Cache Oblivious Sparse Matrix Multiplication

Matteo Dusefante(✉) and Riko Jacob

IT University of Copenhagen, Copenhagen, Denmark
{madu,rikj}@itu.dk

Abstract. We study the problem of sparse matrix multiplication in the Random Access Machine and in the Ideal Cache-Oblivious model. We present a simple algorithm that exploits randomization to compute the product of two sparse matrices with elements over an arbitrary field. Let $A \in \mathbb{F}^{n\times n}$ and $C \in \mathbb{F}^{n\times n}$ be matrices with h nonzero entries in total from a field $\mathbb{F}$. In the RAM model, we are able to compute all the k nonzero entries of the product matrix $AC \in \mathbb{F}^{n\times n}$ using $\tilde{\mathcal{O}}(h + kn)$ time and $\mathcal{O}(h)$ space, where the notation $\tilde{\mathcal{O}}(\cdot)$ suppresses logarithmic factors. In the External Memory model, we are able to compute cache obliviously all the k nonzero entries of the product matrix $AC \in \mathbb{F}^{n\times n}$ using $\tilde{\mathcal{O}}(h/B + kn/B)$ I/Os and $\mathcal{O}(h)$ space. In the Parallel External Memory model, we are able to compute all the k nonzero entries of the product matrix $AC \in \mathbb{F}^{n\times n}$ using $\tilde{\mathcal{O}}(h/PB + kn/PB)$ time and $\mathcal{O}(h)$ space, which makes the analysis in the External Memory model a special case of Parallel External Memory for $P = 1$. The guarantees are given in terms of the size of the field and by bounding the size of $\mathbb{F}$ as $|\mathbb{F}| > kn \log(n^2/k)$ we guarantee an error probability of at most $1/n$ for computing the matrix product.

1 Introduction

Matrix multiplication is a fundamental operation in computer science and mathematics. Despite the effort, the computational complexity is still not settled, and it is not clear whether $\mathcal{O}(n^2)$ operations are sufficient to multiply two dense $n \times n$ matrices.

Given matrices $A \in \mathbb{F}^{n\times n}$ and $C \in \mathbb{F}^{n\times n}$, we define h as the number of nonzero entries in the input, i.e. $h = \mathtt{nnz}(A) + \mathtt{nnz}(C)$, k as the number of nonzero entries in the output, i.e. $k = \mathtt{nnz}(AC)$, where $\mathtt{nnz}(A)$ denotes the number of nonzero entries in matrix A. Let $A_{i,j}$ be the value of the entry in the matrix A with coordinates (i, j). We denote with $A_{*,j}$ and $C_{i,*}$ the j-th column of A and the i-th row of C respectively. Note that we can easily detect, by scanning the input matrices, null vectors. Hence, without loss of generality, we consider only the case where $h \geq 2n$ and the rows of A (resp. columns of C) are not n-dimensional null vectors. Observe that, in contrast cancellations

M. A. Bender et al. (Eds.): LATIN 2018, LNCS 10807, pp. 437–447, 2018.
https://doi.org/10.1007/978-3-319-77404-6_32

can lead to situations where $k \leq n$.[1] The algorithms presented in this paper can compute the product of matrices of arbitrary size and the bounds can be straightforwardly extended to rectangular matrices. However, for the ease of exposition, we restrict our analysis to square matrices. We denote with *column major layout* the lexicographic order where the entries of A are sorted by column first, and by row index within a column. Analogously, we denote with *row major layout* the order where the entries of A are sorted row-wise first, and column-wise within a row. Note that a row major layout can be obtained from column major layout by transposing A, and vice versa. Throughout this paper we use $f(n) = \tilde{\mathcal{O}}(g(n))$ as shorthand for $f(n) = \mathcal{O}(g(n)\log^c g(n))$ for some constant c. The memory hierarchy we refer to is modeled by the I/O model by Aggarwal and Vitter [1], the Ideal Cache-Oblivious model by Frigo et al. [2] and the Parallel External Memory model by Arge et al. [3]. We denote with M the number of elements that can fit into internal memory, B the number of elements that can be transferred in a single block and P as the number of processors. The parameters M, and B are referred as the *memory size* and *block size* respectively. The Ideal Cache-Oblivious model resembles the I/O model except for memory and block size are unknown to the algorithm. Unless otherwise stated, it holds $1 \leq B \leq M \ll h$. Note that, for our parallel algorithm, we consider a cache aware model since concurrency is nontrivial in external memory models whenever the block size B is unknown [4].

