

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

---

Andrei Rotaru, Oliver Bachtler, Lukas Steiner,  
Matthias Jung, Sven O. Krumke, Norbert Wehn

MEMSYS 2025, Washington D.C.

October 7, 2025



# Outline

---

## **Part I:** *Introduction and State of the Art*

- > Setting and Motivation
- > Existing Address Mapping Approaches
- > Our Contributions

## **Part II:** *The Mathematical Model*

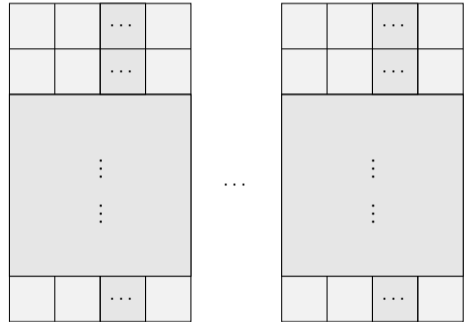
- > Problem Formulation
- > Subspace Selection View
- > Results and Extensions

## **Part I:** *Introduction and State of the Art*

# Formal Setting

---

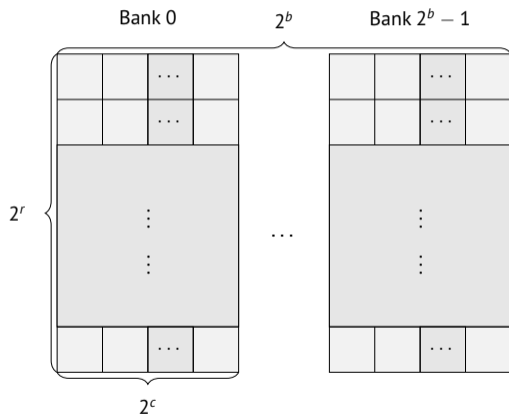
- > Assumptions:
1. Consider  $n$ -bit logical addresses  
 $x \in \{0, 1\}^n$



# Formal Setting

> Assumptions:

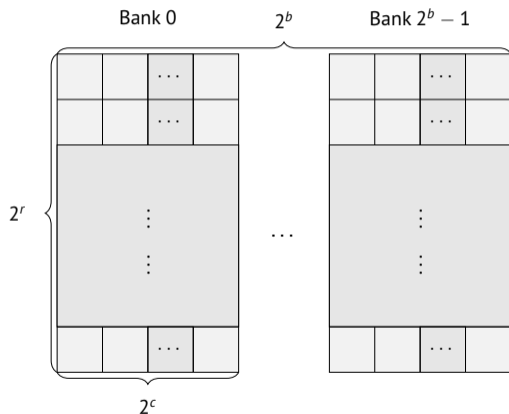
1. Consider  $n$ -bit logical addresses  $x \in \{0, 1\}^n$
2. For  $b, r, c \in \mathbb{Z}_{>0}$  and  $n = b + r + c$ , we assume a DRAM consisting of  $2^b$  banks,  $2^r$  rows and  $2^c$  columns.



# Formal Setting

> Assumptions:

1. Consider  $n$ -bit logical addresses  $x \in \{0, 1\}^n$
2. For  $b, r, c \in \mathbb{Z}_{>0}$  and  $n = b + r + c$ , we assume a DRAM consisting of  $2^b$  banks,  $2^r$  rows and  $2^c$  columns.
3. Channels and ranks are treated as part of the banks.



# Address Mappings

---

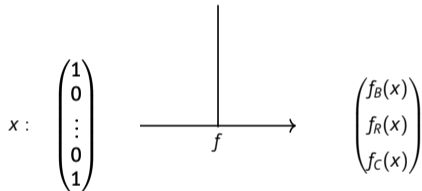
$$x : \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix}$$

$n$ -bit address vector

# Address Mappings

---

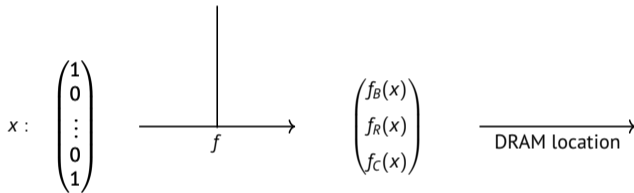
**Address Mapping:**  $f : \{0, 1\}^n \rightarrow \{0, 1\}^n$  bijective



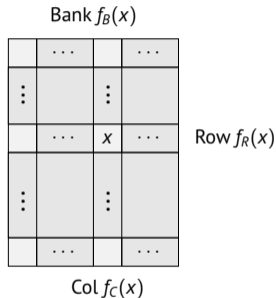
$n$ -bit address vector

# Address Mappings

**Address Mapping:**  $f : \{0, 1\}^n \rightarrow \{0, 1\}^n$  bijective



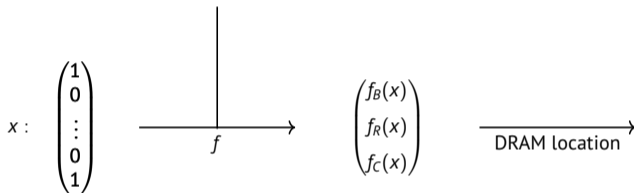
$n$ -bit address vector



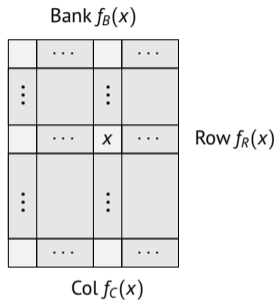
Performance strongly depends on:

