GPU-Accelerated Search for Fast Matrix Multiplication over $\mathbb{F}_2$

Anonymous submission

Abstract. We present a GPU-accelerated algorithm for searching for ways to multiply matrices with few scalar multiplications. Our algorithm searches the matrix multiplication flip graph and, on an NVIDIA H200, performs over a billion search steps per second, a 1000× improvement over previous GPU-accelerated search on the tensor for 7×7 matrix multiplication. We also prove that, over $\mathbb{F}_2$, the directed matrix multiplication flip graph is strongly connected with only flip and plus edges. This removes the need for computationally expensive reduction edges in the connectivity argument used in previous work. Using this GPU-accelerated search procedure on our simplified flip graph, we find a way to multiply 7×7 matrices over $\mathbb{F}_2$ using 245 multiplications, a three-multiplication improvement over the previous record.

1 Introduction The vertices of the matrix multiplication flip graph are ways to multiply matrices. Some ways require fewer multiplications and can outperform the naive algorithm at practical matrix sizes [9], so considerable effort has been spent exploring the flip graph to find them. Indeed, the current records for multiplying 5×5, 6×6, and 7×7 matrices over $\mathbb{F}_2$ were found via flip graph search [6, 7].

In a sense that we make precise below, a way to multiply matrices can be represented as a multiset of triples. The central operation of flip graph search is finding sets of triples with the same value in one position. This is a matching problem CPU-oriented implementations naturally solve with mappings between values and positions [6].

The guiding design principle of this work is to exploit the massive parallelism available in modern GPUs for flip graph search. Thus, rather than maintaining and querying large maps, we use hardware-supported matching primitives to identify sets of triples with the same value in a position, replacing hash-table lookups with hardware.

Our contributions are as follows.

1. We design a GPU-friendly flip-graph search algorithm that uses warp-level matching primitives to detect flip opportunities. This outperforms previous GPU-accelerated search [7] by a factor of over 1000 while searching the 7×7 flip graph and executes over a billion search steps per second on an NVIDIA H200 GPU.
2. We introduce generalized multi-summand flips, which exploit linear dependencies among summands sharing a tensor factor and expose rank reductions efficiently on GPUs. On average, generalized flips discover the record 4×4 scheme 1.66× as fast, versus an implementation without them.
3. We prove that, over $\mathbb{F}_2$, ordinary flip and plus edges alone strongly connect the directed matrix multiplication flip graph, removing the need for a third, computationally expensive reduction edge from prior work [1].
4. These culminate in a way to multiply 7×7 matrices over $\mathbb{F}_2$ with 245 multiplications, a three-multiplication improvement over the prior record [7].

Section 3 defines the flip graph and related notation. Section 4 contains the connectivity proof for flip and plus edges over $\mathbb{F}_2$. Section 5 describes our GPU-accelerated search procedure, including our use of GPU matching primitives and generalized flips. Section 6 compares our implementation to Perminov's GPU-based search algorithm [7] and describes our ablation testing for generalized flips.

2 Related Work The matrix multiplication flip graph was introduced by Kauers and Moosbauer [3], improving on a record for 5×5 matrix multiplication set by AlphaTensor [2]. Kauers and Moosbauer's construction connected schemes if they were related by a flip or reduction transform, and it was shown that with these edges the flip graph is weakly connected: by treating reduction transforms as bidirectional, every scheme is reachable from every other scheme.

Arai, Ichikawa, and Hukushima [1] improved the connectivity situation by introducing a third edge, plus transitions. Empirically, these help search escape local minima, and theoretically their addition was shown to make the flip graph strongly connected.[1] This work

[1] Arai et al. state their connectivity theorem without the qualifier *strongly*; however, their flip graph is defined as a directed

resulted in another improvement of the record for 5×5 matrix multiplication.

Moosbauer and Poole introduced flip graphs with symmetry [6]. A detailed account of these symmetries is outside the scope of this paper, but the practical effect is to reduce the size of the explorable flip graph at the expense of completeness, as non-symmetric schemes become unreachable. This work resulted in further improvements on the record 5×5 and 6×6 matrix multiplication schemes.

Subsequently, Wood adapted flip graph search to the commutative setting [8]; Khoruzhii, Gelß, and Pokutta applied flip graph search to structured matrix multiplication, for example multiplication resulting in a symmetric matrix [5]; Kauers and Wood [4] introduced the meta flip graph, adding extension and projection moves between different matrix formats; and, notably, Perminov gave a GPU-accelerated flip-graph implementation [7].

3 Preliminaries

Throughout, we use $\uplus$ for a multiset union, $\cup$ for the standard set union, and $\setminus$ for both multiset and set difference. Matrices and tensors are over a fixed base field; from section 4 onward we specialize to $\mathbb{F}_2$.

3.1 Matrix Multiplication Tensors

Definition 3.1 (Matrix multiplication tensor). *The matrix multiplication tensor for multiplying two $n \times n$ matrices[2] is the unique tensor $\mathcal{M}^{(n)}$ satisfying, for all $n \times n$ matrices D and F,*

$$(DF)_{k,\ell} = \sum_{g,h,i,j=1}^{n} \mathcal{M}^{(n)}_{g,h,i,j,k,\ell} D_{g,h} F_{i,j}. \tag{3.1}$$

Equivalently, $\mathcal{M}^{(n)}$ records which products of entries of D and F contribute to each entry of DF.

A rank-R decomposition of $\mathcal{M}^{(n)}$ is an expression

$$\mathcal{M}^{(n)} = \sum_{r=1}^{R} A^{(r)} \otimes B^{(r)} \otimes C^{(r)}, \tag{3.2}$$

where each $A^{(r)}, B^{(r)}, C^{(r)}$ is an $n \times n$ matrix and

$$\big(A \otimes B \otimes C\big)_{g,h,i,j,k,\ell} = A_{g,h} B_{i,j} C_{k,\ell}.$$

Such a decomposition gives an algorithm for multiplying D and F using R multiplications involving entries of the input matrices. Indeed, substituting (3.2) into (3.1) gives

$$\begin{aligned}(DF)_{k,\ell} &= \sum_{g,h,i,j} \left(\sum_{r=1}^{R} A^{(r)} \otimes B^{(r)} \otimes C^{(r)} \right)_{g,h,i,j,k,\ell} D_{g,h} F_{i,j} \\ &= \sum_{r=1}^{R} C^{(r)}_{k,\ell} \left(\sum_{g,h} A^{(r)}_{g,h} D_{g,h} \right) \left(\sum_{i,j} B^{(r)}_{i,j} F_{i,j} \right).\end{aligned}$$

Thus the only multiplications involving entries of D and F are the R products

$$m_r = \left(\sum_{g,h} A^{(r)}_{g,h} D_{g,h} \right) \left(\sum_{i,j} B^{(r)}_{i,j} F_{i,j} \right), \qquad r = 1, \ldots, R.$$

Conversely, any way to multiply two matrices with R scalar multiplications gives a rank-R decomposition of $\mathcal{M}^{(n)}$. Our goal is thus to look for low-rank decompositions of $\mathcal{M}^{(n)}$.

3.2 The matrix multiplication flip graph

Definition 3.2 (Schemes and vertices). *The vertices of the matrix multiplication flip graph for $n \times n$ matrix multiplication are multisets*

$$S = \left\{ A^{(1)} \otimes B^{(1)} \otimes C^{(1)}, \ldots, A^{(|S|)} \otimes B^{(|S|)} \otimes C^{(|S|)} \right\}$$

of nonzero rank-one tensors satisfying

$$\sum_{s \in S} s = \mathcal{M}^{(n)}.$$

We call such a multiset a scheme, and we store each summand with a chosen factorization $(A^{(r)}, B^{(r)}, C^{(r)})$; over $\mathbb{F}_2$ the factorization of a nonzero rank-one tensor is unique. Following the flip-graph literature, we call the size $|S|$ the rank of the scheme, although it is the length of this particular decomposition and may exceed the tensor rank of $\mathcal{M}^{(n)}$.

There is a directed edge from a vertex S to a vertex S' in the flip graph when S' is obtained from S via a *flip* or *plus* transform.

Definition 3.3 (Flip and plus transformations). *Let S be a scheme. If S can be transformed into S' by one of the following operations, then there is a directed flip/plus edge from S to S'.*

Flip. *Flip on the A position applies to two summands $A \otimes B \otimes C$, $A' \otimes B' \otimes C'$ with $A = A'$ and replaces them by $A \otimes B \otimes (C + C')$, $A' \otimes (B' - B) \otimes C'$. Equivalently,*

$$\begin{aligned}S' = S \setminus \{ &A \otimes B \otimes C, A' \otimes B' \otimes C' \} \\ &\uplus \{ A \otimes B \otimes (C + C'), A' \otimes (B' - B) \otimes C' \},\end{aligned}$$

graph and their proof applies transformations only in the forward direction, which establishes strong connectivity.