1.1 Contributions

We study the problem of matrix multiplication in the Random Access Machine and in external memory over an arbitrary field $(\mathbb{F}, +, \cdot)$, where $(+, \cdot)$ are atomic operations over elements of $\mathbb{F}$ in the computational models. We present a randomized algorithm for multiplying matrices $A \in \mathbb{F}^{n\times n}$, $C \in \mathbb{F}^{n\times n}$ that, after $\mathcal{O}(h)$ time for preprocessing using deterministic $\mathcal{O}(h)$ space, computes, using $\mathcal{O}(nk\log(n^2/k))$ time all the k nonzero entries of the product matrix $AC \in \mathbb{F}^{n\times n}$, with high probability. We present a cache oblivious algorithm for multiplying matrices $A \in \mathbb{F}^{n\times n}$ and $C \in \mathbb{F}^{n\times n}$. After $\mathcal{O}((h/B)\log_{M/B} h/B)$ I/Os for preprocessing, using deterministic $\mathcal{O}(h)$ space, we are able to compute, using $\mathcal{O}((n/B)k\log(n^2/k))$ I/Os, all the k nonzero entries of the product matrix $AC \in \mathbb{F}^{n\times n}$, with high probability, under a tall cache assumption, i.e. $M \geq B^{1+\varepsilon}$ for some $\varepsilon > 0$. Similarly, in the Parallel External Memory model, we present an algorithm for multiplying matrices $A \in \mathbb{F}^{n\times n}$, $C \in \mathbb{F}^{n\times n}$ that, after $\mathcal{O}((h/PB)\log_d(h/B))$ I/Os for preprocessing, with $d = \max\{2, \min\{M/B, H/PB\}\}$, using deterministic $\mathcal{O}(h)$ space, computes, using $\mathcal{O}((n/PB + \log P)k\log(n^2/k))$ I/Os, all the k nonzero entries of the product matrix $AC \in \mathbb{F}^{n\times n}$, with high probability. Note that, for the External Memory model and the Parallel External Memory model, preprocessing is dominated by $\mathcal{O}(\text{sort}(h))$ I/Os which stems from layout transposition. Although faster

[1] A cancellation occurs when $(AC)_{i,j} = 0$ while elementary products do not evaluate to zero, i.e. $A_{i,\kappa} \cdot C_{\kappa,j} \neq 0$, for some $\kappa \in [n]$.

algorithms for transposing sparse matrices exist, for the ease of exposition, we consider $\mathcal{O}(\text{sort}(h))$ I/Os as an upper bound for preprocessing which weakens the bounds only in the parameters of the logarithmic factors. We give rigorous guarantees on the probability of detecting all the nonzero entries of the output matrix by studying how the process of generating random elements from the field affects the process of locating entries. The guarantees are given in terms of the size of the field. If the algorithms generate random variables from an arbitrary field $\mathbb{F}$ then we are able to detect a nonzero entry in the matrix with probability at least $1 - 2/|\mathbb{F}| + 1/|\mathbb{F}|^2$. On the other hand, given an arbitrary field $\mathbb{F}$, if the random variables are generated from $\mathbb{F}^* = \mathbb{F}\backslash\{0\}$ then we detect a nonzero entry with probability at least $1 - 1/|\mathbb{F}^*|$. By bounding the size of $\mathbb{F}$ as $|\mathbb{F}| \geq ckn \log(n^2/k)$, for some constant c, we guarantee an error probability of at most $1/n$. Conversely, if $|\mathbb{F}| < ckn \log(n^2/k)$ we can improve the error probability by repeating the random process an arbitrary number of time, say $\log n$ times, thus sacrificing a $\log n$ factor in space and time with the effect of decreasing the error probability by a factor of n.

1.2 Related Work

Given two $n \times n$ matrices A and C, the matrix product AC can be trivially computed with $\mathcal{O}(n^3)$ arithmetic operations. Strassen [5], in 1969, provided a recursive algorithm able to multiply two matrices in $\mathcal{O}(n^\omega)$ with $\omega = 2.8073549$ by exploiting cancellations. The last known result is due to Le Gall [6] who holds the current record of $\omega < 2.3728639$. Yuster and Zwick [7] presented an algorithm that multiplies two $n \times n$ matrices over a ring using $\tilde{\mathcal{O}}(h^{0.7}n^{1.2} + n^{2+o(1)})$ arithmetic operations. Iwen and Spencer [8] proved that if each column of AC contains at most $n^{0.29462}$ nonzero entries, then A and C can be multiplied with $\mathcal{O}(n^{2+\varepsilon})$ operations. Our algorithms improve over Yuster and Zwick [7] as well as Iwen and Spencer [8] when $k < n$ and $h \ll n^2$. In addition, we do not require a balanced assumption of the output matrix, e.g. the number of nonzero entries per column, as in [8]. Amossen and Pagh [9] provided a sparse, output-sensitive matrix multiplication that incurs in $\tilde{\mathcal{O}}(h^{2/3}k^{2/3} + h^{0.862}k^{0.408})$ operations and $\tilde{\mathcal{O}}(h\sqrt{k}/(BM^{1/8}))$ I/Os. Lingas [10] presented an output-sensitive, randomized algorithm for computing the product of two $n \times n$ boolean matrices using $\tilde{\mathcal{O}}(n^2k^{\omega/2-1})$ operations. Compared to Amossen and Pagh and Lingas, we allow cancellations of terms and we do not restrict our analysis to boolean matrices. In addition, Amossen and Pagh's I/O algorithm is not cache oblivious, i.e. it requires knowledge of of the memory size. Pagh [11] presented a randomized algorithm that computes an unbiased estimator of AC in time $\tilde{\mathcal{O}}(h + nk)$, with guarantees given in terms of the Frobenius norm. Pagh's compressed algorithm achieves the same time bounds as our algorithms. However, we improve over space complexity whenever $k < h/\log n$, i.e. when cancellations account for a $1/\log n$ factor compared to the number of input entries. Besides this, Pagh's result is algorithmically more involved, since it makes use of hash functions and Fast Fourier Transform. Williams and Yu [12] provided an output-sensitive, randomized algorithm for matrix multiplication with elements over any field, that,

