Fast Output-Sensitive Matrix Multiplication

Riko Jacob[1] and Morten Stöckel[2] *

[1] IT University of Copenhagen
rikj@itu.dk
[2] University of Copenhagen
most@di.ku.dk

Abstract. [3] We consider the problem of multiplying two $U \times U$ matrices A and C of elements from a field $\mathbb{F}$. We present a new randomized algorithm that can use the known fast square matrix multiplication algorithms to perform fewer arithmetic operations than the current state of the art for output matrices that are sparse.

In particular, let ω be the best known constant such that two dense $U \times U$ matrices can be multiplied with $\mathcal{O}(U^\omega)$ arithmetic operations. Further denote by N the number of nonzero entries in the input matrices while Z is the number of nonzero entries of matrix product AC. We present a new Monte Carlo algorithm that uses $\tilde{\mathcal{O}}\left(U^2 \left(\frac{Z}{U}\right)^{\omega-2} + N\right)$ arithmetic operations and outputs the nonzero entries of AC with high probability. For dense input, i.e., $N = U^2$, if Z is asymptotically larger than U, this improves over state of the art methods, and it is always at most $\mathcal{O}(U^\omega)$. For general input density we improve upon state of the art when N is asymptotically larger than $U^{4-\omega}Z^{\omega-5/2}$.

The algorithm is based on dividing the input into "balanced" subproblems which are then compressed and computed. The new subroutine that computes a matrix product with balanced rows and columns in its output uses time $\tilde{\mathcal{O}}\left(UZ^{(\omega-1)/2} + N\right)$ which is better than the current state of the art for balanced matrices when N is asymptotically larger than $UZ^{\omega/2-1}$, which always holds when $N = U^2$.

In the I/O model — where M is the memory size and B is the block size — our algorithm is the first nontrivial result that exploits cancellations and sparsity of the output. The I/O complexity of our algorithm is $\tilde{\mathcal{O}}\left(U^2(Z/U)^{\omega-2}/(M^{\omega/2-1}B) + Z/B + N/B\right)$, which is asymptotically faster than the state of the art unless M is large.

1 Introduction

In this paper we consider computing the matrix product AC of two matrices A and C in the case where the number of nonzero entries of the output AC is

* This work was done while at IT University of Copenhagen. Supported by the Danish National Research Foundation / Sapere Aude program and VILLUM FONDEN.

[3] The final publication is available at Springer via http://dx.doi.org/10.1007/978-3-662-48350-3_64

sparse. In particular we consider the case where matrix elements are from an arbitrary field and cancellations can be exploited, i.e., "Strassen-like" methods are allowed.

The case of sparse output is well-motivated by real-world applications such as computation of covariance matrices in statistical analysis. In general, matrix multiplication is a fundamental operation in computer science and mathematics, due to the wide range of applications and reductions to it — e.g. computing the determinant and inverse of a matrix, or Gaussian elimination. Matrix multiplication has also seen lots of use in non-obvious applications such as bioinformatics [18], computing matchings [15,12] and algebraic reasoning about graphs, e.g. cycle counting [2,3].

Our main result is a new output-sensitive Monte Carlo algorithm, that given as input the $U \times U$ matrices A and C computes the nonzero entries of AC with high probability. For Z the number of nonzero output entries of AC, and N the number of nonzero entries of A and C, our algorithm uses $\tilde{O}\left(U^2\left(\frac{Z}{U}\right)^{\omega-2}+N\right)$ arithmetic operations and its I/O complexity, where memory size M and block size B are parameters in the model, is given by $\tilde{\mathcal{O}}\left(U^2(Z/U)^{\omega-2}/(M^{\omega/2-1}B)+Z/B+N/B\right)$.

The algorithm exploits cancellations, both to avoid computing zero entries of the output and by calling a Strassen-like algorithms.

When the input is dense, i.e., $N = U^2$, the RAM bound is strictly better than all state of the art methods when $Z \gg U$ and is never worse than $O(U^\omega)$. The I/O bound is strictly better than state of the art unless M is large — see Section 1.3. We note that the algorithm works over any field, but not over any semiring.

1.1 Preliminaries

We analyze our algorithms in the standard word-RAM model we assume that the word size is large enough to fit a field element. We further analyze our algorithms in the external memory model [1], where we have a disk containing an infinite number of words and a internal memory that can hold M words. A word can only be manipulated if it is residing in internal memory and words are transferred to internal memory in blocks of B words at a time and the number of such block transfers is called the I/O performance of the algorithm. Here we assume that a word can hold a field element as well as its position in the matrix.