[2]The exposition is for square matrix multiplication, but the definitions extend directly to rectangular matrix multiplication.

with any zero summands deleted.

Plus. *Plus on the A position takes two summands $A \otimes B \otimes C$, $A' \otimes B' \otimes C'$, and replaces the second summand by two summands (leaving the first summand unchanged): $(A' - A) \otimes B' \otimes C'$, and $A \otimes B' \otimes C'$. Equivalently,*

$$S' = S \setminus \{A' \otimes B' \otimes C'\} \uplus \{(A' - A) \otimes B' \otimes C', A \otimes B' \otimes C'\},$$

with any zero summands deleted.

These transformations on the B and C positions are defined analogously.

Both transformations preserve the sum of the rank-one tensors in a scheme: in each case the inserted summands sum to the summands they replace. Therefore, every vertex reached by such a transformation is again a decomposition of $\mathcal{M}^{(n)}$.

4 Connectivity Proof In this section, we prove that flip and plus edges are sufficient for the flip graph to be strongly connected over $\mathbb{F}_2$.

THEOREM 4.1 (Flip-plus connectivity over $\mathbb{F}_2$). *Over $\mathbb{F}_2$, for every $n \geq 2$, the directed matrix multiplication flip graph whose edges are ordinary flip and plus transformations is strongly connected. Equivalently, for any two schemes S and S' for $\mathcal{M}^{(n)}$, there is a directed path from S to S' using only flip and plus transformations.*

Our choice to consider connectivity over $\mathbb{F}_2$ stems from two observations. First, Moosbauer and Poole [6] do not explicitly search for reduction edges in their flip graph algorithm, noting that they occur rarely and slow down search. Second, several existing flip-graph works [3, 1, 6] perform their searches over the field $\mathbb{F}_2$. Because the primary use of reduction edges in the previous connectivity proof [1] is to remove zero-summing submultisets of a matrix multiplication scheme, this led us to consider whether we could exploit the properties of $\mathbb{F}_2$ for it instead. Thus, our proof relies on the following observation of $\mathbb{F}_2$.

Observation 4.2. Over $\mathbb{F}_2$, two identical summands can be deleted using a flip. If two copies of $A \otimes B \otimes C$ appear in a scheme, then a flip on the A position produces

$$A \otimes B \otimes (C + C) \qquad \text{and} \qquad A \otimes (B - B) \otimes C,$$

both of which are zero over $\mathbb{F}_2$ and can therefore be removed.

Throughout this section we assume $n \geq 2$; this is the relevant regime for matrix multiplication search, and it avoids the degenerate one-dimensional case.

LEMMA 4.3. *The matrix multiplication tensor for multiplying two $n \times n$ matrices is*

$$\mathcal{M}^{(n)} = \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} E_{ij} \otimes E_{jk} \otimes E_{ik},$$

where E_{ij} is an $n \times n$ matrix with a 1 at position (i, j) and zeros everywhere else.

Proof. By Definition 3.1, the coefficient $\mathcal{M}^{(n)}_{g,h,i,j,k,\ell}$ is 1 exactly when the product $D_{g,h}F_{i,j}$ contributes to $(DF)_{k,\ell}$. This happens precisely when $g = k$, $h = i$, and $j = \ell$. Hence

$$\mathcal{M}^{(n)}_{g,h,i,j,k,\ell} = \delta_{g,k}\delta_{h,i}\delta_{j,\ell}.$$

Therefore, inspecting the standard basis expansion of $\mathcal{M}^{(n)}$ gives

$$\begin{aligned}\mathcal{M}^{(n)} &= \sum_{i,j,j',k,i',k'} \delta_{i,i'}\delta_{k,k'}\delta_{j,j'} E_{i,j} \otimes E_{j',k} \otimes E_{i',k'} \\ &= \sum_{i,j,k} E_{i,j} \otimes E_{j,k} \otimes E_{i,k}.\end{aligned}$$

∎

LEMMA 4.4. *Let S be a decomposition of $\mathcal{M}^{(n)}$ over a field $\mathbb{F}$:*

$$S = \left\{ A^{(1)} \otimes B^{(1)} \otimes C^{(1)}, \ldots, A^{(|S|)} \otimes B^{(|S|)} \otimes C^{(|S|)} \right\}.$$

Then

$$\begin{aligned}\operatorname{span}_{\mathbb{F}}\{A^{(r)} : r \in [|S|]\} &= \operatorname{span}_{\mathbb{F}}\{B^{(r)} : r \in [|S|]\} \\ &= \operatorname{span}_{\mathbb{F}}\{C^{(r)} : r \in [|S|]\} = \mathbb{F}^{n \times n}.\end{aligned}$$

Proof. An alternative proof appears as Lemma 5.1 of Arai et al. [1].

Let $R = |S|$. We prove the claim for the A-components; the arguments for the B- and C-components are symmetric.

Write each $B^{(r)}$ and $C^{(r)}$ in the standard basis:

$$B^{(r)} = \sum_{j,k'=1}^{n} \beta^{(r)}_{j,k'} E_{j,k'}, \qquad C^{(r)} = \sum_{i,k=1}^{n} \gamma^{(r)}_{i,k} E_{i,k}.$$

By Lemma 4.3 and the assumption that S decom-

poses $\mathcal{M}^{(n)}$,

$$\begin{aligned}\sum_{i,j,k=1}^{n} E_{i,j}\otimes E_{j,k}\otimes E_{i,k} &= \sum_{r=1}^{R} A^{(r)}\otimes B^{(r)}\otimes C^{(r)}\\ &= \sum_{r=1}^{R} A^{(r)}\otimes\left(\sum_{j,k'=1}^{n}\beta_{j,k'}^{(r)}E_{j,k'}\right)\\ &\qquad\otimes\left(\sum_{i,k=1}^{n}\gamma_{i,k}^{(r)}E_{i,k}\right)\\ &= \sum_{j,k',i,k=1}^{n}\left(\sum_{r=1}^{R}A^{(r)}\beta_{j,k'}^{(r)}\gamma_{i,k}^{(r)}\right)\\ &\qquad\otimes E_{j,k'}\otimes E_{i,k}.\end{aligned}$$

Now fix i and j, and choose any $k\in[n]$. Comparing the coefficients of $E_{j,k}\otimes E_{i,k}$ in the last two tensor factors, we get

$$E_{i,j}=\sum_{r=1}^{R}A^{(r)}\beta_{j,k}^{(r)}\gamma_{i,k}^{(r)}.$$

Thus every basis matrix $E_{i,j}$ lies in $\operatorname{span}_{\mathbb{F}}\{A^{(r)} : r\in[R]\}$, so the A-components span all of $\mathbb{F}^{n\times n}$. □

LEMMA 4.5. *Let S be a scheme over $\mathbb{F}_2$, and suppose*

$$A\otimes B\otimes C,\qquad A'\otimes B'\otimes C'$$

are summands of S. Let $X=A+A'$. Then there is a path from S to

$$S\uplus\{X\otimes B\otimes C,\ X\otimes B\otimes C\}.$$

Analogous statements hold for the B and C positions.

Proof. If $X=0$, then the two added summands are zero and are deleted, so the trivial path suffices. Hence assume $X\neq 0$. We suppress all summands not involved in the transformations.

Figure 4.1 shows a visual derivation when $B\neq B'$. In prose, starting with

$$A\otimes B\otimes C,\qquad A'\otimes B'\otimes C',$$

apply a plus on the A position, using $A'\otimes B'\otimes C'$ as the pivot and replacing $A\otimes B\otimes C$. This gives

$$A'\otimes B'\otimes C',\qquad X\otimes B\otimes C,\qquad A'\otimes B\otimes C.$$

Next, apply a plus on the B position, using $A'\otimes B'\otimes C'$ as the pivot and replacing $X\otimes B\otimes C$. This gives

$$A'\otimes B'\otimes C',\qquad X\otimes(B+B')\otimes C,$$
$$X\otimes B'\otimes C,\qquad A'\otimes B\otimes C.$$

Now apply a plus on the A position, using $X\otimes(B+B')\otimes C$ as the pivot and replacing $A'\otimes B\otimes C$. Since $A'+X=A$, this gives

$$A'\otimes B'\otimes C',\qquad A\otimes B\otimes C,\qquad X\otimes(B+B')\otimes C,$$
$$X\otimes B'\otimes C,\qquad X\otimes B\otimes C.$$

Finally, flip the two summands

$$X\otimes(B+B')\otimes C\qquad\text{and}\qquad X\otimes B'\otimes C$$

on the A position. This produces one zero summand and one copy of $X\otimes B\otimes C$. Thus the net effect is to restore the two original summands and add two copies of $X\otimes B\otimes C$.