after $\mathcal{O}(n^2)$ preprocessing, computes each nonzero entry of the matrix product in $\tilde{\mathcal{O}}(n)$ time. We extend their techniques to the sparse input case and we improve whenever $h \ll n^2$, i.e. when the input matrices are sparse, both in time and space complexity. Jacob and Stöckel [13] presented a Monte Carlo algorithm that uses $\tilde{\mathcal{O}}(n^2(k/n)^{\omega-2} + h)$ operations and, with high probability, outputs the nonzero elements of the matrix product. In addition, they state an I/O complexity of $\tilde{\mathcal{O}}(n^2(k/n)^{\omega-2}/(M^{\omega/2-1}B) + n^2/B)$. Their analysis is focused specifically on dense matrices and we improve over their results, both in time and I/O complexity, whenever k is asymptotically smaller than n in the general case while we achieve the same bounds when $k = n$. In addition, we do not require knowledge of the memory size as opposed to [13]. Van Gucht et al. [14] presented a randomized algorithm for multiplying two boolean matrices in $\tilde{\mathcal{O}}(k + h\sqrt{k} + h)$ time. In contrast to their results, our algorithms are not restricted to the boolean case and we are able to compute the product of matrices from an arbitrary field. Matrix multiplication has been widely studied in external memory as well. In a restricted setting, i.e. the semiring model, Hong and Kung [15] provided a lower bound of $\Omega(n^3/\sqrt{M})$ I/Os for multiplying two $n \times n$ matrices using n^3 operations and a memory of size M. Frigo et al. [2] provided a cache oblivious algorithm for multiplying two $n \times n$ matrices cache obliviously using $\mathcal{O}(n^3)$ operations and $\mathcal{O}(n^2/B + n^3/(B\sqrt{M}))$ I/Os. In the I/O model, Pagh and Stöckel [16] provided a randomized, I/O optimal algorithm for matrix multiplication that incurs in $\tilde{\mathcal{O}}((h/B)\min\{\sqrt{k}/\sqrt{M}, h/M\})$ I/Os. However, their algorithm does not allow cancellation of terms and it requires knowledge of the memory size in order to partition the input matrices. In relation to this, we require no knowledge on the size of M and our algorithm run cache obliviously. To the knowledge of the authors, there are no previously known cache oblivious algorithms for sparse matrix multiplication that exploit cancellations.

2 Algorithms

Williams and Yu [12] provided a simple output-sensitive algorithm for matrix multiplication. The intuition behind [12] is that nonzero entries in a submatrix of AC with indices in $[i_1, i_2] \times [j_1, j_2]$ can be detected by testing whether $\langle a, c\rangle = 0$, where $a = \sum_{k=i_1}^{i_2} u_k A_{k,*}$ and $c = \sum_{k=j_1}^{j_2} v_k C_{*,k}$ are random (w.r.t u_k and v_k) linear combinations, i.e. sketches, of rows of A and columns of C respectively. A preprocessing phase, where prefix sums are involved, allows to compute sketches a and c of arbitrary size in linear time during the query process. When the input matrices are sparse, the prefix sums densify the matrices thus having to compute and store n^2 elements. In addition, sparse matrices make nontrivial to compute linear combinations since row/column vectors are not explicitly stored.

We refine the analysis of [12] as follows. In order to detect the k positions (i, j) such that $(AC)_{i,j} \neq 0$, using binary search among the n^2 feasible locations, we need at most $k \log n^2$ comparisons. We note that the algorithms do not yield false positives when querying submatrices. That is, given an all-zero submatrix with related sketches a and c, it holds $\langle a, c\rangle = 0$ always. This leads to the following lemma.