# Address Mappings

**Address Mapping:**  $f : \{0, 1\}^n \rightarrow \{0, 1\}^n$  bijective



$n$ -bit address vector

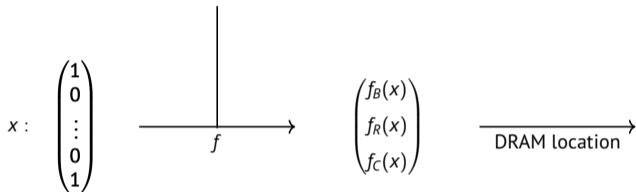


Performance strongly depends on:

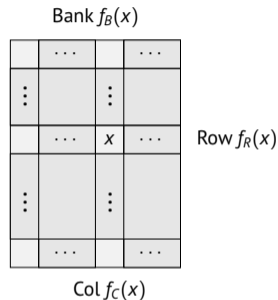
- Row misses (Page conflicts)

# Address Mappings

**Address Mapping:**  $f : \{0, 1\}^n \rightarrow \{0, 1\}^n$  bijective



$n$ -bit address vector

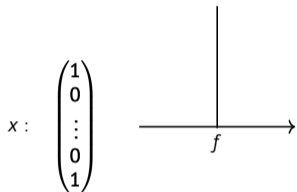


Performance strongly depends on:

- Row misses (Page conflicts)
- Degree of Bank-level parallelism exploited

# Address Mappings

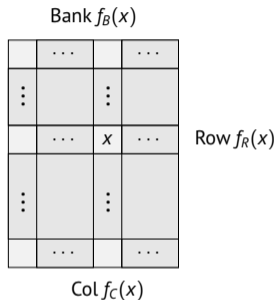
**Address Mapping:**  $f : \{0, 1\}^n \rightarrow \{0, 1\}^n$  bijective



$n$ -bit address vector

$$\begin{pmatrix} f_B(x) \\ f_R(x) \\ f_C(x) \end{pmatrix}$$

DRAM location



Performance strongly depends on:

- Row misses (Page conflicts) ← Our focus!
- Degree of Bank-level parallelism exploited

# Row Misses and Row Hits

Example:

>  $n = 3, b = 0, r = 2$  and  $c = 1$

> Address Mapping:

$$f(x) = \left( \frac{f_R(x)}{f_C(x)} \right) = \left( \frac{x_1 \oplus x_3}{x_1 \oplus x_2} \right)$$

$$\Leftrightarrow f(x) = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot x$$

Bank 0

Row 0	0	7
Row 1	2	5
Row 2	6	1
Row 3	4	3
	Col 0	Col 1

# Row Misses and Row Hits

---

Example:

>  $n = 3, b = 0, r = 2$  and  $c = 1$

> Address Mapping:

$$f(x) = \left( \begin{array}{c} f_R(x) \\ f_C(x) \end{array} \right) = \left( \begin{array}{c} x_1 \oplus x_3 \\ x_1 \oplus x_2 \\ x_3 \end{array} \right)$$

Bank 0

Row 0	0	7
Row 1	2	5
Row 2	6	1
Row 3	4	3
	Col 0	Col 1

# Row Misses and Row Hits

---

Example:

>  $n = 3, b = 0, r = 2$  and  $c = 1$

> Address Mapping:

$$f(x) = \left( \frac{f_R(x)}{f_C(x)} \right) = \left( \frac{x_1 \oplus x_3}{x_3} \right)$$

Access: **4**  $\rightarrow$  **3**

Bank 0

Row 0	0	7
Row 1	2	5
Row 2	6	1
Row 3	4	3
	Col 0	Col 1

# Row Misses and Row Hits

---

Example:

>  $n = 3, b = 0, r = 2$  and  $c = 1$

> Address Mapping:

$$f(x) = \left( \frac{f_R(x)}{f_C(x)} \right) = \left( \frac{x_1 \oplus x_3}{x_1 \oplus x_2} \right)$$

Access: 4  $\rightarrow$  **3**  $\Rightarrow$  Row Hit!

Bank 0

Row 0	0	7
Row 1	2	5
Row 2	6	1
Row 3	4	<b>3</b>
	Col 0	Col 1

# Row Misses and Row Hits

---

Example:

>  $n = 3, b = 0, r = 2$  and  $c = 1$

> Address Mapping:

$$f(x) = \left( \frac{f_R(x)}{f_C(x)} \right) = \left( \frac{x_1 \oplus x_3}{x_3} \right)$$

Access: **3**  $\rightarrow$  5

Bank 0

Row 0	0	7
Row 1	2	5
Row 2	6	1
Row 3	4	3
	Col 0	Col 1

# Row Misses and Row Hits

Example:

>  $n = 3, b = 0, r = 2$  and  $c = 1$

> Address Mapping:

$$f(x) = \left( \frac{f_R(x)}{f_C(x)} \right) = \left( \frac{x_1 \oplus x_3}{x_1 \oplus x_2} \right)$$

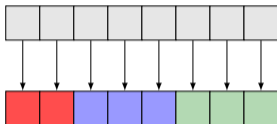
Access: 3  $\rightarrow$  5  $\Rightarrow$  Row Miss!