Let A be a $U_1 \times U_3$ matrix and C be a $U_3 \times U_2$ matrix over any field $\mathbb{F}$, then $A_{i,j}$ is the entry of A located in the i'th row and j'th column and $A_{i,*}$ will be used as shorthand for the entire i'th row ($A_{*,i}$ for column i). The matrix product is given as $(AC)_{i,j} = \sum_{k=1}^{U_3} A_{i,k}C_{k,j}$. We say that a sum of elementary products *cancel* if the sum over them equals zero even though there are nonzero summands. We allow ourselves to use cancellations, i.e., "Strassen-like" methods, and our algorithms does not produce output results that are zero. We use log for the logarithm with base 2 and ln for the natural logarithm and when used in a context that requires integers, we let log stand for $\lceil \log \rceil$ and ln stand for $\lceil \ln \rceil$. Throughout this paper we will use $f(n) = \tilde{\mathcal{O}}(g(n))$ as shorthand for

$f(n) = O\left(g(n) \log^c g(n)\right)$ for any constant c. Here n stands for the input size, in our context it can always be taken as U. We let $\text{sort}(n) = \mathcal{O}((n/B) \log_{M/B}(n/B))$ be shorthand for the I/O complexity [1] of sorting n elements. Note that input layout is irrelevant for our complexities, as sorting the input to a different layout is hidden in $\tilde{\mathcal{O}}\left(\cdot\right)$, i.e., $O(sort(n)) = \tilde{\mathcal{O}}\left(n/B\right)$. When we use this notation there is usually at least a binary logarithm hidden in the $\tilde{\mathcal{O}}\left(\right)$-notation and not only the $\log_{M/B}(n/B)$-factor of sorting.

We will let $f \gg g$ denote that f is asymptotically larger than g. Let ω denote the real number for which the matrix product of two $U \times U$ matrices can be computed in time $\mathcal{O}(U^\omega)$. For a hash function $h \colon [U] \to [d]$ we define the binary projection matrix $P_h \in \{0,1\}^{U \times d}$ by $(P_h)_{i,j} = 1 \iff j = h(i)$. Finally we say that an algorithm has a *silent failure* to mean that the algorithm finished without an error, but the output is not correct.

We will use the following easy fact about the number of arithmetic operations $F_{\text{RAM}}(U,V,W)$, and I/Os $F_{\text{I/O}}(U,V,W)$ needed to multiply a $U \times V$ matrix by a $V \times W$ matrix.

Fact 1 (Folklore) *Let ω be the smallest constant such that an algorithm to multiply two $n \times n$ matrices that runs in time $\mathcal{O}(n^\omega)$ is known. Let $\beta = \min\{U,V,W\}$.*

Fast matrix multiplication has $F_{\text{RAM}}(U,V,W) = \mathcal{O}\left(UVW \cdot \beta^{\omega-3}\right)$ running time on a RAM, and uses
$F_{\text{I/O}}(U,V,W) = \mathcal{O}\left(F_{\text{RAM}}(U,V,W)/(M^{\omega/2-1}B) + (UV + VW + UW)/B\right)$ I/Os.

Proof. Assume wlog that β divides $\alpha = UVW/\beta$. Since β is the smallest dimension we can divide the matrices into α/β^2 submatrices of size $\beta \times \beta$, which can each be solved in $\mathcal{O}\left(\beta^\omega\right)$ operations. The I/O complexity follows from an equivalent blocking argument [10]. ■

For U,V,W and U',V',W' with $UVW = U'V'W'$ we have $F(U,V,W) > F(U',V',W')$ if $\min\{U,V,W\} < \min\{U',V',W'\}$, i.e., the “closer to square” the situation is, the faster the fast matrix multiplication algorithm runs, both in terms of RAM and I/O complexity.

1.2 Our results

We show the following theorem, that provides an output sensitive fast matrix multiplication algorithm granted that the output is balanced.

Theorem 2. *Let A and C be $U \times U$ matrices over the field $\mathbb{F}$ that contain at most N nonzero entries and the product AC contains at most $Z \geq U$ nonzero entries in total and at most $\Theta(Z/U)$ per row and per column. Then there exists an algorithm for which it holds:*

(a) The algorithm uses $\tilde{\mathcal{O}}\left(UZ^{\frac{\omega-1}{2}} + N\right)$ time in the RAM model.

(b) The algorithm uses $\tilde{\mathcal{O}}\left(UZ^{\frac{\omega-1}{2}}/(M^{\omega/2-1}B) + Z/B + N/B\right)$ I/Os

(c) With probability at least $1 - 1/U^2$ the algorithm outputs the nonzero entries of AC.

We then show the main theorem, a fast matrix multiplication algorithm that works on any input and is sensitive to the average number of nonzero entries in the rows and columns of the output.

Theorem 3. *Let A and C be $U \times U$ matrices over field $\mathbb{F}$ that contain at most N nonzero entries and the product AC contains at most Z nonzero entries in total. Then there exists an algorithm for which it holds:*

(a) The algorithm uses time $\tilde{\mathcal{O}}\left(U^2(Z/U)^{\omega-2} + N\right)$ time in the RAM model.
(b) The algorithm uses $\tilde{\mathcal{O}}\left(U^2(Z/U)^{\omega-2}/(M^{\omega/2-1}B) + U^2/B\right)$ I/Os
(c) With probability at least $1 - 1/U^2$ the algorithm outputs the nonzero entries of AC.

