound. Therefore, we cannot say how close the solution computed by the greedy algorithm is to the optimal solution. This also underlines the necessity for improving the ILP-formulation of the problem in future work.

8.2 Additional Common Access Traces

In this section, we want to demonstrate the practical applicability of our proposed greedy approach for the multiple banks case, described in Section 7, for some additional common access traces.

More precisely, the greedy approach for multiple banks is the following: We first choose a set of bank-bit-matrices and then, for each such matrix, we apply Algorithm 1 to the corresponding instance of Problem 3 as discussed in Section 7. Among the found ones, we then take the best solution.

Here, we choose the same set of candidate bank-bit matrices as for permutation-based address mappings, namely bit-selecting matrices. We note that the set of feasible solutions for this extended problem includes all permutation-based address mappings. Suppose that we would have used an exact algorithm for Problem 3 instead of Algorithm 1, then this way we would always obtain an address

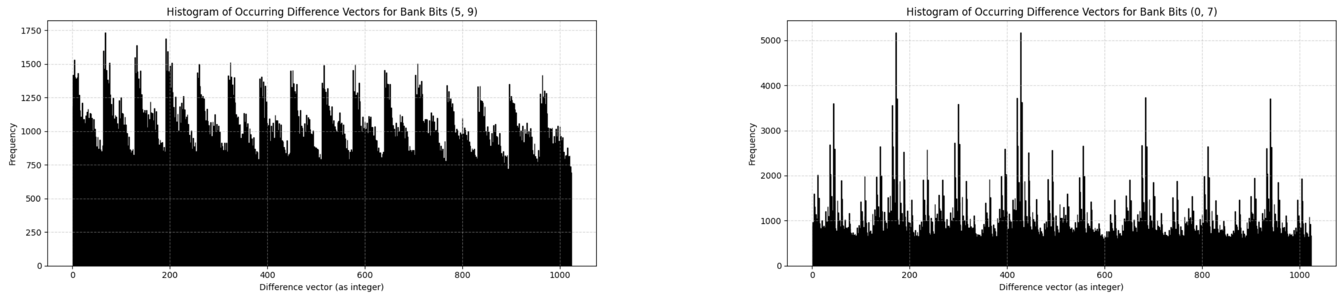


Figure 8: Histogram comparison for a sequence of 12**-bit addresses with sequence length one million generated as in Example 4 for the following bit flip probabilities:** 0.4, 0.4, 0.5, 0.6, 0.3, 0.7, 0.3, 0.4, 0.6, 0.8, 0.4, 0.3**. Each histogram represents the set of difference vectors $\mathcal{D}'$ induced by different bank-bit-matrices. On the left: bank bits (5,9), on the right: bank bits (0,7). Assuming a DRAM consisting of 4 banks, 32 rows and 32 columns, the upper bound on the achievable row hits for the left difference vectors is 47972 and for the right ones it is 94974.**

Table 1: Comparison of Greedy, ConGen2 [18], and the baseline mapping scheme (RBC) for interleaved traces with a varying number of initiators.

Category	**Sequence**	**Sequence Length**	**Row Hit Rate [%]**		
			Greedy[our]	**ConGen2** [18]	**RBC**
Interleaved Traces	2 Initiators	1 Mio.	50 %	1.562%	0.025%
	3 Initiators	1 Mio.	35.418%	8.596%	2.345%
	4 Initiators	1 Mio.	38.282%	26.172%	26.172%

mapping which is equally good or better than the best permutation-based matrix.

However, as our greedy algorithm is only a heuristic we do not expect to get an optimal solution for Problem 3 for a fixed bank bit mapping. Nevertheless, we want to show that the row hit ratios for the mappings determined by the greedy approach for multiple banks almost match the row hit ratio of the best permutation matrix which, in combination with the results obtained in Section 8.1, showcases the robustness of the greedy approach with respect to various access patterns.

To this end, we assume a realistic DRAM, as described in [18]. More precisely, we assume $n = 24$ available bits to be used by the algorithms, $b = 3$ bank, $r = 14$ row and $c = 7$ column bits. We implemented the multiple banks greedy described above in python and parallelized the bank-bit enumeration using OpenMPI, which enabled a parallel execution on a high performance computing cluster.