Bank 0

Row 0	0	7
Row 1	2	5
Row 2	6	1
Row 3	4	3
	Col 0	Col 1

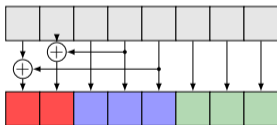
# Common Address Mapping Schemes

Linear Address Mapping (BRC)



1	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0
0	0	1	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	1

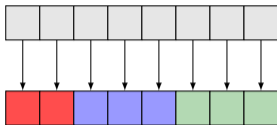
Permutation-Based Page Interleaving



1	0	0	0	1	0	0	0
0	1	0	1	0	0	0	0
0	0	1	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	1

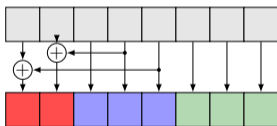
# Common Address Mapping Schemes

Linear Address Mapping (BRC)



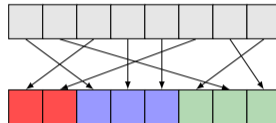
1	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0
0	0	1	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	1

Permutation-Based Page Interleaving



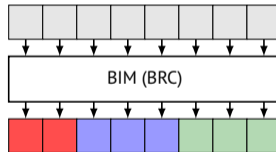
1	0	0	0	1	0	0	0
0	1	0	1	0	0	0	0
0	0	1	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	1

Bit Permutation-Based Address Mapping



0	0	1	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	1	0	0	0
0	0	0	0	0	1	0	0
0	1	0	0	0	0	0	0
0	0	0	0	0	0	1	0

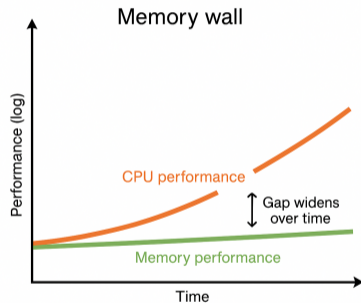
BIM-Based Address Mapping



1	0	0	0	1	0	0	0
0	1	0	1	0	0	0	0
0	0	1	0	0	0	0	1
0	0	0	1	0	0	1	0
0	0	0	0	1	1	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	1

# Application Specific Approaches

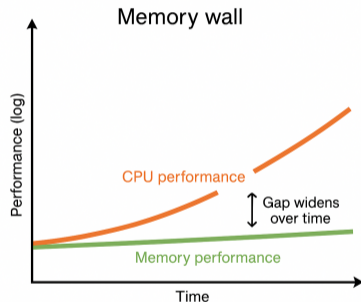
- > Exploit knowledge of application's memory access behaviour to increase the **throughput**, reduce **energy consumption** and lower **access latency**.



# Application Specific Approaches

---

- > Exploit knowledge of application's memory access behaviour to increase the **throughput**, reduce **energy consumption** and lower **access latency**.
- > Here: consider offline methods



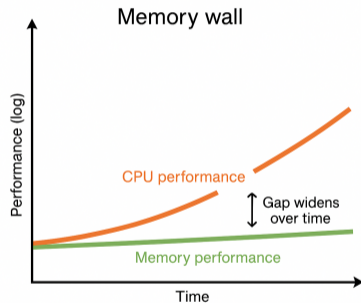
# Application Specific Approaches

- > Exploit knowledge of application's memory access behaviour to increase the **throughput**, reduce **energy consumption** and lower **access latency**.

- > Here: consider offline methods

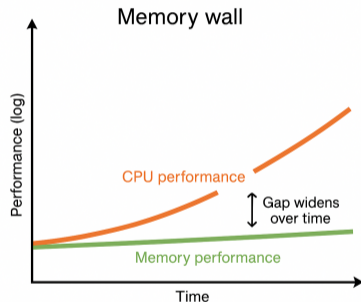
## 1. Input: Access Trace

$$\mathcal{X} = \{x_0, \dots, x_k\} \subseteq \{0, 1\}^n$$



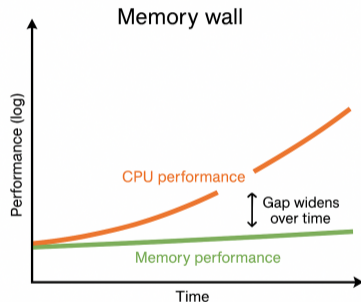
# Application Specific Approaches

- > Exploit knowledge of application's memory access behaviour to increase the **throughput**, reduce **energy consumption** and lower **access latency**.
- > Here: consider offline methods
  1. **Input:** Access Trace  
 $\mathcal{X} = \{x_0, \dots, x_k\} \subseteq \{0, 1\}^n$
  2. **Output:** Custom Address mapping  $f$



# Application Specific Approaches

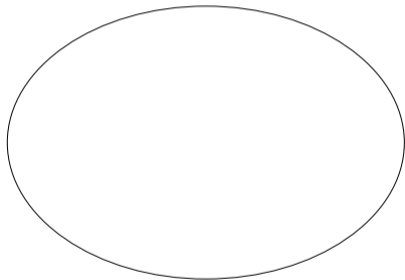
- > Exploit knowledge of application's memory access behaviour to increase the **throughput**, reduce **energy consumption** and lower **access latency**.
- > Here: consider offline methods
  1. **Input:** Access Trace  
 $\mathcal{X} = \{x_0, \dots, x_k\} \subseteq \{0, 1\}^n$
  2. **Output:** Custom Address mapping  $f$
- > Heuristic Approaches: Use reduced bit flip information (e.g. bit flip probabilities, rates or counts)