The algorithm of Theorem 2 has asymptotically lower running time compared to that of Theorem 3 for $1 < Z < U^2$ (for current ω) and for $Z = U^2$ they both match $\tilde{\mathcal{O}}(U^\omega)$. However, the algorithm from Theorem 2 requires balanced rows and columns and in fact the algorithm from Theorem 3, which works in the general case, is based on calling it on balanced partitions. For the sake of simplicity we state and proof Theorem 3 for square inputs only and note that there is nothing in our arguments prevents the generalization to the rectangular case. To the knowledge of the authors there are no previously known output-sensitive I/O-efficient algorithms that exploits cancellations and we outperform the general dense as well as the optimal sparse algorithm by Pagh-Stöckel unless M is large. We summarize the results of this section and the closest related results in Table 1.

Result structure. The paper is split into three parts: the row balanced case, subdivision into balanced parts, and the balanced case. After a discussion of the related work in Section 1.3, we consider in Section 2 the case where we have an upper bound on the number of output entries in each row of the output. In this case we can compress the computation by reducing the number of columns, where the magnitude of the reduction is based on the upper bound on the output entries. Then in Section 3 we make use of the row balanced compression, by showing that a potentially unbalanced output can be divided into such balanced cases, which gives Theorem 3. In Section 4 we show that if there is an upper bound on the number of output entries in *both* rows and columns, then we can compress in both directions which yields Theorem 2.

1.3 Related work

Two $U \times U$ matrices can trivially be multiplied in $\mathcal{O}(U^3)$ arithmetic operations by U^2 inner products of length U vectors. The first one to improve upon the $\mathcal{O}(U^3)$ barrier was Strassen [17] who for $\omega = \log_2 7$ showed an $\mathcal{O}(U^\omega)$ algorithm by exploiting clever cancellations. Since then there has been numerous advances on ω, e.g. [16,6,20] and most recently $\omega < 2.3728639$ was shown due to Le Gall [8].

Method	word-RAM complexity	Notes
General dense	$\mathcal{O}\left(U^{\omega}\right)$	
Lingas	$\tilde{\mathcal{O}}\left(U^2 Z^{\omega/2-1}\right)$	Requires boolean matrices.
Iwen-Spencer, Le Gall	$\mathcal{O}\left(U^{2+\varepsilon}\right)$	Requires $\mathcal{O}\left(n^{0.3}\right)$ nonzeros per column.
Williams-Yu, Pagh	$\tilde{\mathcal{O}}\left(U^2 + UZ\right)$	
Van Gucht et al.	$\tilde{\mathcal{O}}\left(N\sqrt{Z} + Z + N\right)$	
This paper, Thm. 2	$\tilde{\mathcal{O}}\left(UZ^{(\omega-1)/2} + N\right)$	Requires balanced rows and columns.
This paper, Thm. 3	$\tilde{\mathcal{O}}\left(U^2(Z/U)^{\omega-2} + N\right)$	
Method	I/O complexity	Notes
General dense	$\tilde{\mathcal{O}}\left(U^{\omega}/(M^{\omega/2-1}B)\right)$	
Pagh-Stöckel	$\tilde{\mathcal{O}}\left(N\sqrt{Z}/(B\sqrt{M})\right)$	Elements from semirings.
This paper, Thm. 2	$\tilde{\mathcal{O}}\left(UZ^{\frac{\omega-1}{2}}/(M^{\omega/2-1}B) + Z/B + N/B\right)$	Requires balanced rows and columns.
This paper, Thm. 3	$\tilde{\mathcal{O}}\left(U^2(Z/U)^{\omega-2}/(M^{\omega/2-1}B) + U^2/B\right)$	

Table 1. Comparison of matrix multiplication algorithms of two $U \times U$ in the RAM and I/O model. N denotes the number of nonzeros in the input matrices, Z the number of nonzeros in the output matrix and ω is the currently lowest matrix multiplication exponent.

The closest algorithm in spirit to our general algorithm of Theorem 3 is due to Williams and Yu [22]. They recursively, using time $\tilde{\mathcal{O}}\left(U^2\right)$, with high probability compute all positions of nonzero output entries. After this they compute each output value in time $\mathcal{O}(U)$ for a total number of $\tilde{\mathcal{O}}\left(U^2 + UZ\right)$ operations. This matches the exact case of Pagh's compressed matrix multiplication result [13], which is significantly more involved but also gives a stronger guarantee: given a parameter b, using time $\tilde{\mathcal{O}}\left(U^2 + Ub\right)$, it gives the exact answer with high probability if $Z \leq b$, otherwise it outputs a matrix where each entry is close to AC in terms of the Frobenius norm. Our general algorithm of Theorem 3 improves upon both when $Z \gg U$.