Lemma 1. *Let $\mathbb{F}$ be an arbitrary field and let $A = \{a_1, \ldots, a_n\}$ and $C = \{c_1, \ldots, c_n\}$ be two sets of d-dimensional vectors such that $a_i, c_j \in \mathbb{F}^d$, for all $i, j \in [n]$. In addition, let $u_1, \ldots, u_n, v_1, \ldots, v_n$ be $2n$ random variables chosen uniformly at random from $\mathbb{F}$ and let a, c be vectors computed as $a = \sum_{k=1}^{n} u_k a_k$, $c = \sum_{k=1}^{n} v_k c_k$. If $\langle a_i, c_j \rangle = 0$, for all $i, j \in [n]$, then $\langle a, c \rangle = 0$.*

As a consequence, at most k queries produce a positive answer, i.e. $\langle a, c \rangle \neq 0$. Via a level-by-level top-down analysis, we note that at most $\min\{2^i, 2k\}$ nodes are explored at each recursive level, with $i \in [\log n^2]$. Hence, we deduce the following.

$$\sum_{i=1}^{\log n^2} \min\{2^i, 2k\} = \sum_{i=1}^{\log k} 2^i + \sum_{i=\log k}^{\log n^2} k \leq 2k + 2k \log(n^2/k). \tag{1}$$

Accordingly, we recursively split AC into two evenly divided submatrices, which resembles the splitting phase of a k-d tree. We query each submatrix and after at most $\log(n^2/k)$ queries we isolate each nonzero entry. In the following theorem we show how to compute linear combinations of sparse matrices. The intuition is to preserve the sparseness of the input matrices while computing prefix sums and generate sketches via predecessor queries, which can be efficiently computed using *fractional cascading* [17].

Theorem 1 (RAM). *Let $\mathbb{F}$ be an arbitrary field, let $A \in \mathbb{F}^{n \times n}$, $C \in \mathbb{F}^{n \times n}$ and assume A and C have h nonzero entries. After $\mathcal{O}(h)$ time for preprocessing and using deterministic $\mathcal{O}(h)$ space, it is possible to compute all the k nonzero entries of $AC \in \mathbb{F}^{n \times n}$ w.h.p, using $\mathcal{O}(kn \log(n^2/k))$ time.*

Proof. We assume that the input matrices A and C are stored in column major and row major layout respectively. If not, we can transpose A and C using $\mathcal{O}(h)$ time and $\mathcal{O}(h)$ additional space.

Preprocessing: We generate vectors $u = (u_1, \ldots, u_n) \in \mathbb{F}^n$ and $v = (v_1, \ldots, v_n) \in \mathbb{F}^n$ uniformly at random and we initialize the data structures $\mathcal{A}$ and $\mathcal{C}$ as follows: for each $A_{i,j} \neq 0$ and $C_{i,j} \neq 0$ then

$$\mathcal{A}_{i,j} = \sum_{k=1}^{i} u_k A_{k,j} \quad \mathcal{C}_{i,j} = \sum_{k=1}^{j} v_k C_{i,k} \quad A_{k,j}, C_{i,k} \neq 0, i, j \in [n]. \tag{2}$$

Intuitively, $\mathcal{A}_{i,j}$ (resp. $\mathcal{C}_{i,j}$) denotes the prefix sum of the nonzero entries of the column vector $A_{*,j}$ up to row i (row vector $C_{i,*}$ up to column j). After this phase, $\mathcal{A}$ and $\mathcal{C}$ maintain the same sparse structure, as well as the same layout, of the original input matrices. Computing $\mathcal{A}$ and $\mathcal{C}$ requires $\mathcal{O}(h)$ time and $\mathcal{O}(h)$ space.[2] Starting from column $j = n - 1$, every column vector $\mathcal{A}_{*,j}$ is augmented with every element in even position from the sparse column vector $\mathcal{A}_{*,j+1}$. After the augmentation, the vector $\mathcal{A}_{*,j}$ contains entries native to

[2] Initializing $\mathcal{A}$ and $\mathcal{C}$ corresponds to computing prefix sums of each row and column vector of A and C respectively, which requires a linear scan of the input matrices.

$\mathcal{A}_{*,j}$ and entries inherited from $\mathcal{A}_{*,j+1}$. For each inherited entry, we add pointers to its native-predecessor and its native-successor. If $\mathcal{A}_{1,j}$ is undefined, every column vector stores a *dummy* entry in first position with value 0. For each entry in $\mathcal{A}_{*,j}$, we add a *bridge* to the entry with the same row index in $\mathcal{A}_{*,j+1}$ or, if it is undefined, we add a bridge to the predecessor. Dummy entries ensure that every element in $\mathcal{A}_{*,j}$ has at least a bridge towards $\mathcal{A}_{*,j+1}$. The augmentation, together with bridging, requires a linear scan of the column vectors. The space required by the augmented vectors is $T(j) = \texttt{nnz}(A_{*,j}) + T(j+1)/2 + 1$, with $T(n) = \texttt{nnz}(A_{*,n})$ and $j \in [n-1]$, which is a geometric series bounded by $2h$. The data structure $\mathcal{A}$ is further augmented with a dense vector $\mathcal{A}_{*,0}$ where every $\mathcal{A}_{i,0}$ has a bridge to either the entry with the same row index or its predecessor in $\mathcal{A}_{*,1}$. The total space required is $2h + n \leq 3h$. Analogous considerations hold for data structure $\mathcal{C}$.