It remains to handle $B=B'$. After the first plus on the A position (see Figure 4.1), we have

$$A'\otimes B\otimes C',\qquad X\otimes B\otimes C,\qquad A'\otimes B\otimes C.$$

Apply a plus on the A position, using $X\otimes B\otimes C$ as the pivot and replacing $A'\otimes B\otimes C$. Since $A'+X=A$, this gives

$$A'\otimes B\otimes C',\qquad X\otimes B\otimes C,\qquad A\otimes B\otimes C,\qquad X\otimes B\otimes C.$$

This again restores the two original summands and adds two copies of $X\otimes B\otimes C$. □

LEMMA 4.6. *Let S be a scheme of $\mathcal{M}^{(n)}$ over $\mathbb{F}_2$. For arbitrary nonzero X,Y,Z, there is a path from S to*

$$S\uplus\{X\otimes Y\otimes Z,\ X\otimes Y\otimes Z\},$$

in the matrix multiplication flip graph.

Proof. We first show how to add two copies of $X\otimes B_0\otimes C_0$ for some summand $A_0\otimes B_0\otimes C_0$ of S.

By Lemma 4.4, the A-components of the summands of S span $\mathbb{F}_2^{n\times n}$. Since $n\geq 2$, this space has dimension at least two. Therefore we may choose a summand $A_0\otimes B_0\otimes C_0$ with $A_0\neq X$. Extend A_0 to a basis $\mathcal{B}$ of $\mathbb{F}_2^{n\times n}$ using A-components of summands of S.

Write X in this basis. We claim that there exist basis elements $U_1,\dots,U_t\in\mathcal{B}$, each of which is the A-component of some summand of S, such that

$$X=A_0+U_1+\cdots+U_t$$

and every partial sum

$$P_j=A_0+U_1+\cdots+U_j,\qquad j=0,1,\dots,t,$$

is nonzero.

Indeed, if the coefficient of A_0 in the basis expansion of X is 1, then write

$$X=A_0+U_1+\cdots+U_t$$

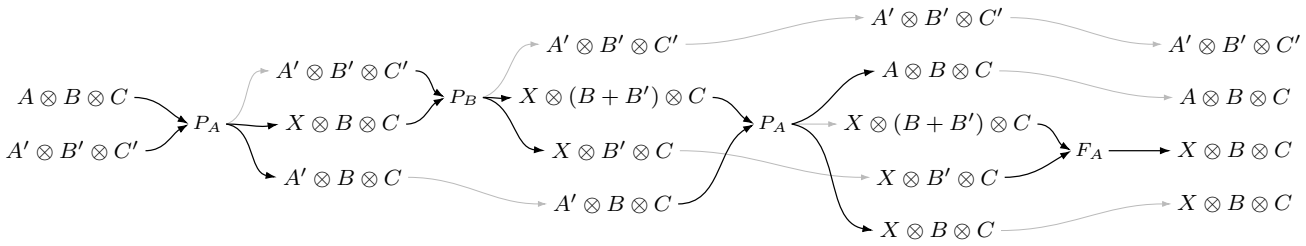

Figure 4.1: Derivation of S_{A+}, our name for the scheme $S \uplus \{X \otimes B \otimes C, X \otimes B \otimes C\}$ produced by Lemma 4.5. Here, X denotes $A + A'$, P_A and P_B denote a plus on the A and B positions of their inputs, and F_A denotes a flip on the A position.

using the remaining basis elements appearing in the expansion. Since $A_0 \neq X$, we have $t \geq 1$, and every partial sum is a nonempty sum of distinct basis elements, hence nonzero. If the coefficient of A_0 in the expansion of X is 0, write

$$X = U_1 + \cdots + U_m$$

and use the ordered expression

$$X = A_0 + U_1 + \cdots + U_m + A_0.$$

All proper partial sums contain A_0 together with some subset of the other basis elements, and are therefore nonzero; the final sum is $X \neq 0$. Now apply Lemma 4.5 iteratively. Starting from the summand $A_0 \otimes B_0 \otimes C_0$, the first application adds two copies of $P_1 \otimes B_0 \otimes C_0$. Using one of these copies together with a summand whose A-component is U_2, the next application adds two copies of $P_2 \otimes B_0 \otimes C_0$. Continuing in this way, we add two copies of $P_j \otimes B_0 \otimes C_0$ for each $j = 1, \ldots, t$, and in particular two copies of $X \otimes B_0 \otimes C_0$. The duplicate pairs corresponding to the intermediate partial sums $P_1, \ldots, P_{t-1}$ can be deleted using Observation 4.2. Thus we reach

$$S \uplus \{X \otimes B_0 \otimes C_0,\ X \otimes B_0 \otimes C_0\}.$$

We next change the B-component from B_0 to Y. If $B_0 = Y$, there is nothing to do. Otherwise, repeat the same argument in the B position, starting from one of the two copies of $X \otimes B_0 \otimes C_0$: using Lemma 4.4, choose a nonzero sequence of partial sums from B_0 to Y, apply Lemma 4.5 in the B position, and delete the duplicate pairs corresponding to the old component B_0 and to the intermediate partial sums. This leaves two copies of $X \otimes Y \otimes C_0$. If $C_0 = Z$, we are done. Otherwise, repeat the same argument in the C position, deleting the old duplicate pair $X \otimes Y \otimes C_0$ after the pair with C-component Z has been created. The result is $S \uplus \{X \otimes Y \otimes Z,\ X \otimes Y \otimes Z\}$, as desired. ∎

LEMMA 4.7. *Let S be a scheme for $\mathcal{M}^{(n)}$ over $\mathbb{F}_2$. Let R be a submultiset of S such that*

$$\sum_{r \in R} r = 0.$$

Then there is a path from S to $S \setminus R$ using only flip and plus edges.

Proof. We show how to replace every summand in R by a multiset of standard basis tensors, without changing the represented tensor. Since the summands in R sum to zero, the resulting basis tensors will occur with even multiplicity and can then be deleted in identical pairs.

Let $A \otimes B \otimes C$ be a summand in the current copy of R. If A is not a standard basis matrix, write

$$A = E_{i_1,j_1} + E_{i_2,j_2} + \cdots + E_{i_t,j_t}, \qquad t \geq 2.$$

By Lemma 4.6, add two copies of $E_{i_1,j_1} \otimes B \otimes C$. Use one of these two copies in a flip on the B position with $A \otimes B \otimes C$. Since the two summands have the same B-component and the same C-component, this replaces them by $(A - E_{i_1,j_1}) \otimes B \otimes C$ after deleting the zero summand. The other added copy remains. Hence the net effect is to replace

$$A \otimes B \otimes C$$

by

$$(A - E_{i_1,j_1}) \otimes B \otimes C \qquad \text{and} \qquad E_{i_1,j_1} \otimes B \otimes C.$$

Repeating this step decomposes the A-component into standard basis matrices. Applying the same procedure to the B-components and then to the C-components decomposes every summand of R into a multiset of standard basis tensors $E_{i,j} \otimes E_{k,\ell} \otimes E_{p,q}$. After decomposing all summands in R, the resulting scheme has the form $(S \setminus R) \uplus R'$, where R' is a multiset of standard basis

tensors. Since each decomposition step preserves the represented tensor, we have

$$\sum_{r \in R'} r = \sum_{r \in R} r = 0.$$

The tensors $E_{i,j} \otimes E_{k,\ell} \otimes E_{p,q}$ are distinct standard basis tensors of $(\mathbb{F}_2^{n \times n})^{\otimes 3}$, and hence are linearly independent. Therefore every standard basis tensor appearing in R' appears with even multiplicity (as their coefficient must be zero over $\mathbb{F}_2$). Pair equal tensors and delete each pair using Observation 4.2. This removes all of R' and leaves $S \setminus R$. □

We are now in a position to prove the main theorem.

Proof of Theorem 4.1. Let S and S' be two schemes for $\mathcal{M}^{(n)}$ over $\mathbb{F}_2$. Starting from S, use Lemma 4.6 to add two copies of every summand of S'. This gives a directed path from S to $S \uplus S' \uplus S'$. Since both S and S' sum to $\mathcal{M}^{(n)}$, the submultiset $R = S \uplus S'$ of $S \uplus S' \uplus S'$ sums to $\mathcal{M}^{(n)} + \mathcal{M}^{(n)} = 0$ over $\mathbb{F}_2$. By Lemma 4.7, there is a directed path from $S \uplus S' \uplus S'$ to $(S \uplus S' \uplus S') \setminus (S \uplus S') = S'$. Thus there is a directed path from S to S'. Since S and S' were arbitrary, the directed flip/plus graph is strongly connected. □

5 GPU-Accelerated Search

This section gives an overview of our algorithm and of a generalization of flip transforms suitable for a performant in-GPU implementation.