# Mathematically Optimal Address Mappings

---

$F$  :  studied  unstudied

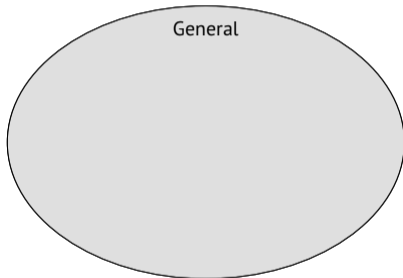


## Problem (MRH)

max # Row Hits induced by  $f$  for  $\mathcal{X}$   
s.t.  $f \in F \subseteq \{f : \{0, 1\}^n \rightarrow \{0, 1\}^n \text{ bijective}\}$

# Mathematically Optimal Address Mappings

$F$  :  studied  unstudied



## Problem (MRH)

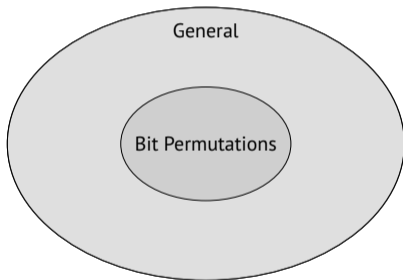
max # Row Hits induced by  $f$  for  $\mathcal{X}$   
s.t.  $f \in F \subseteq \{f : \{0, 1\}^n \rightarrow \{0, 1\}^n \text{ bijective}\}$

Studied in [Jung et al. \[2016\]](#) → Cut problems on the access graph derived from  $\mathcal{X}$

> **Problem:** solutions are usually hardware-inefficient!

# Mathematically Optimal Address Mappings

$F$  :     studied     unstudied



## Problem (MRH)

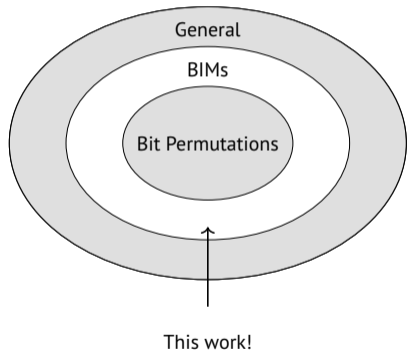
max # Row Hits induced by  $f$  for  $\mathcal{X}$   
s.t.  $f \in F \subseteq \{f : \{0, 1\}^n \rightarrow \{0, 1\}^n \text{ bijective}\}$

Studied in [Natale et al. \[2021\]](#)  $\rightarrow$  Min  $k$ -Union on hypergraph derived from  $\mathcal{X}$

- > **Good:** hardware-efficient (via multiplexers)
- > **Problem:** unsuitable for handling concurrent accesses to memory

# Mathematically Optimal Address Mappings

$F$  :     studied     unstudied



- >  $|H(A)|$ : # Row hits induced by  $A$  for  $\mathcal{X}$
- >  $\|A\|_0$ : # 1-entries in  $A$

## Problem (MCBIMP)

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

# Contributions

---

- > Extend the model from [Natale et al. \[2021\]](#): from Bit Permutations to Binary Integer Matrices (BIM).

# Contributions

---

- > Extend the model from [Natale et al. \[2021\]](#): from Bit Permutations to Binary Integer Matrices (BIM).
- > Reduce the problem of finding a row-miss minimizing BIM to a subspace selection problem and provide an ILP-formulation (exact) and a Greedy-algorithm (heuristic) to solve it.

# Contributions

---

- > Extend the model from [Natale et al. \[2021\]](#): from Bit Permutations to Binary Integer Matrices (BIM).
- > Reduce the problem of finding a row-miss minimizing BIM to a subspace selection problem and provide an ILP-formulation (exact) and a Greedy-algorithm (heuristic) to solve it.
- > Enable interesting insights into the problem of finding a row-miss minimizing BIM-based address mapping:

# Contributions

---

- > Extend the model from [Natale et al. \[2021\]](#): from Bit Permutations to Binary Integer Matrices (BIM).
- > Reduce the problem of finding a row-miss minimizing BIM to a subspace selection problem and provide an ILP-formulation (exact) and a Greedy-algorithm (heuristic) to solve it.
- > Enable interesting insights into the problem of finding a row-miss minimizing BIM-based address mapping:
  1. Lower and upper bounds for the number of row hits induced by a BIM.

# Contributions

---

- > Extend the model from [Natale et al. \[2021\]](#): from Bit Permutations to Binary Integer Matrices (BIM).
- > Reduce the problem of finding a row-miss minimizing BIM to a subspace selection problem and provide an ILP-formulation (exact) and a Greedy-algorithm (heuristic) to solve it.
- > Enable interesting insights into the problem of finding a row-miss minimizing BIM-based address mapping:
  1. Lower and upper bounds for the number of row hits induced by a BIM.
  2. Possibility to model probabilistic access traces.

# Contributions

---

- > Extend the model from [Natale et al. \[2021\]](#): from Bit Permutations to Binary Integer Matrices (BIM).
- > Reduce the problem of finding a row-miss minimizing BIM to a subspace selection problem and provide an ILP-formulation (exact) and a Greedy-algorithm (heuristic) to solve it.
- > Enable interesting insights into the problem of finding a row-miss minimizing BIM-based address mapping:
  1. Lower and upper bounds for the number of row hits induced by a BIM.
  2. Possibility to model probabilistic access traces.
  3. Show that BIMs handle concurrent memory accesses more effectively than bit permutations.