Computing AC: We recursively divide AC into two evenly divided submatrices (which resembles the splitting of a *k-d* tree) and query each submatrix in order to detect nonzero entries. Each query is answered via an inner product $\langle a, c\rangle$ where sketches a and c are constructed using fractional cascading. Given a generic submatrix of AC with indices in $[i_1, i_2] \times [j_1, j_2]$ we compute sketches of matrix A with rows in $[i_1, i_2]$ and of matrix C with columns in $[j_1, j_2]$ respectively. We start by indexing $\mathcal{A}_{i_1,0}$ which redirects to an entry $\mathcal{A}_{i,1}$. We probe the data structure for the native-predecessor, call it $\mathcal{A}_{i_p,1}$, and the native-successor, call it $\mathcal{A}_{i_s,1}$, of $\mathcal{A}_{i,1}$. Recall $i_2 \geq i_1$ and $i \leq i_1$.

1. If $\mathcal{A}_{i,1}$ is native then: (a) if $i_s < i_1$ then we emit $\mathcal{A}_{i_s,1}$, (b) if $i = i_1$ then we emit $\mathcal{A}_{i_p,1}$, (c) otherwise we emit $\mathcal{A}_{i,1}$.
2. If $\mathcal{A}_{i,1}$ is inherited then: (a) if $i_s < i_1$ then we emit $\mathcal{A}_{i_s,1}$, (b) otherwise we emit $\mathcal{A}_{i_p,1}$.

Note that, if the predecessor or the successor of $\mathcal{A}_{i,j}$ is not defined in the j-th column vector we simply output 0 or $\mathcal{A}_{i,j}$ respectively. Accordingly, we correct the following lookup by redirecting the search from either the successor $\mathcal{A}_{i_s,1}$, if $i_s < i_1$, or to $\mathcal{A}_{i,1}$, otherwise, and following its bridge to $\mathcal{A}_{i,2}$. We iterate the process up to the n-th column and we produce a n-dimensional vector a_{i_1}. The process for i_2 is analogous. Note that, for i_2, the case (1b) is omitted and inequalities become non-strict as we want to capture the elements with row index i_2. After cascading through the n columns we have vectors a_{i_1} and a_{i_2}. The sketch of the submatrix A with row indices in $[i_1, i_2]$ stems from $a = a_{i_2} - a_{i_1}$, i.e. the element-wise difference. We repeat the same process for $\mathcal{C}$ thus computing c and we query the submatrix of AC by performing the inner product $\langle a, c\rangle$. The construction of sketches a and c requires to probe the data structure a constant number of times per column and per row respectively. Hence, $\mathcal{O}(n)$ time is required per query. By Formula (1) at most $k \log(n^2/k)$ queries are required to isolate the k nonzero entries of AC. The claim follows. □

The algorithm from Theorem 1 computes k locations (i, j) to as many nonzero entries in $AC \in \mathbb{F}^{n \times n}$. In order to compute $(AC)_{i,j}$ we can retrieve, using

Formula (2), the entry value as follows $(AC)_{i,j} = \langle A_{i,*}, C_{*,j} \rangle = \sum_k \left[(\mathcal{A}_{i,k} - \mathcal{A}_{i-1,k})(\mathcal{C}_{k,j} - \mathcal{C}_{k,j-1}) \right] / u_i v_j$ while querying unit length matrices.

2.1 External Memory and Parallel External Memory

Fractional cascading relies on random memory accesses for cascading through $\mathcal{A}_{*,j}$, with $j > 1$. In the worst case, $\mathcal{O}(n)$ blocks must be loaded in memory. Instead, we use a data structure which is close in spirit to the *range coalescing* data structure by Demaine et al. [18].

Theorem 2 (Ideal Cache-Oblivious). *Let $\mathbb{F}$ be an arbitrary field, let $A \in \mathbb{F}^{n \times n}$, $C \in \mathbb{F}^{n \times n}$ and assume A and C have h nonzero entries. Let $M \geq B^{1+\varepsilon}$ for some $\varepsilon > 0$. After $\mathcal{O}((h/B) \log_{M/B}(h/B))$ I/Os for preprocessing and using deterministic $\mathcal{O}(h)$ space, it is possible to compute all the k nonzero entries of $AC \in \mathbb{F}^{n \times n}$ w.h.p., using $\mathcal{O}((kn/B) \log(n^2/k))$ I/Os.*