5.1 Search Procedure

The most performance-sensitive part of random walks on the flip graph is identifying flip opportunities, i.e. summands that match in the A, B, or C position.

$$\overbrace{A \otimes \underbrace{B}_{\text{Term at } B \text{ position}} \otimes C}^{\text{Summand}}$$

Over $\mathbb{F}_2$, an 8×8 term fits in a single 64-bit word, so existing flip graph search procedures for small matrix sizes maintain, for each position A, B, C, a map that stores the indices of summands with a particular value at that position. For example, if summands at indices $1, 2, 3$ have the value x for their A term and m_A is the map for A terms, then $m_A(x) = [1, 2, 3]$. For very small matrix sizes, 4×4 or less, this map can be implemented with an array by interpreting a term's bit-level representation as an index, whereas larger sizes require hashing. However, this approach is not GPU-friendly, as it is branchy, requires operating on sizable buffers in shared memory, and is generally difficult to adapt to the single instruction, multiple threads model used by GPUs.[3]

Fortunately, recent CUDA GPUs have a warp-level primitive, `__match_any_sync(mask, value)`, which takes the mask of participating lanes and a value, and returns the bitmask of participating lanes whose values equal the caller's. For a calling thread t, this means

$$\big(\texttt{\_\_match\_any\_sync}(x_t)\big)_i = 1 \quad \leftrightarrow \quad \text{participating thread } i \text{ has } x_i = x_t.$$

When there are fewer summands than threads in a warp, this can be used as follows: Each thread stores a single summand. At each step, all threads agree on a term position (A, B, or C) and execute `__match_any_sync` on their summand's value at that position. Ones in the resulting bitmasks correspond to flip opportunities. This means flips are identified with a hardware primitive instead of a hash table.

In practice, CUDA GPUs have 32 threads per warp. As world-record decompositions of $\mathcal{M}^{(4)}$ and $\mathcal{M}^{(5)}$ involve 47 [2] and 93 [6] summands, respectively, multiple summands must be stored per thread. Flip search then becomes probabilistic: At each step, warps agree on a term position; then, for k rounds, each thread picks a random summand and calls `__match_any_sync`. Increasing k increases the likelihood that available flip opportunities are found. Because each thread contributes one summand per round, two matching summands stored by the same thread are never matched with each other; detection relies on cross-thread matches and on the constant churn of components. We evaluate the effect of k in subsection 6.3, and Figure C.1 in Appendix C specifies the complete step.

5.2 Generalized Flips

With `__match_any_sync`, identifying multiple summands that share a term costs no additional matching instructions, as this corresponds to more than two bits being set in the returned bitmask. Flip semantics can be generalized to operate on multiple summands and used to look ahead for rank reductions.

Let us sketch a flip opportunity as matrix multiplication over subterms, with $\otimes$ as multiplication.

$$\begin{aligned} & A \otimes (B^{(1)} \otimes C^{(1)} + B^{(2)} \otimes C^{(2)}) \\ &= A \otimes \Big(\underbrace{\begin{bmatrix} B^{(1)} & B^{(2)} \end{bmatrix}}_{B} \underbrace{\begin{bmatrix} C^{(1)} \\ C^{(2)} \end{bmatrix}}_{C} \Big) \end{aligned}$$

[3] A GPU consists of streaming multiprocessors that execute threads in groups of 32, called warps, under the single-instruction-multiple-threads model: the threads (lanes) of a warp issue the same instruction on their own data. When control flow diverges within a warp, the two sides of a branch execute serially under different active masks, reducing utilization; this is why branchy code performs poorly. Warp-level primitives such as `__match_any_sync` let the lanes of a warp exchange and compare values directly.

From this perspective, applying a flip has the same effect as transforming BC, above, into $BGG^{-1}C$ for a particular G.

$$\begin{aligned} & A \otimes \Big(\begin{bmatrix} B^{(1)} & B^{(2)} \end{bmatrix} \overbrace{\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}}^{G} \overbrace{\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}}^{G^{-1}} \begin{bmatrix} C^{(1)} \\ C^{(2)} \end{bmatrix} \Big) \\ & = A \otimes (B^{(1)} + B^{(2)}) \otimes C^{(1)} \\ & \quad + A \otimes B^{(2)} \otimes (C^{(1)} + C^{(2)}) \end{aligned}$$

This is easily generalized to multiple summands with a shared term and an arbitrary invertible G.

$$\begin{aligned} & A \otimes (\sum_{i=1}^{N} B^{(i)} \otimes C^{(i)}) \\ & = A \otimes \Big(\begin{bmatrix} B^{(1)} & \dots & B^{(N)} \end{bmatrix} GG^{-1} \begin{bmatrix} C^{(1)} \\ \dots \\ C^{(N)} \end{bmatrix} \Big) \end{aligned}$$

A consequence of this framing is that if the $B^{(i)}$ or $C^{(i)}$ terms are not linearly independent, then an appropriate choice of G results in a zero in the B or C vectors, and a zero corresponds to a rank reduction. Thus, an appropriate choice of G can skip ahead in the search space to find rank reductions that may require many pairwise flip and plus transforms to set up the needed linear combination.

It remains to discuss how to efficiently determine on a GPU whether $B^{(1)}, \dots, B^{(N)}$ are linearly dependent. Sets of summands sharing a term appear to be overwhelmingly of size ≤ 5 on the 5×5 flip graph: among $43,386$ sets of more than two summands sharing a term in $3,273$ schemes sampled from random walks, 99.93% had size ≤ 5; Figure 5.1 visualizes this. Over $\mathbb{F}_2$, determining whether $B^{(1)}, \dots, B^{(5)}$ are linearly dependent reduces to determining whether there is a nonempty subset whose sum (i.e. XOR, i.e. $\oplus$) is zero. This makes 5 a natural design threshold: sets of size 5 have 31 nonempty subsets, and warps have 32 threads, so we can assign each subset to a thread. Larger groups are not enumerated; when a matching class has more than five members, the step falls back to an ordinary pairwise flip within the class. Figure 5.2 reports the corresponding group-size distributions for sizes 4×4 through 7×7, measured on schemes recorded along descents; groups within the threshold account for the large majority at every size.

More precisely, for a group of $N \leq 5$ summands and each thread index $0 \leq s < 2^N$, let $D_s \subseteq \{1, \dots, N\}$ be the set of one-based positions of the set bits of s. Threads s with $1 \leq s < 2^N$ compute

$$\bigoplus_{i \in D_s} B^{(i)} = 0.$$

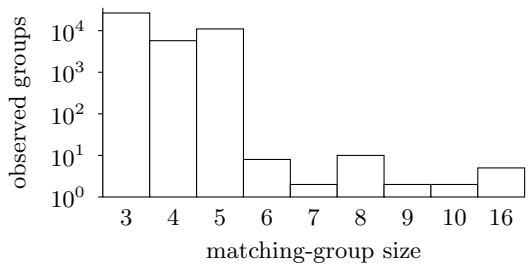

Figure 5.1: Distribution of sizes of sets of summands sharing a term from $3,273$ schemes sampled from random walks on the 5×5 flip graph.

Size	groups of size 3–5	size 6–8	size ≥ 9
4×4	0.1–7.9	0	0
5×5	11–18	0	0
6×6	23–34	0.4–2.6	0
7×7	33–49	3.4–10.2	0

Figure 5.2: Shared-factor groups per scheme by group size, measured on schemes recorded along descents from the naive scheme (two runs per size; ranges span descent phases from early to late). Groups of size at most five, which the warp enumerates exhaustively, account for the large majority at every size; larger groups fall back to ordinary pairwise flips.

A warp ballot then finds the lowest-index thread for which the above is true. If one exists, call it thread m, and let $r = \min D_m$ be the least index in its selected subset. The threads rewrite their summands from

$$\sum_{i=1}^{N} A \otimes B^{(i)} \otimes C^{(i)} \quad \text{to} \quad \sum_{\substack{i=1 \\ i \neq r}}^{N} A \otimes B^{(i)} \otimes C'^{(i)}$$

where

$$C'^{(i)} = \begin{cases} C^{(i)} \oplus C^{(r)} & i \in D_m, i \neq r \\ C^{(i)} & i \notin D_m \end{cases}$$

and the summand indexed by r is removed, which is implemented by setting it to zero.