## **Part II:** *The Mathematical Model*

# Assumptions and Notation

---

> Given: access trace  $\mathcal{X} = \{x_0, \dots, x_k\} \subseteq \{0,1\}^n$

# Assumptions and Notation

---

- > Given: access trace  $\mathcal{X} = \{x_0, \dots, x_k\} \subseteq \{0,1\}^n$
- > Find:  $A \in GF(2)^n$  optimal  $\rightarrow A_B \in GF(2)^{b \times n}, A_R \in GF(2)^{r \times n}, A_C \in GF(2)^{c \times n}$ .

# Assumptions and Notation

---

- > Given: access trace  $\mathcal{X} = \{x_0, \dots, x_k\} \subseteq \{0,1\}^n$
- > Find:  $A \in GF(2)^n$  optimal  $\rightarrow A_B \in GF(2)^{b \times n}, A_R \in GF(2)^{r \times n}, A_C \in GF(2)^{c \times n}$ .
- > Focus on  $b = 0$  to facilitate subsequent discussion.

# Assumptions and Notation

---

- > Given: access trace  $\mathcal{X} = \{x_0, \dots, x_k\} \subseteq \{0,1\}^n$
- > Find:  $A \in GF(2)^n$  optimal  $\rightarrow A_B \in GF(2)^{b \times n}, A_R \in GF(2)^{r \times n}, A_C \in GF(2)^{c \times n}$ .
- > Focus on  $b = 0$  to facilitate subsequent discussion.
- > The multiple bank case can be reduced to a single bank problem.

# Assumptions and Notation

---

- > Given: access trace  $\mathcal{X} = \{x_0, \dots, x_k\} \subseteq \{0,1\}^n$
- > Find:  $A \in GF(2)^n$  optimal  $\rightarrow A_B \in GF(2)^{b \times n}, A_R \in GF(2)^{r \times n}, A_C \in GF(2)^{c \times n}$ .
- > Focus on  $b = 0$  to facilitate subsequent discussion.
- > The multiple bank case can be reduced to a single bank problem.

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

# Characterizing Row Hits by Difference Vectors

---

> Row hit at  $i$  if and only if

$$A_R \cdot x_i = A_R \cdot x_{i-1} \quad (1)$$

$$\Leftrightarrow A_R \cdot (x_i \oplus x_{i-1}) = 0 \quad (2)$$

$$\Leftrightarrow d_i := x_i \oplus x_{i-1} \in \text{Ker}(A_R) \quad (3)$$

# Characterizing Row Hits by Difference Vectors

---

> Row hit at  $i$  if and only if

$$A_R \cdot x_i = A_R \cdot x_{i-1} \quad (1)$$

$$\Leftrightarrow A_R \cdot (x_i \oplus x_{i-1}) = 0 \quad (2)$$

$$\Leftrightarrow d_i := x_i \oplus x_{i-1} \in \text{Ker}(A_R) \quad (3)$$

> Define set of **difference vectors**

$$\mathcal{D} := \{d_i := x_{i-1} \oplus x_i \mid i \in \{1, \dots, k\}\}.$$

# Characterizing Row Hits by Difference Vectors

---

- > Row hit at  $i$  if and only if

$$A_R \cdot x_i = A_R \cdot x_{i-1} \quad (1)$$

$$\Leftrightarrow A_R \cdot (x_i \oplus x_{i-1}) = 0 \quad (2)$$

$$\Leftrightarrow d_i := x_i \oplus x_{i-1} \in \text{Ker}(A_R) \quad (3)$$

- > Define set of **difference vectors**

$$\mathcal{D} := \{d_i := x_{i-1} \oplus x_i \mid i \in \{1, \dots, k\}\}.$$

- > Compress into weight function:

$$w_{\mathcal{D}} : GF(2)^n \rightarrow \mathbb{Z}_{\geq 0},$$

$$d \mapsto |\{i \in \{1, \dots, k\} : d_i = d\}|$$

# Characterizing Row Hits by Difference Vectors

- > Row hit at  $i$  if and only if

$$A_R \cdot x_i = A_R \cdot x_{i-1} \quad (1)$$

$$\Leftrightarrow A_R \cdot (x_i \oplus x_{i-1}) = 0 \quad (2)$$

$$\Leftrightarrow d_i := x_i \oplus x_{i-1} \in \text{Ker}(A_R) \quad (3)$$

- > Define set of **difference vectors**  
 $\mathcal{D} := \{d_i := x_{i-1} \oplus x_i \mid i \in \{1, \dots, k\}\}$ .
- > Compress into weight function:

$$w_{\mathcal{D}} : GF(2)^n \rightarrow \mathbb{Z}_{\geq 0},$$

$$d \mapsto |\{i \in \{1, \dots, k\} : d_i = d\}|$$

## Observation

$$\begin{aligned} |H(A)| &= |\{i \in \{1, \dots, k\} : d_i \in \text{Ker}(A_R)\}| \\ &= \sum_{d \in \text{Ker}(A_R)} w_{\mathcal{D}}(d) \end{aligned}$$