In the case of sparse *input* matrices, Yuster and Zwick [23] showed how to exploit sparseness of input using an elegant and simple partitioning method. Their result was extended to be both input and output sensitive by Amossen and Pagh [4], leading to a time bound of $\tilde{\mathcal{O}}\left(N^{2/3}Z^{2/3} + N^{0.862}Z^{0.546}\right)$ based on current ω. In our (non-input sensitive) case where $N = U^2$ we are strictly better for all $U > 1$ and $Z > U$. We note that the algorithm of Amossen and Pagh is presented for boolean input and claimed to work over any ring, however both the result of Yuster-Zwick and Amossen-Pagh do not support cancellations, i.e., their bounds are in terms of the number of vector pairs of the input that have nonzero elementary products. The above, as well as Pagh [13] and Williams-Yu [22] were improved recently by Van Gucht et al. [19] to be $\tilde{\mathcal{O}}\left(Z + N\sqrt{Z} + N\right)$ operations, stated in the binary case but claimed to work over any field. Compared to this we use $\tilde{\mathcal{O}}\left(U^2(Z/U)^{\omega-2}\right)$ operations by Theorem 3, which for $N > U^{4-\omega}Z^{\omega-5/2}$ improves Van Gucht et al. in the general case. For dense inputs, $N = U^2$, this threshold on N simplifies to $Z > U^{(\omega-2)/(\omega-5/2)}$ which holds for all positive integers $Z \geq 1$ and $U > 1$.

In the balanced case, Iwen and Spencer [9] showed that if every column of the output matrix has $\mathcal{O}(U^{0.29462})$ nonzero entries, then the matrix product can be computed using time $\mathcal{O}(U^{2+\varepsilon})$ for constant $\varepsilon > 0$. Recently due to Le Gall [8] this result now holds for output matrices with columns with at most $\mathcal{O}(U^{0.3})$ nonzeros. In this case our balanced matrix multiplication algorithm of Theorem 2 uses time $\tilde{\mathcal{O}}\left(U^{2.19}\right)$ (for $\omega = 2.3728639$), which is asymptotically worse, but our method applies to general balanced matrices. Compared to Van Gucht et al., in the balanced case, we improve upon their operation count when $N > UZ^{\omega/2-1}$. When $N = U^2$ this always holds since $Z \leq U^2$.

For boolean input matrices, the output sensitive algorithm of Lingas [11] runs in time $\tilde{\mathcal{O}}\left(U^2 Z^{\omega/2-1}\right)$, which we improve on for $1 \leq Z < U^2$ by a relative factor of $Z^{1-\omega/2}$ and match when $Z = U^2$. Additionally our algorithm works over any field. Lingas however shows a partial derandomization that achieves the same bound using only $\mathcal{O}(\log^2 U)$ random bits, which is a direction not pursued in this paper. The general case, i.e., dense input and output without fast matrix multiplication allowed, multiplying two boolean matrices has time complexity $\mathcal{O}\left(U^3 \text{poly}(\log\log)/\log^4 U\right)$ due to Yu [21], which is an improvement to Chans time $\mathcal{O}\left(U^3\left(\log\log U\right)^3/\log^3 U\right)$ algorithm [5]. Both are essentially improvements on the four russians trick: 1) divide the matrix into small $t \times t$ blocks 2) pre-compute results of all $t \times t$ blocks and store them in a dictionary. Typically $t = \mathcal{O}(\log U)$ and the gain is that there are $(U/t)^2 = U^2/\log^2 U$ blocks instead of U^2 cells.

In terms of I/O complexity, an optimal general algorithm was presented by Hong and Kung [10] that uses $\mathcal{O}(U^3/(B\sqrt{M})$ I/Os using a blocking argument. An equivalent blocking argument gives in our case where we are allowed to exploit cancellations an I/O complexity of $\mathcal{O}(U^\omega/(M^{\omega/2-1}B))$. This is the complexity of the black box we will invoke on dense subproblems (also stated in Fact 1). For sparse matrix multiplication in the I/O model over semirings, Pagh and Stöckel [14] showed a $\tilde{\mathcal{O}}(N\sqrt{Z}/(B\sqrt{M})$ algorithm and a matching lower bound. To the knowledge of the authors the algorithm presented in this paper is the first algorithm that exploits cancellations and is output sensitive. Our algorithm is asymptotically better than the general dense for all $1 \leq Z < U^2$ and we match their complexity for $Z = U^2$. Our new algorithm is asymptotically faster than Pagh-Stöckel precisely when $Z > \left(U^{4-\omega}M^{3/2-\omega/2}/N\right)^{1/(5/2-\omega)}$. To simplify, consider the case of dense input, i.e., $N = U^2$, then our new algorithm is better unless M much is larger than Z, which is typically assumed to not be the case.