We performed experiments using two representative trace for each category of the same bandwidth demanding applications as in [12, 18], which all produce deterministic memory access patterns:

- Image Rotation: An image with a resolution of 1024×576 pixels is rotated 90 °. Therefore, the image is written into the DRAM in x-direction and is read in y-direction. We tested such traces for 8 and 32 bit pixel sizes.
- Convolutional Neural Network (CNN): An optical neighborhood operation is performed on an image with a resolution of 1024×576 pixels and a pixel size of 32 bit. We tested such traces for 5×5 and 7×7 kernel size.
- 3D Image Rotation: A three-dimensional image with 128×128×128 voxels is rotated. We tested this category of traces for 8 and 32 bit pixel sizes.

We observed the longest execution time for the CNN workload with the $7x7$ kernel size, which took around 73 minutes to compute. The lowest execution time was obtained for the 8-bit rotation workload, which took approximately 4 minutes to compute. Note that the computation time of our greedy approach solely depends on the number of non-zero weight difference vectors that occur for a fixed bank bit mapping, while the execution time of ConGen2 also depends on the trace length. The results obtained are summarized in Table 2. Furthermore, we observe that the naive baseline, given by the RBC mapping scheme yields moderate results for the CNN workloads and low row hit rates for the (3D) Rotation workloads. On the other hand, the exact permutation-based address mapping determined by ConGen2 is able to achieve over 90 % row hits for all workloads, strongly indicating that permutation-based address mappings are already optimal with respect to Problem 3 for these access traces. Although our heuristic multiple banks greedy approach does not find the optimal solution, we see that the row hit ratio almost matches the quality of ConGen2, with row hit ratios over 90 % and only marginal deviations from the row hit ratios of solutions obtained by ConGen2. As already mentioned, this is due to the fact that the greedy approach is only a heuristic. If we were to replace the usage of Algorithm 1 with an exact algorithm for Problem 3 (such as e.g. with the proposed ILP-based algorithm) in combination with the enumeration of all bit-selecting matrices, we would always get a solution that achieves at least as many row hits as the optimal solution computed by ConGen2.

9 CONCLUSION AND FUTURE WORK

We presented a new mathematical model for computing application-specific row hit maximizing (respectively, row miss minimizing) BIM-based address mappings, which have minimal number of one-entries. We proposed an exact ILP formulation for the model. While it turns out that a straightforward implementation using off-the-shelf solvers is currently only feasible for solving relatively small instances, we also designed a heuristic greedy algorithm for solving larger instances. The provided bounds in our framework certify that this heuristic already achieves good results, in particular outperforming standard permutation-based approaches for access traces resulting from concurrent accesses to memory.

Furthermore, we observed that our general model encompasses permutation-based address mappings, which represent only a small subset of all feasible solutions, thereby making it a natural generalization of the mathematical model developed in [18]. We saw that, especially for traces stemming from random or concurrent accesses to memory, general BIMs may yield considerable row hit ratio improvements. More specifically, we observed that a marginal increase in implementation complexity, in the form of additional XOR gates (reflected by the number of one-entries in the matrix), enables a notable improvement in performance.

Further, we also saw how the consideration of the full bit flip information in the form of a vector space structure and corresponding vector weight for a given trace yields interesting insights. These enable the computation of trace-specific upper bounds on the number of row hits achievable with a general BIM-based approach as well as the possibility of extending the consideration of deterministic traces to probabilistic ones.

In future work, we plan to improve the ILP-formulation provided for Problem 3 in order to be able to solve bigger instances to proven optimality. Additionally, we plan to conduct a rigorous analysis of probabilistic access traces using, for example, Monte Carlo analysis.

ACKNOWLEDGMENTS

Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - GRK 2982, 516090167 “Mathematics of Interdisciplinary Multiobjective Optimization”. Furthermore, this work was partly funded by the Federal Ministry of Research, Technology and Space (BMFTR) under grant 16ME0934K (DI-DERAMSys).