# Characterizing Row Hits by Difference Vectors

> Row hit at  $i$  if and only if

$$A_R \cdot x_i = A_R \cdot x_{i-1} \quad (1)$$

$$\Leftrightarrow A_R \cdot (x_i \oplus x_{i-1}) = 0 \quad (2)$$

$$\Leftrightarrow d_i := x_i \oplus x_{i-1} \in \text{Ker}(A_R) \quad (3)$$

> Define set of **difference vectors**

$$\mathcal{D} := \{d_i := x_{i-1} \oplus x_i \mid i \in \{1, \dots, k\}\}.$$

> Compress into weight function:

$$w_{\mathcal{D}} : GF(2)^n \rightarrow \mathbb{Z}_{\geq 0},$$

$$d \mapsto |\{i \in \{1, \dots, k\} : d_i = d\}|$$

## Observation

$$\begin{aligned} |H(A)| &= |\{i \in \{1, \dots, k\} : d_i \in \text{Ker}(A_R)\}| \\ &= \sum_{d \in \text{Ker}(A_R)} w_{\mathcal{D}}(d) \end{aligned}$$

$\Rightarrow$  For  $A, A'$  address mappings:

$$\text{Ker}(A_R) = \text{Ker}(A'_R) \implies |H(A)| = |H(A')|$$

# Restricting The Search Space

---

$\text{Ker}(A_R)$  is a subspace of  $GF(2)^n$  of dimension

$$\dim(\text{Ker}(A_R)) = n - \text{rank}(A_R) = n - r = c$$

# Restricting The Search Space

---

$\text{Ker}(A_R)$  is a subspace of  $GF(2)^n$  of dimension

$$\dim(\text{Ker}(A_R)) = n - \text{rank}(A_R) = n - r = c$$

## Reverse-Engineering the optimal Nullspace

1. Choose the best  $c$ -dimensional subspace  $X$  w.r.t.  $\mathcal{D}$

# Restricting The Search Space

---

$\text{Ker}(A_R)$  is a subspace of  $GF(2)^n$  of dimension

$$\dim(\text{Ker}(A_R)) = n - \text{rank}(A_R) = n - r = c$$

## Reverse-Engineering the optimal Nullspace

1. Choose the best  $c$ -dimensional subspace  $X$  w.r.t.  $\mathcal{D}$
2. Determine sparsest matrix  $C \in GF(2)^{r \times n}$  s.t.  $\text{Ker}(C) = X$

# Restricting The Search Space

---

$\text{Ker}(A_R)$  is a subspace of  $GF(2)^n$  of dimension

$$\dim(\text{Ker}(A_R)) = n - \text{rank}(A_R) = n - r = c$$

## Reverse-Engineering the optimal Nullspace

1. Choose the best  $c$ -dimensional subspace  $X$  w.r.t.  $\mathcal{D}$
2. Determine sparsest matrix  $C \in GF(2)^{r \times n}$  s.t.  $\text{Ker}(C) = X$
3. Set  $A_R := C$  and obtain  $A_C$  by completing the rows of  $A_R$  to a basis of  $GF(2)^n$

# Subspace Selection Problem

---

$$\mathcal{D}' := \{d \in GF(2)^n : w_{\mathcal{D}}(d) > 0\}.$$

## Problem (SSP)

$$\begin{aligned} \max \quad & \sum_{d \in \mathcal{D}' : d \in K} w_{\mathcal{D}}(d) \quad =: |H(K)| \\ \text{s.t.} \quad & K \in \mathcal{K}_c := \{X \leq GF(2)^n \mid \dim(X) = c\} \end{aligned}$$

# Subspace Selection Problem

$$\mathcal{D}' := \{d \in GF(2)^n : w_{\mathcal{D}}(d) > 0\}.$$

## Problem (SSP)

$$\begin{aligned} \max \quad & \sum_{d \in \mathcal{D}' : d \in K} w_{\mathcal{D}}(d) \quad =: |H(K)| \\ \text{s.t.} \quad & K \in \mathcal{K}_c := \{X \leq GF(2)^n \mid \dim(X) = c\} \end{aligned}$$

## Theorem

*Problem SSP is NP-hard.*

# Approaches to Solving the SSP

---

- > **Smart Enumeration:** Tackle the problem by an exact ILP-based branch and bound approach.

# Approaches to Solving the SSP

---

- > **Smart Enumeration:** Tackle the problem by an exact ILP-based branch and bound approach.
- > Disadvantage of the ILP-based approach: Only small instances are solvable with it at the moment.

# Approaches to Solving the SSP

---

- > **Smart Enumeration:** Tackle the problem by an exact ILP-based branch and bound approach.
- > Disadvantage of the ILP-based approach: Only small instances are solvable with it at the moment.
- > Idea: Try Greedy-based approach for bigger traces!

## Greedy Algorithm

```
1:  $K \leftarrow \{0\}, V \leftarrow GF(2)^n$ 
2: for  $i = 1, \dots, c$  do
3:   Let  $a_i \in$   
    $\arg \max \{ |H(\text{span}(K \cup \{a\}))| : a \in V \}$ 
4:    $K \leftarrow \text{span}(K \cup \{a_i\}), V \leftarrow V \setminus K$ 
5: end for
6: return  $K$ 
```

# Approaches to Solving the SSP

---

- > **Smart Enumeration:** Tackle the problem by an exact ILP-based branch and bound approach.
- > Disadvantage of the ILP-based approach: Only small instances are solvable with it at the moment.
- > Idea: Try Greedy-based approach for bigger traces!
- > This already yields surprisingly good results for real-world traces (later).