Comparison summary. The general algorithm of Theorem 3 works over any field and can exploit cancellations (and supports cancellation of terms). In the RAM model it is never worse than $\mathcal{O}(U^\omega)$ and in the case of dense input, which is the case that suits our algorithms best, we improve upon state of the art (Pagh [13] and Williams-Yu [22]) when $Z \gg U$. For arbitrary input density we improve upon state of the art (Van Gucht et al [19] when $N \gg U^{4-\omega}Z^{\omega-5/2}$. In the I/O model our algorithm is the first output sensitive algorithm that exploits cancellations and it outperforms the known semiring algorithms for almost the entire parameter space.

2 The row balanced case

Lemma 1. *Let $A \in \mathbb{F}^{U' \times U}$ and $C \in \mathbb{F}^{U \times U}$ be two matrices with $U' \leq U$. Assume each row of the $U' \times U$ product matrix AC has at most $d/5$ non-zero entries. For any natural constant c there is a data structure where*

(a) initializing the data structure takes time $\tilde{\mathcal{O}}\left(\mathtt{nnz}(C) + F_{\mathrm{RAM}}(U', U, d)\right)$ on the RAM and $\tilde{\mathcal{O}}\left(\mathrm{sort}_{M,B}(\mathtt{nnz}(C)) + F_{\mathrm{I/O}}(U', U, d)\right)$ I/Os
(b) the silent failure probability of initialization is at most U^{-c}
(c) the data structure can answer queries for entries of (AC) in time $O(\log(U^{2+c}))$
(d) the data structure can report all (non-zero) entries with $O(U' sort(kU))$ I/Os

Proof. We chose a parameter k (number of repetitions) by $k = 6\ln(U^{2+c})$. The data structure consists of k independent compressed versions of AC, $G_i \in \mathbb{F}^{U' \times d}$. We choose k independent random (hash) functions $h_l \colon [U] \to [d]$ where each $h_l(j)$ is chosen independently and uniformly from $[d]$. Each h_l is stored as a table in the data structure. We compute k compressed matrices $G_l = ACP_{h_l}$. See also Algorithm 1. This computation can be achieved by scanning C and changing all column indexes by applying the hash function. The resulting matrix C' with d columns is multiplied by A using a fast dense method described in Fact 1.

There are $U'U \leq U^2$ different queries that are answered using Algorithm 2. A particular $z_l = (G_l)_{i,j}$ is correct if no (other) non-zero entry of AC has been hashed to the same position of G_l. Because we are hashing only within rows, there are at most $d/5$ such non-zero entries. For a random hash function the probability that a particular one does so is $1/d$, so the probability that z_l is correct is at least

$$(1 - 1/d)^{d/5} \geq (1/4)^{1/5} \geq 3/4\,.$$

Here, the first inequality stems from $(1 - 1/d)^d$ approaching e^{-1} from below and being $\geq 1/4$ for $d \geq 2$, and the second by $4^4 > 3^5$. Now, the probability of the median (or a majority vote) being false is at most that of the following Bernoulli experiment: If heads (errors) turn up with probability 1/4 and k trials are performed, at least $k/2$ heads show up. By a Chernoff-Hoeffding inequality [7, Theorem 1.1, page 6] (equation (1.6) with $\epsilon = 1$) this failure probability is at most $\exp(-\frac{1}{3}\frac{k}{2}) = \exp(-k/6) \leq U^{-2-c}$. Hence a union bound over all U^2 possible queries shows that a randomly chosen set of compression matrices leads to a correct data structure with probability at least $1 - U^{-c}$.

To report all entries of AC we proceed row by row. We can assume that all G_l are stored additionally row wise, i.e., first the first row of G_1, then the first row of G_2 and so on, after that the second row of each G_l and so on. This reordering of the data structure does not change the asymptotic running time of the initialization phase. For row i we copy each entry $g_{i,j'} \in G_l$ as many times as there is an $j \in U$ with $h_l(j) = j'$ and annotate the copy with the index j. This can be achieved by scanning if the graph of the hash function is sorted by target element. Then all these copies are sorted mainly by the annotated index and secondarily by their value. With this, for each j, the entries of the G_l corresponding to (i, j) are sorted together and a median or majority vote can easily be computed. If desired, the zero entries can be filtered out. The I/O complexity is obviously as claimed. ■

```
Input: Matrices $A \in \mathbb{F}^{U' \times U}$ and $C \in \mathbb{F}^{U \times U}$
Output: $k$, hash functions $h_1, \dots, h_k \colon [U] \to [d]$,
        compressed matrices $G_i \in \mathbb{F}^{U' \times d}$
Set $k = k = 6 \ln(U^2/\delta)$
for $l \leftarrow 1$ to $k$ do
    Create random function $h_l \colon [U] \to [d]$
    Create $C' \in \mathbb{F}^{U \times d}$, initialized to 0
    foreach $(i, j)$ with $C_{i,j} \neq 0$ do
        $C'_{i,h_l(j)} += C_{i,j}$ ;                    /* $C' = CP_{h_l}$ */
    Fast Multiply $G_i = AC'$
```