Proof. We describe the procedure for preprocessing matrix A and generating the sketch a. We transpose the input matrix A in column major layout using $\mathcal{O}((h/B) \log_{M/B}(h/B))$ I/Os with a cache oblivious sorting algorithm [19] (this requires the tall cache assumption $M \geq B^{1+\varepsilon}$) and we compute column-wise prefix sums using $\mathcal{O}(h/B)$ I/Os. Given the matrix A, we generate a sparse 0–1 representation A' of A, where $A'_{i,j} = 1$ if and only if $A_{i,j} \neq 0$, $A'_{i,j} = 0$ otherwise, using $\mathcal{O}(h/B)$ I/Os. We compute a counting vector $H = A'\mathbf{1}$, where $\mathbf{1} \in 1^n$ and $H_i = \sum_i \texttt{nnz}(A_{i,*})$, using a cache oblivious Sparse Matrix Vector Multiplication algorithm [20] and $\mathcal{O}((h/B) \log_{M/B}(n/M))$ I/Os. After a prefix sum over H we are able to emit h/n index positions $r_l \in [n]$ such that $\sum_{i=r_l}^{r_{l+1}} \texttt{nnz}(A_{i,*}) \leq 3n$. As a consequence, we build h/n buckets $\mathcal{A}_l$ of size $\mathcal{O}(n)$ where the elements of $\mathcal{A}_l$ are the entries $A_{i,j}$ such that $i \in [r_l, r_{l+1})$. Starting from $\mathcal{A}_2$, we incrementally augment the bucket $\mathcal{A}_l$ with elements from $\mathcal{A}_{l-1}$ such that, after the augmentation, for every column index j, there is an entry with value equal to the prefix sum up to bucket l. As in Theorem 1, we augment the data structure with a column vector $\mathcal{A}_{*,0}$ of size n, where $\mathcal{A}_{i,0}$ indices the l-th bucket if and only if $i \in [r_l, r_{l+1})$, with $l \in [h/n]$. A query on the data structure $\mathcal{A}$ probes $\mathcal{A}_{i_1,0}$ using a single I/O and it incurs in $\mathcal{O}(n/B)$ I/Os for scanning the bucket, thus generating the sketch a. Analogously, we generate the sketch c and we compute the inner product $\langle a, c \rangle$ by scanning the vectors using $\mathcal{O}(n/B)$ I/Os. □

Corollary 1 (Parallel External Memory). *Let $\mathbb{F}$ be an arbitrary field, let $A \in \mathbb{F}^{n \times n}$, $C \in \mathbb{F}^{n \times n}$, assume A and C have h nonzero entries and let $P \leq n/B$. After $\mathcal{O}((h/PB) \log_d(h/B))$ I/Os for preprocessing, with $d = \max\{2, \min\{M/B, H/PB\}\}$, and using deterministic $\mathcal{O}(h)$ space, it is possible to compute all the k nonzero entries of $AC \in \mathbb{F}^{n \times n}$ w.h.p., using $\mathcal{O}((n/PB + \log P)k \log(n^2/k))$ I/Os.*

3 Probabilistic Error Analysis

We proceed to give guarantees on the probability of detecting non-zero entries in the output matrix and we study how altering the process of random generation

alters the probability of detection. The guarantees are given in terms of the field size and not on the size of the matrix as, e.g., in [11]. Throughout the paper we gave no restriction on the field $\mathbb{F}$. Nevertheless, when $\mathbb{F}$ is infinite and countable, we require to sample from a finite subset of $\mathbb{F}$. This constraint is justified since random variables cannot be uniformly distributed among infinite and countable sets. Fields, in contrast with other algebraic structures, guarantee the existence of the multiplicative inverse for elements of $\mathbb{F}$, a property we use for proving the following lemmas.

Lemma 2. *Let $A \in \mathbb{F}^{n\times n}$, $C \in \mathbb{F}^{n\times n}$ and let $AC \in \mathbb{F}^{n\times n}$ have at most k nonzero entries. Consider a submatrix of AC with indices $[i_1, i_2] \times [j_1, j_2]$ and assume to query the submatrix with sketches a, c as in Theorem 1. (i) The matrix has a nonzero entry if and only if $\langle a, c\rangle \neq 0$ with probability at least $1 - 2/|\mathbb{F}| + 1/|\mathbb{F}|^2$. (ii) The submatrix is all zero if and only if $\langle a, c\rangle = 0$ with probability at least $1 - 2k\log(n^2/k)/|\mathbb{F}| + k\log(n^2/k)/|\mathbb{F}|^2$.*

$\mathbf{Pr}(\langle a, c\rangle = \langle u_i a_i, v_j c_j\rangle = 0)$, with $\langle a_i, c_j\rangle \neq 0$, is given by the probability of choosing either u_i or v_j zero uniformly at random from $\mathbb{F}$. By altering the algorithm, such that random entries are now generated from $\mathbb{F}^* = \mathbb{F}\backslash\{0\}$, we derive the following lemma.