To verify correctness,

$$\begin{aligned}\sum_{\substack{i=1\\ i\neq r}}^{N} A\otimes B^{(i)}\otimes C'^{(i)} &= \left(\sum_{\substack{i=1\\ i\neq r}}^{N} A\otimes B^{(i)}\otimes C^{(i)}\right)\\ &\quad\oplus \sum_{\substack{i\in D_m\\ i\neq r}} A\otimes B^{(i)}\otimes C^{(r)}\\ &= \left(\sum_{\substack{i=1\\ i\neq r}}^{N} A\otimes B^{(i)}\otimes C^{(i)}\right)\\ &\quad\oplus A\otimes B^{(r)}\otimes C^{(r)}\\ &= \sum_{i=1}^{N} A\otimes B^{(i)}\otimes C^{(i)}.\end{aligned}$$

Note that generalized flips do not move the search outside the space of schemes: a generalized flip yields another decomposition of $\mathcal{M}^{(n)}$, and by Theorem 4.1 any such decomposition is reachable from the previous scheme using ordinary flip and plus transformations alone. Whether the transformation applied by a generalized flip can always be realized as a short path of ordinary flips, and at what length, is a question we plan to study more closely (section 8).

For a theoretical analysis of the optimal rank reduction attainable by a generalized flip, see Appendix B.

6 Evaluation

6.1 Comparison with Prior Implementations As benchmarks, we compare our implementation to the previous GPU-accelerated search procedure by Perminov [7], which is the only other known GPU algorithm for flip graph search at the time of writing, and to the original CPU implementation of Kauers and Moosbauer [3] (their updated code, which incorporates the plus transition of the adaptive flip graph method [1]), measured on a single core of the same machine. Throughput is reported in search steps per second, where one step applies one elementary transition (a flip, a plus, or a group reduction) to one walk; for the baselines we count each implementation's analogous elementary transition. Our kernels are compiled with `nvcc -O3` for `sm_90`; the Kauers-Moosbauer baseline is compiled with `g++ -O3` from the authors' repository; and each reported cell is ten independent runs unless stated otherwise. Our implementation executes between 4 and 21,000 times as many search steps per second as Perminov's implementation for matrix sizes between 3×3 and 8×8, as shown in Figure 6.1.

Figure 6.2(a) shows how this throughput translates into search progress within a fixed time budget. Every implementation starts from the trivial rank-n^3 scheme and searches for 300 s; we repeat this 10 times per implementation and size and plot the number of ranks removed, so taller bars are better. The solid portion of each bar is the mean over the 10 runs, and the lighter extension reaches the best of the 10. One run means one search on the whole device: a single H200 for the GPU implementations, and for Kauers-Moosbauer an ensemble of 15 single-threaded processes (one per available host vCPU) reporting the ensemble best. Perminov's implementation is built in its $\mathbb{Z}_2$ mode with size-changing transitions disabled, so all three searches explore the same space of schemes; their transition sets differ, however: ours applies flips and plus transitions (generalized flips are enabled only for the ablation of subsection 6.2), Kauers-Moosbauer applies flips and plus transitions, and Perminov's kernel applies the moves of [7] with the size-changing ones disabled. At 4×4 and 5×5 every implementation makes progress, though only ours reliably reaches the record rank-47 scheme at 4×4 (in eight of ten runs; with generalized flips enabled it does so in all ten, subsection 6.2); at 6×6 and 7×7 the prior implementations remain within 3 ranks of the trivial scheme even in their best runs, while ours removes 10 and 8 ranks on average.

Size	Ours	Perminov [7]	Ratio	KM [3]	Ratio
3×3	1.75 G	449 M	∼4×	9.5 M	∼180×
4×4	1.81 G	236 M	∼8×	6.2 M	∼290×
5×5	1.89 G	76.3 M	∼25×	3.8 M	∼500×
6×6	1.52 G	11.9 M	∼127×	2.8 M	∼540×
7×7	1.32 G	1.24 M	∼1,060×	1.7 M	∼780×
8×8	1.08 G	51.5 K	∼21,000×	1.2 M	∼900×

Figure 6.1: Flip-graph search throughput in search steps per second: ours vs. Perminov's FlipGraphGPU [7] and the Kauers-Moosbauer CPU implementation [3] (single core), all over $\mathbb{F}_2$ from the trivial rank-n^3 scheme. Each ratio column compares ours to the implementation on its left. GPU measurements on an NVIDIA H200.

The ensemble design also supports a comparison at equal hardware cost rather than equal device count. Because each Kauers-Moosbauer process is an independent single-core walk, an N-core ensemble returns exactly the best of N independent runs, so the expected result for any N follows from the empirical distribution of single-core outcomes by order statistics. From the 150 independent single-core 300 s runs underlying Figure 6.2(a), an ensemble sized to match the cost of one H200 at on-demand cloud prices (roughly 88 vCPUs) is expected to reach ranks 51.6, 107.4, 215.0, and 341.2 at sizes 4×4 through 7×7, versus our single-GPU means of 47.4, 108.1, 205.8, and 335.2 in the same 300 s. (The measured configuration is the 15-process ensemble; the

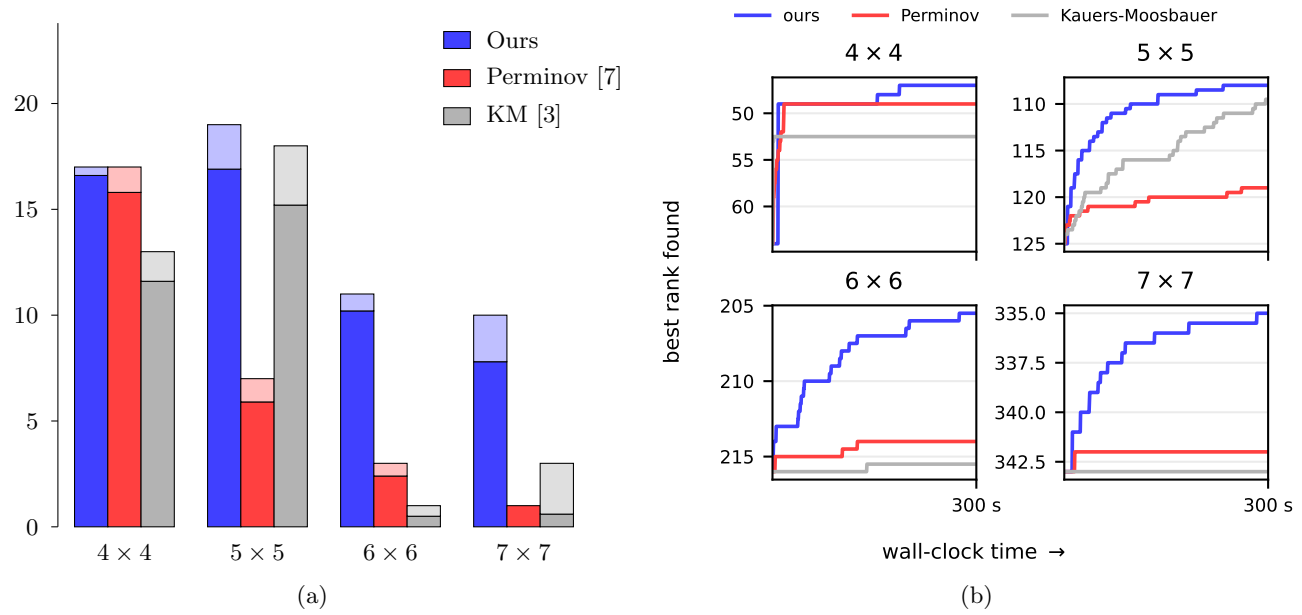

Figure 6.2: Search progress from the naive rank-n^3 scheme, over ten independent 300 s runs per implementation and size. **(a)** Rank reduction below the trivial scheme: solid bars are the mean, lighter extensions the best of the ten runs. **(b)** Median best rank found versus wall-clock time, with the vertical axis inverted so that better (lower rank) is higher; each curve's endpoint corresponds to the bar in (a) (bars: means; curves: medians). In both panels ours runs with generalized flips disabled; they are evaluated in subsection 6.2. Kauers-Moosbauer runs as a 15-process CPU ensemble; both GPU implementations use the H200.

88-core figures are expectations computed over the measured single-core distribution, not a separate run.) The comparison holds cost and wall-clock time fixed simultaneously; only 7 of the 150 single-core runs find any reduction at 6×6 and 5 of 150 at 7×7, so under the empirical single-core outcome distribution the predicted best-of-N result saturates rapidly at the larger sizes (bootstrap 95% intervals for the expected best-of-88 rank: [215.0, 215.3] at 6×6 and [340.3, 342.6] at 7×7).

Figure 6.2(b) shows the full search trajectories: for each implementation and size, the median best rank found so far as a function of wall-clock time, over the same ten runs. Our implementation reaches lower ranks than both prior methods at every size, and the gap is visible throughout the search rather than only at the final time limit. The descent is steepest in the first tens of seconds and then flattens, so most of the rank reduction available within a fixed budget is obtained early.