Algorithm 1: Initialization Balanced Rows

```
Input: $k$, Hash Functions $h_1, \dots, h_k \colon U \to d$, $i, j$
Output: $(AC)_{i,j}$
for $l \leftarrow 1$ to $k$ do
    Set $z_l = (G_l)_{i,h_l(j)}$
Report Median of $z_l$
```

Algorithm 2: Query Balanced Rows

3 Subdividing into balanced parts

To make use of Lemma 1 for general matrices, it is important that we can estimate the number of nonzero elements in the rows of the output:

Fact 4 ([14], Lemma 1) *Let A and C be $U \times U$ matrices with entries of field $\mathbb{F}$, $N = \mathtt{nnz}(A) + \mathtt{nnz}(C)$ and let $0 < \varepsilon, \delta \leq 1$.*
We can compute estimates $z_1, \dots, z_U$ using $\mathcal{O}(\varepsilon^{-3} N \log(U/\delta) \log U)$ RAM operations and $\tilde{\mathcal{O}}(\varepsilon^{-3} N/B)$ I/Os, such that with probability at least $1 - \delta$ it holds that $(1 - \varepsilon)\, \mathtt{nnz}([AC]_{k*}) \leq z_k \leq (1 + \varepsilon)\, \mathtt{nnz}([AC]_{k*})$ for all $1 \leq k \leq U$.

With this we can get the following data structure for potentially unbalanced output matrices.

Lemma 2. *Let $A, C \in \mathbb{F}^{U \times U}$ be two square matrices, and let $N = \mathtt{nnz}(A) + \mathtt{nnz}(C)$, $Z = \mathtt{nnz}(AC)$. For any natural constant c there is a data structure where*

(a) initializing the data structure takes time $\tilde{\mathcal{O}}\left(F_{\mathrm{RAM}}(U, Z/U, U) + N\right) = \tilde{\mathcal{O}}\left(U^2(Z/U)^{\omega-2} + N\right)$ on the RAM and $\tilde{\mathcal{O}}\left(F_{\mathrm{I/O}}(U, Z/U, U)\right)$ I/Os
(b) the silent failure probability of initialization is at most U^{-c}
(c) the data structure can answer queries for entries of (AC) in time $O(\log(U^2/\delta))$
(d) the data structure can report all (non-zero) entries with $O(U\mathit{sort}(kU) + \mathit{sort}(U^2))$ I/Os

Proof. We use Fact 4 to partition the output matrix into blocks of rows, for each of which we use the data structure of Lemma 1.

Observe that permuting the rows of AC is the same as permuting the rows of A. We use Fact 4 with $\varepsilon = .3$ and $\delta = U^{1+c}/2$ to estimate the number of non-zeros in row i of AC as z_i. We group the rows accordingly, row i belongs to group l if $U \cdot 2^{-l-1} < 1.3 z_i \leq U \cdot 2^{-l}$. We create a table that states for each original row its group and position in the group (row in the smaller matrix) Hence with overall probability at least $1 - U^{-c}/2$, each group l contains x_l rows where the number of non-zeros is between $U \cdot 2^{-l-2}$ and $U \cdot 2^{-l}$. At the cost of sorting A we make these at most $\log U$ matrices $A_l \in \mathbb{F}^{x_l \times U}$ explicit in sparse format. We create the overall data structure from smaller ones, one for each $R_l = A_l C$ using Lemma 1 with $c' = c + 1$. These data structures have an individual success probability of at least $1 - U^{-c'}$, and because there are at most $\log U$ such data structures, by a union bound, an overall success probability of $1 - U^{-c}$.

The overall creation cost hinges on the cost for multiplying the smaller matrices, i.e., $\sum_l F_{\mathrm{RAM}}(U, x_l, 5U2^{-l})$. To bound this, we estimate x_l by the upper bound $x_l \leq 4 \cdot Z \cdot 2^l/U$ which stems from $U \cdot 2^{-l-2} \cdot x_l \leq Z$. For $l > \log(Z/U)$ this bound is bigger than U and we should estimate such x_l as U. This implies that this part of the sum forms a geometric series, and asymptotically it is sufficient to consider

$$\sum_{l=1}^{\log(Z/U)} F_{\mathrm{RAM}}(U, 4Z \cdot 2^l/U, 5U2^{-l})\,.$$

For this sum of fast matrix multiply running times we realize that the product of the dimensions is always $20UZ$, and hence each term is at most that of the most unbalanced case. Hence we can estimate the overall running time on the RAM as $F_{\mathrm{RAM}}(U, 4Z/U, 5U)\log(U)$. The same argument works for the overall number of I/Os. By observing that the required work on A takes $\tilde{\mathcal{O}}\left(\mathrm{sort}(U^2)\right) = \tilde{\mathcal{O}}\left(F_{\mathrm{I/O}}(U, Z/U, U)\right)$ we get that the I/O performance of initializing is as stated.