REFERENCES

[1] Shashank Adavally and Krishna Kavi. 2021. Towards Application-Specific Address Mapping for Emerging Memory Devices. In *Proceedings of the International Symposium on Memory Systems* (Washington, DC, USA) *(MEMSYS '20)*. Association for Computing Machinery, New York, NY, USA, 105–113. https://doi.org/10.1145/3422575.3422785

[2] Berkin Akin, Franz Franchetti, and James C. Hoe. 2015. Data reorganization in memory using 3D-stacked DRAM. *SIGARCH Comput. Archit. News* 43, 3S (June 2015), 131–143. https://doi.org/10.1145/2872887.2750397

[3] Rachata Ausavarungnirun, Kevin Kai-Wei Chang, Lavanya Subramanian, Gabriel H. Loh, and Onur Mutlu. 2012. Staged Memory Scheduling: Achieving High Performance and Scalability in Heterogeneous Systems. In *Proceedings of the 39th Annual International Symposium on Computer Architecture (Isca '12)*. IEEE Computer Society, Washington, DC, USA, 416–427.

[4] N. Deo. 1975. Graph Theory with Aapplications to Engineering and Computer Science. *Networks* 5, 3 (July 1975), 299–300. https://doi.org/10.1002/net.1975.5.3.299

[5] Jack Edmonds. 1971. Matroids and the greedy algorithm. *Mathematical Programming* 1, 1 (1971), 127–136. https://doi.org/10.1007/BF01584082

[6] Mohsen Ghasempour, Aamer Jaleel, Jim D. Garside, and Mikel Luján. 2016. DReAM: Dynamic Re-arrangement of Address Mapping to Improve the Performance of DRAMs. In *Proceedings of the Second International Symposium on Memory Systems* (Alexandria, VA, USA) *(MEMSYS '16)*. Association for Computing Machinery, New York, NY, USA, 362–373. https://doi.org/10.1145/2989081.2989102

[7] Gurobi Optimization, LLC. 2024. Gurobi Optimizer Reference Manual. https://www.gurobi.com

[8] W. C. (William Cary) Huffman and Vera. Pless. 2003. *Fundamentals of error-correcting codes / W. Cary Huffman and Vera Pless.* Cambridge University Press, Cambridge, U.K. ;.

[9] Ibrahim Hur and Calvin Lin. 2004. Adaptive History-Based Memory Schedulers. In *Proceedings of the 37th Annual IEEE/ACM International Symposium on Microarchitecture (Micro 37)*. IEEE Computer Society, Washington, DC, USA, 343–354. https://doi.org/10.1109/MICRO.2004.4

[10] Bruce Jacob, Spencer Ng, and David Wang. 2008. *Memory Systems: Cache, DRAM, Disk.*

[11] Matthias Jung, Sven O. Krumke, Christof Schroth, Elisabeth Lobe, and Wolfgang Mauerer. 2025. QCEDA: Using Quantum Computers for EDA. In *Embedded Computer Systems: Architectures, Modeling, and Simulation*, Luigi Carro, Francesco Regazzoni, and Christian Pilato (Eds.). Springer Nature Switzerland, Cham, 32–46.

[12] Matthias Jung, Deepak M. Mathew, Christian Weis, Norbert Wehn, Irene Heinrich, Marco V. Natale, and Sven O. Krumke. 2016. ConGen: An Application Specific DRAM Memory Controller Generator. In *Proceedings of the Second International Symposium on Memory Systems* (Alexandria, VA, USA) *(MEMSYS '16)*. Association for Computing Machinery, New York, NY, USA, 257–267. https://doi.org/10.1145/2989081.2989131

[13] Xingchen Li, Zhihang Yuan, Yijin Guan, Guangyu Sun, Tao Zhang, Rongshan Wei, and Dimin Niu. 2022. Flatfish: A Reinforcement Learning Approach for Application-Aware Address Mapping. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41, 11 (2022), 4758–4770. https://doi.org/10.1109/TCAD.2022.3146204

[14] Yuxi Liu, Xia Zhao, Magnus Jahre, Zhenlin Wang, Xiaolin Wang, Yingwei Luo, and Lieven Eeckhout. 2018. Get Out of the Valley: Power-Efficient Address Mapping for GPUs. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. 166–179. https://doi.org/10.1109/ISCA.2018.00024

[15] Pasin Manurangsi. 2018. Inapproximability of Maximum Biclique Problems, Minimum k-Cut and Densest At-Least-k-Subgraph from the Small Set Expansion Hypothesis. *Algorithms* 11, 1 (2018). https://doi.org/10.3390/a11010010

[16] Onur Mutlu and Thomas Moscibroda. 2008. Parallelism-Aware Batch-Scheduling: Enhancing Both Performance and Fairness of Shared DRAM Systems. In *35th International Symposium on Computer Architecture (ISCA)*. Association for Computing Machinery, Inc.