## Greedy Algorithm

```
1:  $K \leftarrow \{0\}, V \leftarrow GF(2)^n$ 
2: for  $i = 1, \dots, c$  do
3:   Let  $a_i \in$   
    $\arg \max \{ |H(\text{span}(K \cup \{a\}))| : a \in V \}$ 
4:    $K \leftarrow \text{span}(K \cup \{a_i\}), V \leftarrow V \setminus K$ 
5: end for
6: return  $K$ 
```

Combine greedy heuristic in combination with sparsity optimization!

# Lower and Upper Bounds

---

- > We can determine a **lower** and an **upper** bound on the number of row hits achievable by an optimal address mapping based on a BIM.

# Lower and Upper Bounds

---

- > We can determine a **lower** and an **upper** bound on the number of row hits achievable by an optimal address mapping based on a BIM.
- > We assume that  $w_{\mathcal{D}}(d_1) \geq \dots \geq w_{\mathcal{D}}(d_l)$  and let  $l^* \in \{1, \dots, l\}$  be the maximal index that satisfies  $K^* := \text{span}(\{d_i : i \leq l^*\}) \in \mathcal{K}_c$ .

# Lower and Upper Bounds

---

- > We can determine a **lower** and an **upper** bound on the number of row hits achievable by an optimal address mapping based on a BIM.
- > We assume that  $w_{\mathcal{D}}(d_1) \geq \dots \geq w_{\mathcal{D}}(d_l)$  and let  $l^* \in \{1, \dots, l\}$  be the maximal index that satisfies  $K^* := \text{span}(\{d_i : i \leq l^*\}) \in \mathcal{K}_c$ .
- > Lower Bound:  $\underline{z} := |H(K^*)|$

# Lower and Upper Bounds

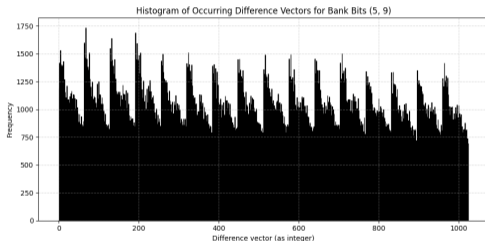
---

- > We can determine a **lower** and an **upper** bound on the number of row hits achievable by an optimal address mapping based on a BIM.
- > We assume that  $w_{\mathcal{D}}(d_1) \geq \dots \geq w_{\mathcal{D}}(d_l)$  and let  $l^* \in \{1, \dots, l\}$  be the maximal index that satisfies  $K^* := \text{span}(\{d_i : i \leq l^*\}) \in \mathcal{K}_c$ .
- > Lower Bound:  $\underline{z} := |H(K^*)|$
- > Every  $K \in \mathcal{K}_c$  contains  $2^c$  elements.

# Lower and Upper Bounds

- > We can determine a **lower** and an **upper** bound on the number of row hits achievable by an optimal address mapping based on a BIM.
- > We assume that  $w_{\mathcal{D}}(d_1) \geq \dots \geq w_{\mathcal{D}}(d_l)$  and let  $l^* \in \{1, \dots, l\}$  be the maximal index that satisfies  $K^* := \text{span}(\{d_i : i \leq l^*\}) \in \mathcal{K}_c$ .
- > Lower Bound:  $\underline{z} := |H(K^*)|$
- > Every  $K \in \mathcal{K}_c$  contains  $2^c$  elements.
- > Upper Bound:  $\bar{z} := \sum_{i=1}^{\max\{l, 2^c\}} w_{\mathcal{D}}(d_i)$

Example Sequence: 1 million 12-bit addresses

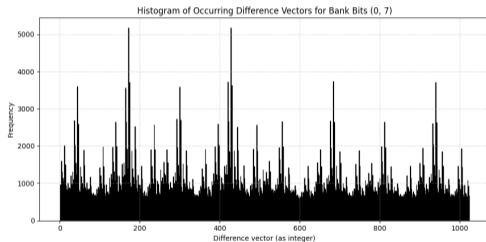


**Upper Bound: 47972 i.e.  $\sim 5\%$**

# Lower and Upper Bounds

- > We can determine a **lower** and an **upper** bound on the number of row hits achievable by an optimal address mapping based on a BIM.
- > We assume that  $w_{\mathcal{D}}(d_1) \geq \dots \geq w_{\mathcal{D}}(d_l)$  and let  $l^* \in \{1, \dots, l\}$  be the maximal index that satisfies  $K^* := \text{span}(\{d_i : i \leq l^*\}) \in \mathcal{K}_c$ .
- > Lower Bound:  $\underline{z} := |H(K^*)|$
- > Every  $K \in \mathcal{K}_c$  contains  $2^c$  elements.
- > Upper Bound:  $\bar{z} := \sum_{i=1}^{\max\{l, 2^c\}} w_{\mathcal{D}}(d_i)$

Example Sequence: 1 million 12-bit addresses



**Upper Bound: 94974 i.e.  $\sim 10\%$**

# Probabilistic Extension

---

- > Further advantage of our model: Extension to probabilistic traces.