To report all output entries, we use the reporting possibility of the smaller data structures. That output has to be annotated with the original row number and sorted accordingly. This yields the claimed I/O-bound. ■

Proof (of Theorem 3). We use the data structure of Lemma 2 with $c = 2$. To produce the output RAM efficiently, we perform U^2 queries to the data structure which is $\tilde{\mathcal{O}}\left(U^2\right) = \tilde{\mathcal{O}}\left(U^2(Z/U)^{\omega-2}\right)$. To produce the output I/O efficiently, we use the procedure of item (d). Because $O(U\mathrm{sort}(kU) + \mathit{sort}(U^2)) = \tilde{\mathcal{O}}\left(U^2/B\right) = \tilde{\mathcal{O}}\left(F_{\mathrm{I/O}}(U, Z/U, U)\right)$ by Fact 1 we also get the claimed I/O performance. ■

4 The balanced case

If the matrices are square and have d as an upper bounds on the number of nonzero entries both in columns and row we can hash in both columns and rows. This is beneficial because we achieve a more balanced setting of the dimensions when calling the fast matrix multiplication algorithm.

4.1 A data structure for balanced output

Lemma 3. *Let $A \in \mathbb{F}^{U' \times U}$ and $C \in \mathbb{F}^{U \times U}$ be two matrices with $U' \leq U$. Let $Z = \texttt{nnz}(AC)$ and $N = \texttt{nnz}(A) + \texttt{nnz}(C)$. Assume each row and each column of the $U' \times U$ product matrix AC has at most $d/5$ entries. Assume further that $Z > Ud/O(1)$. For any constant c there is a data structure where*

(a) initializing the data structure takes time $\tilde{\mathcal{O}}\left(N + F_{\mathrm{RAM}}(\sqrt{Ud}, U, \sqrt{Ud})\right) = \tilde{\mathcal{O}}_\delta\left(UZ^{\frac{\omega-1}{2}} + N\right)$ on the RAM and $\tilde{\mathcal{O}}\left(N/B + F_{\mathrm{I/O}}(\sqrt{Ud}, U, \sqrt{Ud})\right)$ I/Os

(b) the silent failure probability of initialization is at most $3U^{-c}$

(c) the data structure can answer queries for entries of (AC) in time $O(\log(U^{2+c}))$

(d) A batched query for up to $2Z$ entries can be performed with $\tilde{\mathcal{O}}\left((Z)/B\right)$ I/Os.

Proof. (Sketch) The construction is as in Lemma 1 (including $k = 6\ln(U^{2+c})$), only that both A and C are compressed using hash functions. The compression is only down to $\sqrt{Ud} > d$ columns and rows. Additionally to collisions within a row or within a column, now also collision of arbitrary elements are possible. The failure probability of all three cases can be derived just as in Lemma 1. For the chosen parameters the sum of the three failure probabilities yields the claim.

To perform a batched query for $2Z$ entries, the following I/O-efficient algorithm can be used. Observe that all described sorting steps are on the queries or on compressed matrices and hence operate on $O(Z)$ elements. We first consider the compressed matrices individually. With a first sorting step (by row), the queries are annotated by the hashed row. In a second sorting step they are annotated by column. Now all annotated queries are sorted into the compressed matrix. For each query, the corresponding entry of the compressed matrix is extracted and annotated with the query. When this is done for all compressed matrices, the annotated entries are sorted by the index of query and a median or majority vote is performed. ■

4.2 Creating the output as sparse matrix

Unlike in the other settings, here the time to query for all possible entries of the output matrix might dominate the overall running time. Hence, in this section we propose an additional algorithm to efficiently extract the entries of AC that are nonzero, relying on the bulk query possibility of the data structure.

Lemma 4. *Let $A, C \in \mathbb{F}^{U \times U}$ be two matrices. Let $Z = \texttt{nnz}(AC)$ and $N = \texttt{nnz}(A) + \texttt{nnz}(C)$. Assume each row and each column of the $U \times U$ product matrix AC has at most $d/5$ entries. Assume further that $Z > Ud/O(1)$. For any natural constant c there is an algorithm that*

(a) takes time $\tilde{\mathcal{O}}\left(Z + N + F_{\mathrm{RAM}}(\sqrt{Ud}, U, \sqrt{Ud})\right)$ on the RAM
and $\tilde{\mathcal{O}}\left(\mathrm{sort}_{M,B}(N + Z) + F_{\mathrm{I/O}}(\sqrt{Ud}, U, \sqrt{Ud})\right)$ I/Os

(b) the silent failure probability is at most $5U^{-c}$

Proof. For each column of AC, we create a perfectly balanced binary tree where the leafs are the individual entries of the column. An internal node of the tree is annotated with the information whether there is a nonzero leaf in the subtree. A simultaneous BFS traversal of all these trees (one for each column) allows to identify all positions of nonzero entries. The number of elements on the "wavefront" of the BFS algorithm is at most Z. To advance the wavefront by one level, a batched query to at most $2Z$ positions of the tree is sufficient. Finally, the matrix AC itself is stored in a data structure as in Lemma 3 with failure probability $3U^{-c}$.