[17] So Nakashima and Takanori Maehara. 2019. Subspace Selection via DR-Submodular Maximization on Lattices. *Proceedings of the AAAI Conference on Artificial Intelligence* 33, 01 (Jul. 2019), 4618–4625. https://doi.org/10.1609/aaai.v33i01.33014618

[18] Marco V. Natale, Matthias Jung, Kira Kraft, Frederik Lauer, Johannes Feldmann, Chirag Sudarshan, Christian Weis, Sven Krumke, and Norbert Wehn. 2021. Efficient Generation of Application Specific Memory Controllers. In *Proceedings of the International Symposium on Memory Systems* (Washington, DC, USA) *(MEMSYS '20)*. Association for Computing Machinery, New York, NY, USA, 233–247. https://doi.org/10.1145/3422575.3422796

[19] Christos Papadimitriou and Mihalis Yannakakis. 1988. Optimization, approximation, and complexity classes. In *Proceedings of the twentieth annual ACM symposium on Theory of computing - STOC '88 (STOC '88)*. ACM Press, 229–234. https://doi.org/10.1145/62212.62233

[20] Scott Rixner, William J. Dally, Ujval J. Kapasi, Peter Mattson, and John D. Owens. 2000. Memory Access Scheduling. In *Proceedings of the 27th Annual International Symposium on Computer Architecture (Isca '00)*. ACM, New York, NY, USA, 128–138. https://doi.org/10.1145/339647.339668

[21] Gert-Jan van den Braak, Juan Gómez-Luna, José María González-Linares, Henk Corporaal, and Nicolás Guil. 2016. Configurable XOR Hash Functions for Banked Scratchpad Memories in GPUs. *IEEE Trans. Comput.* 65, 7 (2016), 2045–2058. https://doi.org/10.1109/TC.2015.2479595

[22] H. Vandierendonck and K. De Bosschere. 2005. XOR-based hash functions. *IEEE Trans. Comput.* 54, 7 (2005), 800–812. https://doi.org/10.1109/TC.2005.122

[23] Vijay V. Vazirani. 2003. *Approximation Algorithms.* Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-662-04565-7

[24] Rongshan Wei, Chenjia Li, Chuandong Chen, Guangyu Sun, and Minghua He. 2021. Memory Access Optimization of a Neural Network Accelerator Based on Memory Controller. *Electronics* 10, 4 (2021). https://doi.org/10.3390/electronics10040438

[25] Wm. A. Wulf and Sally A. McKee. 1995. Hitting the Memory Wall: Implications of the Obvious. *SIGARCH Comput. Archit. News* (March 1995). https://doi.org/

Table 2: Comparison of Greedy, ConGen2 [18], and the baseline mapping scheme (RBC) for various traces from typical applications in embedded systems.

Category	Sequence	Sequence Length	Row Hit Rate [%]		
			Greedy[our]	**ConGen2** [18]	**RBC**
CNN	7x7	6187164	98.245%	99.986%	81.224%
	5x5	4258224	97.658%	99.98 %	72.57 %
3D Rotations	8-bit	2129920	97.692%	98.847%	1.514%
	32-bit	2228224	94.118%	95.589%	5.79 %
Rotations	8-bit	599040	98.101%	98.881%	75.362%
	32-bit	626688	94.434%	96.698%	5.792%

10.1145/216585.216588
[26] Zhao Zhang, Zhichun Zhu, and Xiaodong Zhang. 2000. A permutation-based page interleaving scheme to reduce row-buffer conflicts and exploit data locality. In *Proceedings of the 33rd Annual ACM/IEEE International Symposium on Microarchitecture* (Monterey, California, USA) *(MICRO 33)*. Association for Computing Machinery, New York, NY, USA, 32–41. https://doi.org/10.1145/360128.360134

A PROOF OF THEOREM 1

In this section, we prove Theorem 1 by reducing the minimum k-cut problem to Problem 3. Let us first define the minimum k-cut problem.

Problem 4 (Minimum k-Cut)**.**
Instance A graph G with edge-weights $w\colon E(G) \to \mathbb{Z}_{\geq 0}$ and an integer $k \geq 2$.
Question Find a partition of $V(G)$ into k disjoint sets $(V_1, \dots, V_k)$ such that $\sum_{e \in F} w(e)$ is minimized, where F is the edges of G whose ends are in different parts of the partition.