Lemma 3. *Let $A \in \mathbb{F}^{n\times n}$, $C \in \mathbb{F}^{n\times n}$ and let $AC \in \mathbb{F}^{n\times n}$ have at most k nonzero entries. Let $\mathbb{F}^* = \mathbb{F}\backslash\{0\}$, consider the submatrix of AC with indices $[i_1, i_2] \times [j_1, j_2]$ and assume to query the submatrix with sketches a, c as in Theorem 1 where the entries of the vectors u and v are chosen uniformly at random from $\mathbb{F}^*$. (i) The submatrix has a nonzero entry if and only if $\langle a, c\rangle \neq 0$ with probability at least $1 - 1/|\mathbb{F}^*|$. (ii) The submatrix is all zero if and only if $\langle a, c\rangle = 0$ with probability at least $1 - k\log((n^2/k) - 1)/|\mathbb{F}^*|$.*

A Omitted Proofs

Proof (Lemma 2). (i) If $\langle a, c\rangle \neq 0$, then there exist $i, j \in [n]$ such that $u_i, v_j \neq 0$ and $\langle a_i, c_j\rangle \neq 0$, hence, $(AC)_{i,j} \neq 0$. If there is a nonzero entry then $\langle a, c\rangle \neq 0$ with probability at least $1-2/|\mathbb{F}|+1/|\mathbb{F}|^2$. This is equivalent of saying that if there is a nonzero entry then $\langle a, c\rangle = 0$ with probability at most $2/|\mathbb{F}|-1/|\mathbb{F}|^2$. Without loss of generality, let $i_1 = i_2 = i$ and $j_1 = j_2 = j$. Considering a bigger submatrix with exactly one nonzero entry leaves the probability unchanged, while considering more nonzero entries will only increase the probability of $\langle a, c\rangle \neq 0$. Therefore, we consider the case where we want to isolate, with high probability, the location of a single nonzero entry in a submatrix of unit size. It follows that, in order to query the submatrix we have to perform the following inner product $\langle a, c\rangle = \langle u_i a_i, v_j c_j\rangle$, where u, v are chosen uniformly at random from $\mathbb{F}$. Since $\langle a_i, c_j\rangle \neq 0$ by hypothesis, we have that $\mathbf{Pr}(\langle a, c\rangle = 0) \geq 2/|\mathbb{F}| - 1/|\mathbb{F}|^2$.

(ii) If the submatrix of AC with indices $[i_1, i_2]\times[j_1, j_2]$ is all zero then $\langle a, c\rangle = 0$ with probability at least $1 - 2k\log(n^2/k)/|\mathbb{F}| + k\log(n^2/k)/|\mathbb{F}|^2$. By Lemma 1 this is true. If $\langle a, c\rangle = 0$ then the submatrix of AC with indices $[i_1, i_2] \times [j_1, j_2]$

is all zero with probability at least $1-2k\log(n^2/k)/|\mathbb{F}|+k\log(n^2/k)/|\mathbb{F}|^2$. That is, if $\langle a,c\rangle = 0$ then the submatrix has a nonzero entry with probability at most $2k\log(n^2/k)/|\mathbb{F}| - k\log(n^2/k))/|\mathbb{F}|^2$. Without loss of generality, let $i_1 = i_2 = i$ and $j_1 = j_2 = j$. We have that $\langle a,c\rangle = \langle ua_i, vc_j\rangle = 0$, where u, v are chosen uniformly at random from $\mathbb{F}$. Therefore, $\mathbf{Pr}(\langle a_i, c_j\rangle \neq 0) \geq 2/|\mathbb{F}| - 1/|\mathbb{F}|^2$. The latter is a lower bound on the probability to not detect a nonzero entry in the output matrix. A union bound over the $k\log(n^2/k)$ queries needed to isolate the k nonzero entries, gives us the probability to incur in at least one false negative. By considering its complement, the claim follows. □