6.2 Generalized Flips Figure 6.3 ablates generalized flips. For discovering the current world-record scheme [2] for 4×4 matrix multiplication, generalized flips result in a 1.66× speedup on average. We report this ablation at 4×4 only: the rate at which walks encounter groups with a usable linear dependency falls sharply with the matrix size (in our measurements, by roughly three orders of magnitude between 4×4 and 7×7), so at larger sizes generalized flips fire too rarely to affect wall-clock search time. Making generalized flips productive at larger sizes, including computing the optimal reduction of Theorem B.1, is future work (section 8).

Search variant	Mean	Median	Min	Max	Hits
Generalized flips	42.13	36.53	7.94	78.98	10/10
No Generalized flips	69.83	65.47	26.33	126.70	10/10

Figure 6.3: Wall-clock time to discover a rank-47 decomposition of the $4 \times 4 \times 4$ matrix multiplication tensor on one NVIDIA H200. All runs started from the naive rank-64 decomposition, and all times are in seconds. The aggregate columns summarize ten independent runs for each variant. Generalized flips result in a mean 1.66× speedup.

6.3 The Probe-Round Parameter k The parameter k controls how persistently a step searches for a transition: each step attempts up to $3k$ matching rounds (k per tensor position) and applies the first can-

Task	k=1	k=2	k=4	k=6	k=8
4×4: naive $\to$ 47	–	2.6	8.6	51.3	17.0*
5×5: naive $\to$ 112	–	–	65.0*	47.0	37.1
6×6: naive $\to$ 210	–	45.0*	65.7	102	102
7×7: rank 248 $\to$ 246	–	–	21.0*	11.2	16.0

Figure 6.4: Mean wall-clock seconds to reach a fixed target rank as a function of the probe-round parameter k (three to five runs per cell). A dash means no run reached the target within the step budget; * marks cells where only some runs reached it. The first three tasks descend from the naive scheme; the last pushes the 7×7 frontier from a corpus of rank-248 schemes.

didate found (Figure C.1). One might expect k to trade throughput against the probability of finding available edges. Measured, it does not: because a step leaves its probe loop on the first hit and matches are plentiful, steps per second are flat in k (within 3% between $k = 1$ and $k = 8$ at every size we measured, e.g., 1.07 versus 1.04 billion steps per second at 7×7). Search quality, however, depends on k strongly and non-monotonically. Figure 6.4 reports wall-clock time to reach a fixed target rank: $k = 1$ fails on every task, and the best k varies with size and task. Since a plus transition is attempted only when every round misses, k indirectly sets the walk's flip-to-plus balance, and the best balance is size- and regime-dependent; per-size tuning of k is therefore necessary, and the configurations of subsection 6.1 use the tuned values.

6.4 Provenance of the Rank-245 Scheme The rank-245 scheme was found by a fully automated deployment of the search described in this paper, which ran unattended and continuously reseeded its walks from the pool of best schemes found so far (the mechanism of Figure C.1). The deployment predates the measurement harness used in this section, and its exact configuration, seeds, and timing were not recorded; we therefore make no quantitative claim about the cost of the discovery, and we treat the scheme itself, which is independently verifiable from Appendix A, as the result. The neighborhood of the record is reproducible, however: from a pool of rank-248 schemes produced by the same search, the tuned configuration of Figure 6.4 reaches rank 246 in about 11 seconds on average, consistently across runs.

7 Conclusion We presented a GPU-accelerated procedure for searching the matrix multiplication flip graph over $\mathbb{F}_2$. The implementation replaces global data structures for detecting flip opportunities with warp-local matching operations, making random walks on the flip graph better suited to modern GPUs. We also introduced generalized multi-summand flips, which act as GPU-efficient macro-transformations exposing local rank reductions through linear dependencies among summands.

On the theoretical side, we proved that over $\mathbb{F}_2$ the directed matrix multiplication flip graph is strongly connected using only ordinary flip and plus transformations. This removes reduction edges from the connectivity argument in the binary setting.

Experimentally, our implementation executes over a billion search steps per second on an NVIDIA H200 and substantially improves throughput over previous GPU-accelerated flip-graph search. Using this search procedure, we found a decomposition of the $7 \times 7 \times 7$ matrix multiplication tensor over $\mathbb{F}_2$ of rank 245, improving the previous best known rank by three multiplications; the scheme itself, together with a program that verifies it, is given in Appendix A.

Limitations. Our packed representation supports factors with up to 64 entries, i.e., square sizes through 8×8; all experiments are over $\mathbb{F}_2$, and we make no claim that the rank-245 scheme accelerates numeric matrix multiplication; the search is stochastic; dependency detection is exhaustive only for groups of at most five summands; and all measurements are on a single GPU architecture.

8 Future Work Four directions seem promising to us. First, extending our GPU implementation to search the symmetric flip graphs of Moosbauer and Poole [6], which reach lower ranks at 5×5 and 6×6 by restricting search to symmetric schemes. Second, computing optimal generalized flips on the GPU: Theorem B.1 characterizes the largest rank reduction a generalized flip can achieve, but our implementation does not attempt to find this optimum; a related theoretical question is which generalized flips can be realized as short paths of ordinary flips. Third, combining our search with systems such as FalconGEMM [9] to search directly for schemes that perform well on particular hardware, rather than for low rank alone. Fourth, extending our connectivity proof to flip graphs for structured matrix multiplication [5].

References

[1] Y. Arai, Y. Ichikawa, and K. Hukushima, *Adaptive flip graph algorithm for matrix multiplication*, in Proceedings of the 2024 International Symposium on Symbolic and Algebraic Computation, ISSAC '24, New York, NY, USA, 2024, Association for Computing Machinery, pp. 292–298, https://doi.org/10.1145/3666000.3669701.

[2] A. Fawzi, M. Balog, A. Huang, T. Hubert, B. Romera-Paredes, M. Barekatain, A. Novikov, F. J. R. Ruiz, J. Schrittwieser, G. Swirszcz, D. Silver, D. Hassabis, and P. Kohli, *Discovering faster matrix multiplication algorithms with reinforcement learning*, Nature, 610 (2022), pp. 47–53, https://doi.org/10.1038/s41586-022-05172-4.

[3] M. Kauers and J. Moosbauer, *Flip graphs for matrix multiplication*, in Proceedings of the 2023 International Symposium on Symbolic and Algebraic Computation, ISSAC '23, New York, NY, USA, 2023, Association for Computing Machinery, pp. 381–388, https://doi.org/10.1145/3597066.3597120.

[4] M. Kauers and I. Wood, *Exploring the meta flip graph for matrix multiplication*, 2025, https://doi.org/10.48550/arXiv.2510.19787, https://arxiv.org/abs/2510.19787. arXiv preprint arXiv:2510.19787.

[5] K. Khoruzhii, P. Gelss, and S. Pokutta, *Faster algorithms for structured matrix multiplication via flip graph search*, 2025, https://doi.org/10.48550/arXiv.2511.10786, https://arxiv.org/abs/2511.10786. arXiv preprint arXiv:2511.10786.

[6] J. Moosbauer and M. J. Poole, *Flip graphs with symmetry and new matrix multiplication schemes*, in Proceedings of the 2025 International Symposium on Symbolic and Algebraic Computation, ISSAC '25, New York, NY, USA, 2025, Association for Computing Machinery, pp. 233–239, https://doi.org/10.1145/3747199.3747566.

[7] A. I. Perminov, *Fast matrix multiplication via ternary meta flip graphs*, 2025, https://doi.org/10.48550/arXiv.2511.20317, https://arxiv.org/abs/2511.20317. arXiv preprint arXiv:2511.20317.

[8] I. Wood, *Exploring commutative matrix multiplication schemes via flip graphs*, 2025, https://doi.org/10.48550/arXiv.2506.22113, https://arxiv.org/abs/2506.22113. arXiv preprint arXiv:2506.22113.

[9] H. Zhu, J. Cao, J. Shao, S. Feng, Q. Qiu, P. Chen, X. Zhang, Y. Zhou, M. L. Yiu, G. Ji, M. Deng, J. Meng, and W. Zhu, *FalconGEMM: Surpassing hardware peaks with lower-complexity matrix multiplication*, 2026, https://arxiv.org/abs/2605.06057.