This problem is NP-hard [23].

Proof of Theorem 1. We show NP-completeness by a reduction from minimum k-cut. To this end, let G be a graph with edge-weights $w\colon E(G) \to \mathbb{Z}_{\geq 0}$ and $k \geq 2$. Let $V = \{v_1, \dots, v_n\}$ and define an instance of Problem 3 as follows:

We choose $c = n-k$, that is, we are looking for $(n-k)$-dimensional subspaces of $GF(2)^n$. To each edge $e \in E(G)$ we assign the vector $d(e) \in GF(2)^n$ which corresponds to e's column in the incidence matrix, that is, for $e = v_i v_j$, $d(e)$ has a 1-entry exactly at the positions i and j. The objective $w_{\mathcal{D}}$ is determined by

$$w_{\mathcal{D}}(d) = \sum_{e \in E(G)\colon d(e)=d} w(e).$$

This transformation only requires polynomial time. We note that $\sum_{e \in E(G)} w(e) = \sum_{d \in \mathcal{D}'} w_{\mathcal{D}}(d)$ and write W for this value.

We now show how to transform a solution $(V_1, \dots, V_k)$ of minimum k-cut of weight w into a subspace $K \in \mathcal{K}_c$ of weight $W - w$ and vice versa (in polynomial time). To this end, let $(V_1, \dots, V_k)$ be a solution of minimum k-cut of weight w and let F be the edges whose ends are in different parts of the partition. Thus, $\sum_{e \in F} w(e) = w$. Let $E' = E(G) \setminus F$ and consider the subspace $K = \operatorname{span}(\{d(e)\colon e \in E'\})$, for which we can provide a basis in polynomial time by removing linearly dependent vectors $d(e)$.

The dimension $\dim(K)$ is $n-k = c$ since it is equal to the rank of the matrix D, whose columns are the vectors in $\{d(e)\colon e \in E(G) \setminus F\}$. But this is the incidence matrix (over $GF(2)$) of the graph $G - E'$, and it has rank $n - k = c$, by [4, Problem 7.3]. Hence, $K \in \mathcal{K}_c$ and K is feasible.

For the objective, we note that no vector in $d(e)$ for $e \in F$ is part of K, since these cannot be written as a linear combination of the vectors $d(e)$ for $e \in E'$. (Any such linear combination has an even number of 1-entries corresponding to vertices of a component of $G - F$.) As a result, $\sum_{d \in \mathcal{D}'\colon d \in K} w_{\mathcal{D}}(d) = \sum_{d(e)\colon e \in E'} w(d) = W - w$.

Conversely, if we are given a basis of a subspace $K \in \mathcal{K}_c$ of weight $W-w$, then we can determine the set $K' = \{d \in \mathcal{D}'\colon d \in K\}$ in polynomial time. Let E' be the edges of G whose corresponding vectors are in K', that is, $E' = \{e \in E(G)\colon d(e) \in K'\}$. The rank of the incidence matrix of $G[E']$ is c since its rows are the vectors $d(e)$ for $e \in E'$. We conclude that, for $F \coloneqq E(G) \setminus E'$, $G - F = G[E']$ has k components. By choosing $V_1, \dots, V_k$ as the vertex sets of these components, we get a k-cut. Its weight is

$$\sum_{e \in F} w(e) = W - \sum_{e \in E'} w(e) = W - \sum_{d \in K'} w_{\mathcal{D}}(d) = w.$$

Thus, we get a polynomial time reduction from the decision version minimum k-cut to the decision version of Problem 3, proving NP-hardness. □

We note that, had we tried to minimize the number of row misses, then our reduction above would have yielded a function f mapping an instance I of minimum k-cut to an instance $f(I)$ of Problem 3 with the property that their optimal solutions coincide. Additionally, we obtained a function g that maps a solution K for I' to a solution $(V_1, \dots, V_k)$ for I and their objectives coincide as well. Thus, since both f and g are computable in polynomial time, they meet the requirements of an L-reduction, see [19]. Hence, an α-approximation of the row miss minimization variant of Problem 3 would yield and α-approximation of minimum k-cut. By [15], minimum k-cut is NP-hard to approximate within a $(2-\varepsilon)$ factor (for every $\varepsilon > 0$), assuming the small set expansion hypothesis. Thus, this result carries over to our problem.

Accepted 1 August 2025