Instead of annotating the tree nodes directly, we compute several random dot products with the leaf-values. More precisely, for each leaf we choose a coefficient uniformly from $\{0, 1\} \subset \mathbb{F}$. Now, if one leaf has value $r \neq 0$, then with probability at least $1/2$ the dot product is non-zero: Assume all other coefficients are fixed, leading to a certain partial sum s; now it is impossible that both s and $s + 1 \cdot r$ are zero. If we have $(c + 3)\log U$ many such coefficients, the failure probability is at most $1/U^{c+3}$.

Observe that for one level of the tree and one random choice, the dot products form a matrix that can be computed as HAC for a $\{0, 1\}$-matrix H, and HA can be computed by sorting A. Observe further that the number of nonzero entries of the columns and rows of HAC are such that each of these matrices can be encoded using the data structure of Lemma 3 with failure probability $3U^{-c-1}$.

If there is no failure, the algorithm, using the batch queries, achieves the claimed running times and I/Os.

The probability that there is an encoding error in any of the matrices encoding the trees is, for sufficiently big U, at most U^{-c} because there are only $\log(U)(c + 3)\log(U) < U/3$ such matrices. The probability of a single tree node being encoded incorrectly is at most $3/U^{c+3}$. Because there are $U^2 \log U < U^3/3$ tree nodes in total, all trees are correct with probability at least $1 - U^{-c}$. Hence the overall failure probability, including that the data structure for the entries of AC failed, is hence as claimed. ■

Proof (of Theorem 2). We invoke Lemma 4 with $c = 2$. Combining the statements about the RAM running time with Fact 1 and the calculation $ZU\sqrt{Z}^{\omega-3} = UZ^{\frac{\omega-1}{2}}$ gives the claimed RAM running time. Doing the same for the I/O bound and using that $\tilde{\mathcal{O}}\left(\mathrm{sort}_{M,B}(N + Z)\right) = \tilde{\mathcal{O}}\left(N/B + Z/B\right)$ gives the claimed I/O performance. ■

References

1. A. Aggarwal and S. Vitter, Jeffrey. The input/output complexity of sorting and related problems. *Commun. ACM*, 31(9):1116–1127, Sept. 1988.
2. N. Alon, R. Yuster, and U. Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995.
3. N. Alon, R. Yuster, and U. Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997.
4. R. R. Amossen and R. Pagh. Faster join-projects and sparse matrix multiplications. In *International Conference on Database Theory*, ICDT '09, 2009.
5. T. M. Chan. Speeding up the four russians algorithm by about one more logarithmic factor. In *SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 212–217, 2015.
6. D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3):251 – 280, 1990.
7. D. P. Dubhashi and A. Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms.* Cambridge University Press, 2009.
8. F. L. Gall. Powers of tensors and fast matrix multiplication. *arXiv preprint arXiv:1401.7714*, 2014.
9. M. Iwen and C. Spencer. A note on compressed sensing and the complexity of matrix multiplication. *Information Processing Letters*, 109(10):468 – 471, 2009.
10. H. Jia-Wei and H. T. Kung. I/O complexity: The red-blue pebble game. In *ACM Symposium on Theory of Computing*, STOC '81, 1981.
11. A. Lingas. A fast output-sensitive algorithm for boolean matrix multiplication. In *Algorithms - ESA 2009*, pages 408–419. Springer-Verlag, 2009.
12. K. Mulmuley, U. Vazirani, and V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, 1987.
13. R. Pagh. Compressed matrix multiplication. *ACM Trans. Comput. Theory*, 5(3):9:1–9:17, Aug. 2013.
14. R. Pagh and M. Stöckel. The input/output complexity of sparse matrix multiplication. In *Algorithms - ESA 2014*. Springer-Verlag, 2014.
15. M. O. Rabin and V. V. Vazirani. Maximum matchings in general graphs through randomization. *J. Algorithms*, 10(4):557–567, Dec. 1989.
16. A. J. Stothers. *On the complexity of matrix multiplication.* PhD thesis, The University of Edinburgh, 2010.
17. V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4):354–356, 1969.
18. S. van Dongen. *Graph Clustering by Flow Simulation.* PhD thesis, University of Utrecht, 2000.
19. D. Van Gucht, R. Williams, D. P. Woodruff, and Q. Zhang. The communication complexity of distributed set-joins with applications to matrix multiplication. In *Proceedings of the 34th ACM Symposium on Principles of Database Systems*, PODS '15, pages 199–212, New York, NY, USA, 2015. ACM.
20. V. V. Williams. Multiplying matrices faster than coppersmith-winograd. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, 2012.
21. H. Yu. An improved combinatorial algorithm for boolean matrix multiplication. *ICALP*, 2015.
22. H. Yu and R. Williams. *Finding orthogonal vectors in discrete structures*, chapter 135, pages 1867–1877. SIAM, 2014.
23. R. Yuster and U. Zwick. Fast sparse matrix multiplication. *ACM Trans. Algorithms*, 1(1):2–13, July 2005.