A The Scheme This block encodes our rank-245 decomposition of the $7 \times 7 \times 7$ matrix multiplication tensor over $\mathbb{F}_2$. For each rank-one term, its three 7×7 binary coefficient matrices are stored as seven little-endian bytes each, in A, B, C order. Within each matrix mask, bit $7i + j$ records coordinate $[i, j]$. The resulting byte string is compressed with zlib and encoded with Base32.

```
BEGIN-RANK245-7X7X7-ZLIB-BASE32
PDNF2WCNRAOMOFL6KXJTHWY2QZG3PPJWVM4IPXTGBHBSFVQTWBX3UFBNXTDAJWEN2GBO4Q4IIYE
X4CBW7WMDB22NGS3J4KCWQ4TTZFSZYQ5RYEDRQQSJAOAFBPIG3DWSEB5COPSZUDUXFQJEQHHDCG7
N5226B3JM7SJWV7VL26XV5N5V7DIHPCSQAAHAHUKL45X5ECXIF77577AYEZHIQMPP4XI6RYEMYBO
L6AJY7KZB2CT46WV6MK6IJ6AKHSHWNKAMJPU7F6XCMPNQAABGBFP6IMP5LR4ESR6AHMMGPMPPYQB
DS6R3S7QEYI5GMWQOMZ6RBM5BGSH2BMPQ3SHMLFLRFF2VEUME6CHMSPE7K5ZB6CQCUI2YN77GD6H
B3SCAJXBMF2IMA5RSTVNV4NAP3JDQB4DKYPESPSHD6HR3Q2RQP4C2OWQRNWDTVSOJDLDXA4RXRP4
ZDRMH6PZQBOXQWVOLPMGUA6GSJLETQGEFV6SBFSJM5H5DJFHDND6HUT6F7KBHSBLJWJTUSRI56LS
OI4LPYLIPZYLAPKRDNDJ6KQVYA246EWXYRVLGSFXLA47TSJSYCQYGCEGPADWFKF4LAOD3HTG6I4Z
OGC4CBIOD6AZPNPH2VRQV4XWKVE6DJ4Q5ZZWDJT6D34EH3VQ7S6GJ27OP5D7KX7OLDW4FHSMWMCD
AZ2AKIYG7KY3S4OZ4T22WDKNEC6POZJL754WATBRXEF2FOZFPP6TGLTTKXB4RTFQIW3GDL7GUTZ4
ONCVK2DANYYSJMSVJ3ZLNTG7PW4ZZZKOYEJL4ZGQPUFFSBUN4TK4GGWPTLTYBFSYU7RGHIJAF5J3
GC3TZH23GZ2ADCYL6VPW3YHQEHTRIK3MFOJTEAPZTCM5C3H76OM3HZ6IUMYYHDYHHSPRPNBWFAZU
EEICHP6XUWEBKGOOXEX4H6IOUPQHA6L2JNVG2XMHION4EFEKTZ4JOA76D7TKL2UHHWGYAQ3SCSJK
NNL6MYCNOPRQB4SZOOMDVVNQKCPPSTG6DTEOQ6LM3DF7UGONL6G645RPGCPRUCAEYQ2GLUS4ZTMS
BOGB37L3MNXJNST4JHT6BPZXZRJJP5HOGFKWJEMZWJKUQ6UBIHSSYSNEGJFAVPCVTRZN4I2KJCZ2
CICXDAPH4NPGKW6TSFLGXF4KCL3LMVII43ZJOGTMDGZFD66OPVXAAJOIMFIGC46I7NZUVVFCNMNE
BJED7FBNTAUMQRWIXOQ2MOE7RCLPQ5DXHHKPOIO5PIRFUIXYTCTG4YJB6E3QKOP2IV2TO6UHJYT4
SWV232XJFOR7GV5NDN7NZL5FDVV6SES2MXWXCDRGIN243QMOLXNL2J6REA4ZEOXYENNJZKNJJACX
HN6FD2E4B67XRUABWVCODF25K2YDO3ETPJCNURLJH6SVRKLDGQHM4QLREOPECBTRNYBJ2OHVWGNZ
7C2BCRVASZ74EVTZV3A3WIRAHJ5W7YQJHDLA5WLANVLWZF5LYMV462OAWI3QW7NZYD4YCUEVJM2M
LKSFVTREZEMNXNLJV6PD2ZI6WR2VFU4UOZPB2CBYEFYALU7DDKWPMJQWORQHVAJGGUZQFKMZVKS6
BINCGZYUIE4GZXOH35UIJOJW7A45FYFMLHVWXKOYT3QAFUBA7F43N3OXXYNJD2OTQV2HSZBFAGJK
VXWRRMVFRSTNFR3NDDVLNIC4DH3OCCGBNXVTFUWH7JZQFEKUNLTK3ARKNPOJ7756MYHFDUDWNQ3R
MCBKRAFEG2UQ6N3GHU2RLFASJMA3ROPSYMNEL4W4CQYEW5Y5U7VMAD2E4YF7EPCFB4Z5BH75VZPF
TOSVVYYGZTFJXRBPMERYPNY7OEQYEH5BEMBLCQVYLKSSCKOILUKETVLVJAKWM6UMAUGN4TQQSUDW
435HIKKDJOJHYC34UZS5KVQOK5HYWRSBBB5XQ6H2WCZXMAN7FOWPBYVR37TOCSIYJ3FTEWAWRKJA
4HDRWELMY6ZH5465OOC4KSN3PFL6ELGAPAB3PQ354U54VCNW4MPU7QWD76EWLG354ZHRK3DLKJCB
PWC2KXOIYJ6V6LHHBRCW2RT3DIZE6BFDS77F7MPCEG3SWYYLFIHNA7LFCRHBIFEGSKYGMBKU5SUK
YA4BYFF4QIQPK54ZSP5DWSSNSKFSINL74I5HSDRIE5JSHDS6QBEQJS7Z6URINVA4TOS4J5RIC6LY
AO25P3DS6RYEWIDY6B4WYH2GB7IKQKSC4BR2E5XOV2XI7JWGIKRFNS3BXE2BPFRYG444RXRNIGPL
J674OIH6BNUISCVWQBHVASYKPMYTTEBOFUS4S5AIKGMLVSQRZAZHKYJHCOKSVANEKQCSJJOKKHRY
5QM7RLIF25WZPUBYX7HHPX2Y53JOYVUDVFVXKTIEDSAKSIAUHWDTPJLFFO4UFS7BGOK5RTD2YUNY
WOWS4IZLUQSPTRZGI4ELJZEMLFCITZNGJC4EAQR6KNMTQSTQBWUAIGCE3GNTSBSFDYOBASCCQLAX
LFAHFQO4PFS5BTBHX4VSMAINK7LWZKQPJ53X5XDMAYDJK6OASPTNXBXEFJD2VDDJ2TMNILRIASYK
QUCIGLBEPK6TAQJQM6SWVTOT54E7QZ2OTU2CEPC4A4CFEKC4CZ2B54D6CRTBHOY76GEZXEGM3ZRX
BYNMDBZ7S5GJDIV6STCKOKCUV7ASVWRZA7ECRHJEXHDAAEKX7CBAROXILNPLUXVBNA2WMD6EFOPV
GWB2BFWWZPSX4OHZ5WUVCNLHU6BD6JVD6CRYKMIALXXAJZGLTVUBISU6LYX565ER5G2R3AWURVKF
QTWX4EUQ3D6EEKBKBYJXEEAU2NA76XTHW4H5WZVYKLJI7MZ5WVADA47RHS4GFUG2YVGVCAUBPW4B
LVWPDP5WMQXI2KJ6EEESZDDGTJG2MC7CD667CZSKIEL4J6YJWLG5CONRBSPV25H4TMAIHLZLFD6T
24ZH6H3QE4OEMR2WFCIF7WWQD766H3Q4NRYXFVDUFIQFL5PRO4AHOX5756AKQL2F3UJZTCMPEUZ3
PLDZXYPZZ44YGAYC7WIYCO6URNNRACAUEGH5SFKQDMFFUFSYXMPZ7GV6UPV6UDSXZH23HYFLFCPP
6ON27T2B2BWPCL2YW5Z2LJK2JH3VRPBL6AY5WZCGMLHBA5VV5MUMS5IHED5Y5DBT6DNT36MNBY43
PZHZ44GSQV3ZTNNTFZUHWJETPEIVZSAFIS7U2RUOTB7NCC25QFIQMNDEFFOJP4OPBIQ42BNEZSY2
LXVAPG4FZFVT7ABUIQK3WUFJ3K2BTEZYRQKL4ROHZZLBASMS4BQZQMUQ6J7R4L2M5TE4LTFJZQLV
DKWKWJDVE7LCTKBDXCLV3673QV3AJAA5C7XT2CF65FPIFPY76UC4YTTNKN3MHY3BNHD3LOJKGB2U
QLAOLZZ625OCRCS77GX6AJQJFH5I2LPXJLJWEGVKJN6J7MDOEHPBU37ZZQMLLVYPFUT57ADBB67C
5Q
END-RANK245-7X7X7-ZLIB-BASE32
```