# Probabilistic Extension

---

- > Further advantage of our model: Extension to probabilistic traces.
- > Define a probabilistic access sequence  $\mathcal{X}_{\text{prob}} := (X_0, \dots, X_k)$ , where the  $X_i$  are random variables with values in  $GF(2)^n$ .

# Probabilistic Extension

---

- > Further advantage of our model: Extension to probabilistic traces.
- > Define a probabilistic access sequence  $\mathcal{X}_{\text{prob}} := (X_0, \dots, X_k)$ , where the  $X_i$  are random variables with values in  $GF(2)^n$ .
- > We get a random sequence of difference vectors  $\mathcal{D}_{\text{prob}} = (D_1, \dots, D_k) = (X_1 \oplus X_0, \dots, X_k \oplus X_{k-1})$  as well as random weights  $w_{\mathcal{D}_{\text{prob}}}(d)$  for  $d \in GF(2)^n$

# Probabilistic Extension

---

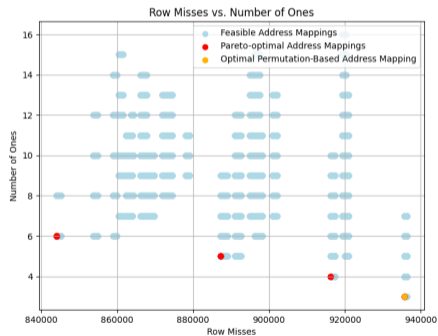
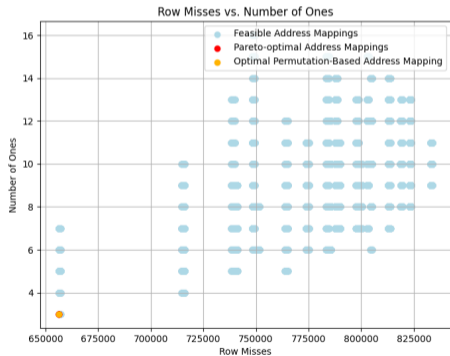
- > Further advantage of our model: Extension to probabilistic traces.
- > Define a probabilistic access sequence  $\mathcal{X}_{\text{prob}} := (X_0, \dots, X_k)$ , where the  $X_i$  are random variables with values in  $GF(2)^n$ .
- > We get a random sequence of difference vectors  $\mathcal{D}_{\text{prob}} = (D_1, \dots, D_k) = (X_1 \oplus X_0, \dots, X_k \oplus X_{k-1})$  as well as random weights  $w_{\mathcal{D}_{\text{prob}}}(d)$  for  $d \in GF(2)^n$
- > Aim: maximize the expected number of row hits for the given trace.

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

# Bit Permutations vs. BIMs

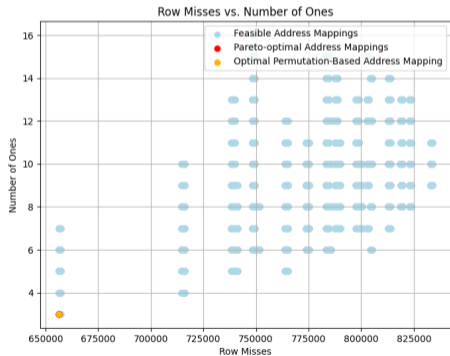
Let  $n = 6, r = c = 3$  and generate two sequences from random walks (same bit flip probability in every bit):

1.  $\mathcal{X}_l$ : Bit flip rate 0.3 (low)
2.  $\mathcal{X}_h$ : Bit flip rate 0.6 (high)



# Bit Permutations vs. BIMs

Let  $n = 6, r = c = 3$  and generate two sequences from random walks (same bit flip probability in every bit):



Permutation matrices make up only around 1.4% of all feasible solutions!



# Application to Interleaved Traces

- > Evaluate address mapping using synthetic workload: Simulate shared-memory systems where multiple initiators issue interleaved memory requests with varying stride lengths
- > Assumption: 24-bit addresses and 12 row and 12 column bits targeting the same DRAM bank.
- > For  $k \in \mathbb{Z}_{>0}$  initiators, the access stride of initiator  $i$  is defined as  $2^{i \cdot \frac{n}{k}}$ .

Category	Sequence	Sequence Length	Row Hit Rate [%]		
			Greedy[our]	ConGen2 [Natale et al, 2021]	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%

# Acknowledgments

---



Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - GRK 2982, 516090167  
“Mathematics of Interdisciplinary Multiobjective Optimization”.

## References

---

- Matthias Jung, Deepak M. Mathew, Christian Weis, Norbert Wehn, Irene Heinrich, Marco V. Natale, and Sven O. Krumke. Congen: An application specific dram memory controller generator. In *Proceedings of the Second International Symposium on Memory Systems, MEMSYS '16*, page 257–267, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450343053. doi: 10.1145/2989081.2989131. URL <https://doi.org/10.1145/2989081.2989131>.
- Marco V. Natale, Matthias Jung, Kira Kraft, Frederik Lauer, Johannes Feldmann, Chirag Sudarshan, Christian Weis, Sven Krumke, and Norbert Wehn. Efficient generation of application specific memory controllers. In *Proceedings of the International Symposium on Memory Systems, MEMSYS '20*, page 233–247, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450388993. doi: 10.1145/3422575.3422796. URL <https://doi.org/10.1145/3422575.3422796>.