This Python program reads a copy of the scheme above from standard input and verifies that it represents the $7 \times 7 \times 7$ matrix multiplication tensor over $\mathbb{F}_2$.

```
import base64
import sys
import zlib

BEGIN = "BEGIN-RANK245-7X7X7-ZLIB-BASE32"
END = "END-RANK245-7X7X7-ZLIB-BASE32"
N = 7
RANK = 245

def support_bits(mask):
    return [b for b in range(N * N) if mask & (1 << b)]

text = sys.stdin.read()
payload = text.split(BEGIN, 1)[1].split(END, 1)[0]
payload = "".join(payload.split())
payload += "=" * (-len(payload) % 8)
data = zlib.decompress(base64.b32decode(payload))
if len(data) != RANK * 3 * N:
    raise ValueError("unexpected decoded length")
blocks = set()
for o in range(0, len(data), 21):
    blocks.add(data[o : o + 21])
if len(blocks) != RANK:
    raise ValueError("duplicate rank-one terms")

scheme = []
for o in range(0, len(data), 21):
    masks = [
        int.from_bytes(data[o + p : o + p + 7], "little")
        for p in (0, 7, 14)
    ]
    if any(mask >> (N * N) for mask in masks):
        raise ValueError("nonzero padding bits")
    supports = [support_bits(mask) for mask in masks]
    if any(not s for s in supports):
        raise ValueError("zero factor in rank-one term")
    scheme.append(supports)
if len(scheme) != RANK:
    raise ValueError("wrong number of rank-one terms")

parity = set()
for A, B, C in scheme:
    for a in A:
        for b in B:
            for c in C:
                key = (a, b, c)
                if key in parity:
                    parity.remove(key)
                else:
                    parity.add(key)

expected = set()
for i in range(N):
    for j in range(N):
        for k in range(N):
            expected.add(
                (i * N + j, j * N + k, i * N + k))

if parity != expected:
    raise ValueError(
        f"{len(expected - parity)} missing, "
        f"{len(parity - expected)} extra")

print("verified rank-245 decomposition of M^(7) over F_2")
```

B Rank Reduction of Generalized Flips We now formalize the best possible rank reduction that can be obtained from a generalized flip on a group of summands sharing one tensor factor. Suppose

$$T = \sum_{i=1}^{N} A \otimes B_i \otimes C_i$$

is a group of N summands that share the first factor. Consider $E = \mathbb{F}_2^N$ with the standard basis $e_1, \dots, e_N$, and define linear maps

$$B : E \to V_B, \qquad e_i \mapsto B_i$$

and

$$C : E \to V_C \qquad e_i \mapsto C_i.$$

Equivalently, the "non-shared" part of the group is the order-two tensor

$$T_{BC} = \sum_{i=1}^{N} B_i \otimes C_i \in V_B \otimes V_C.$$

A generalized flip chooses the basis $g_1, \dots, g_N$ of E and the dual basis $h_1, \dots, h_N$ of E^*. It then rewrites

$$\sum_{i=1}^{N} A \otimes B_i \otimes C_i$$

as

$$\sum_{j=1}^{N} A \otimes B(g_j) \otimes C(h_j),$$

where we identify C with the induced map $E^* \to V_C$ given by

$$h \mapsto \sum_{i=1}^{N} h(e_i) C_i.$$

Equivalently, if $G \in \mathrm{GL}_N(\mathbb{F}_2)$ has columns $g_1, \dots, g_N$, this is the transformation

$$B, C \longmapsto BG,\ G^{-1}C.$$

Theorem B.1 (Optimal Generalized Flip Reduction). *Let*

$$T = \sum_{i=1}^{N} A \otimes B_i \otimes C_i$$

be a group of N summands sharing the first tensor factor. Among all generalized flips

$$B, C \longmapsto BG,\ G^{-1}C, \qquad G \in \mathrm{GL}_N(\mathbb{F}_2),$$

the minimum possible number of non-zero surviving summands is

$$\mathrm{rank}(T_{BC}),$$

where

$$T_{BC} = \sum_{i=1}^{N} B_i \otimes C_i$$

is viewed as an order-two tensor, or equivalently as a linear map $V_C^* \to V_B$. Therefore the maximum number of summands that can be deleted by a generalized flip on this group with a shared factor is just

$$N - \mathrm{rank}(T_{BC}).$$

Analogous statements hold for groups sharing the B or C factor.

Proof. Let

$$U = \ker B \subseteq E.$$

A transformed B-factor is zero precisely when its basis vector lies in U. Next define

$$W = \{h \in E^* : C(h) = 0\}.$$

Thus a transformed C-factor is zero precisely when the corresponding dual basis vector lies in W. Let

$$L = W^{\perp} \subseteq E.$$

Equivalently, L is the image of the dual map

$$C^* : V_C^* \to E.$$

The order-two tensor

$$T_{BC} = \sum_{i=1}^{N} B_i \otimes C_i$$

corresponds to the linear map

$$V_C^* \longrightarrow V_B, \qquad \varphi \longmapsto \sum_{i=1}^{N} \varphi(C_i) B_i.$$

This map factors as

$$V_C^* \xrightarrow{C^*} L \xrightarrow{B|_L} V_B.$$

Since C^* has image L, we have

$$\mathrm{rank}(T_{BC}) = \mathrm{rank}(B|_L) = \dim L - \dim(U \cap L).$$

We now construct a generalized flip with exactly this many surviving summands. Choose a basis of E adapted to the pair U, L as follows: first choose a basis of $U \cap L$, extend it to a basis of L, and then extend it to a basis of all of E. Let $g_1, \dots, g_N$ be the resulting basis, and let $h_1, \dots, h_N$ be the dual basis.

If $g_j \in U \cap L$, then

$$B(g_j) = 0,$$

so the jth transformed summand is deleted. If $g_j \notin L$, then the dual vector h_j annihilates L, hence lies in $L^{\perp} = W$, and so $C(h_j) = 0$, and the jth transformed summand is also deleted.

The only surviving summands correspond to those basis vectors in

$$L/(U \cap L).$$

The number of surviving summands is

$$\dim L - \dim(U \cap L) = \operatorname{rank}(T_{BC}).$$

It remains to show optimality. Every generalized flip rewrites T_{BC} as a sum of surviving non-zero terms

$$B(g_j) \otimes C(h_j).$$

Thus, if a generalized flip leaves s non-zero summands, then T_{BC} has an order-two tensor decomposition using s simple tensors. Hence

$$s \geq \operatorname{rank}(T_{BC}).$$

The construction above achieves equality, so the minimum possible number of survivors is exactly $\operatorname{rank}(T_{BC})$. □

Theorem B.1 suggests a natural implementation question: can the optimal local rank reduction be computed efficiently enough to improve a random-walk search, especially for groups larger than a single warp can enumerate by subset tests? Making this optimal reduction efficient on the GPU is one of the directions we discuss in section 8.

C The Search Algorithm Figure C.1 specifies one random walk of the search. One walk is executed by one warp, and W walks (4096–8192 in our configurations) run concurrently. All walk state resides in registers: each of the 32 lanes owns s slots, a slot holds one summand (A, B, C) as packed n^2-bit masks, and per-lane masks track which slots are live, so a warp collectively holds a scheme of up to $32s$ summands.

Three details deserve emphasis. First, a lane contributes one summand per matching round, so two matching summands stored by the same lane are never matched with each other; detection relies on cross-lane matches and on the constant churn of components. Second, the early exit taken as soon as any lane holds a candidate is why measured throughput is flat in k (subsection 6.3). Third, walks run in launches of a fixed step budget; after each launch the host appends improved schemes to an on-disk pool, and the next launch reseeds every walk from the accumulated pool with fresh random streams. A plus transition is skipped when every slot is live (the walk is at capacity).

```
for step = 1, ..., T do
    p ← uniform random position in {A, B, C}
    for round = 1, ..., 3k do
        each lane reads the factor v at position p of a uniform random live slot
        m ← __match_any_sync(active, v)                          ▷ mask of lanes holding equal v
        if 3 ≤ |m| ≤ 5 then
            lane i tests subset i of the group: XOR of both residual factors, packed in one 64-bit word
            D ← lowest-indexed zero subset, if any               ▷ lowest index implies inclusion-minimal
            if D found then record candidate (REDUCE, random key)
            end if
        end if
        every lane with |m| ≥ 2 records candidate (FLIP, random key)
        if any lane holds a candidate then break
        end if                                                   ▷ early exit: throughput is flat in k
        p ← next position, cyclic
    end for
    c ← warp arg-max over candidates by (kind, key)              ▷ REDUCE outranks FLIP
    if no candidate then
        with probability 1/d, and only if r < capacity: apply a plus (Definition 3.3) to a random pair
    else if c is FLIP then
        apply the flip (Definition 3.3) at p to the winner and a random member of its class; delete zeroed summands
    else
        delete one summand of D; add its residual factor into the rest of D; delete any summand this zeroes
    end if
    update the rank r; on improvement, snapshot the scheme
end for
```

Figure C.1: One walk of the GPU search, executed by one warp with all steps warp-synchronous; W walks run concurrently.