Proof (Lemma 3). (i) If $\langle a,c\rangle \neq 0$, then there exist $i,j \in [m]$ such that $u_i, v_j \neq 0$ and $\langle a_i, c_j\rangle \neq 0$, hence, $(AC)_{i,j} \neq 0$. If there is a nonzero entry then $\langle a,c\rangle \neq 0$ with probability at least $1 - 1/|\mathbb{F}^*|$. This is equivalent of saying that if there is a nonzero entry then $\langle a,c\rangle = 0$ with probability at most $1/|\mathbb{F}^*|$. If $i_1 = i_2 = i$, $j_1 = j_2 = j$ and $\langle a_i, c_j\rangle \neq 0$ then $\langle a,c\rangle \neq 0$ since scaling vectors with random elements from $\mathbb{F}^*$ preserves non orthogonality. If $i_1 < i_2$ and $j_1 < j_2$ and the submatrix contains exactly one nonzero entry, the same reasoning applies. If the submatrix has $\ell > 1$ nonzero entries, then, without loss of generality, there exist $a_1,\ldots,a_\ell,c_1,\ldots,c_\ell$ such that $\langle u_1a_1, v_1c_1\rangle + \cdots + \langle u_\ell a_\ell, v_\ell c_\ell\rangle = 0$ and $\langle a_i, c_j\rangle \neq 0$, for all $i,j \in [\ell]$. That is, ℓ inner products that generate as many nonzero entries and produce a false negative when the submatrix is queried. By the linearity of the inner product, we have that $\langle u_1a_1, v_1c_1\rangle + \cdots + \langle u_\ell a_\ell, v_\ell c_\ell\rangle = u_1v_1\langle a_1, c_1\rangle + \cdots + u_\ell v_\ell\langle a_\ell, c_\ell\rangle$. Hence, the sum cancels whenever $u_i = -(u_1v_1\langle a_1,c_1\rangle + \cdots + u_\ell v_\ell\langle a_\ell, c_\ell\rangle)/v_i\langle a_i, c_i\rangle$ for a generic $i \in [\ell]$. Note that, such a u_i is in $\mathbb{F}$ since fields guarantee the existence of additive and multiplicative inverses. The probability to choose u_i such that it cancels the other inner products is the same as choosing an element from $\mathbb{F}^*$ uniformly at random, i.e. $1/|\mathbb{F}^*|$.

(ii) If the submatrix of AC with indices $[i_1,i_2]\times[j_1,j_2]$ is all zero then $\langle a,c\rangle = 0$ with probability at least $1 - k\log((n^2/k)-1)/|\mathbb{F}^*|$. By Lemma 1 this is true. If $\langle a,c\rangle = 0$ then the submatrix of AC with indices $[i_1,i_2]\times[j_1,j_2]$ is all zero with probability at least $1-k\log((n^2/k)-1)/|\mathbb{F}^*|$. That is, if $\langle a,c\rangle = 0$ then the submatrix of AC has a nonzero entry with probability at most $k\log((n^2/k)-1)/|\mathbb{F}^*|$. If $\langle a,c\rangle = 0$ and $i_1 = i_2 = i$, $j_1 = j_2 = j$ then $\langle a_i, c_j\rangle = 0$. The same reasoning applies for $i_1 < i_2$, $j_1 < j_2$ and exactly one nonzero entry in the submatrix. Let $\langle a,c\rangle = 0$ and suppose there exist $a_1,\ldots,a_\ell,c_1,\ldots,c_\ell$ such that $\langle u_1a_1, v_1c_1\rangle + \cdots + \langle u_\ell a_\ell, v_\ell c_\ell\rangle = 0$ and $\langle a_i, c_j\rangle \neq 0$, for all $i,j \in [\ell]$. Hence, as in (i), the sum cancels with probability $1/|\mathbb{F}^*|$. The latter is a lower bound on the probability to not detect a nonzero entry in the output matrix. A union bound over the $k\log((n^2/k)-1)$ queries needed to isolate the k nonzero entries, gives us the probability to incur in at least one false negative.[3] By considering its complement, the claim follows. □

[3] We do not consider the last layer, i.e. $\log(n^2/k)$, as it does not involve any stochastic process.

References

1. Aggarwal, A., Vitter, J.S.: The input/output complexity of sorting and related problems. Commun. ACM **31**(9), 1116–1127 (1988)
2. Frigo, M., Leiserson, C.E., Prokop, H., Ramachandran, S.: Cache-oblivious Algorithms. In: 40th Annual Symposium on Foundations of Computer Science, pp. 285–297. IEEE (1999)
3. Arge, L., Goodrich, M.T., Nelson, M., Sitchinava, N.: Fundamental parallel algorithms for private-cache chip multiprocessors. In: Proceedings of the 20th Annual Symposium on Parallelism in Algorithms and Architectures, SPAA 2008, pp. 197–206. ACM, New York (2008)
4. Bender, M.A., Fineman, J.T., Gilbert, S., Kuszmaul, B.C.: Concurrent cache-oblivious B-trees. In: Proceedings of the 17th Annual ACM Symposium on Parallelism in Algorithms and Architectures, pp. 228–237. ACM (2005)
5. Strassen, V.: Gaussian elimination is not optimal. Numer. Math. **13**(4), 354–356 (1969)
6. Le Gall, F.: Powers of tensors and fast matrix multiplication. In: Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation, pp. 296–303. ACM (2014)
7. Yuster, R., Zwick, U.: Fast sparse matrix multiplication. ACM Trans. Algorithms (TALG) **1**(1), 2–13 (2005)
8. Iwen, M.A., Spencer, C.V.: A note on compressed sensing and the complexity of matrix multiplication. Inf. Process. Lett. **109**(10), 468–471 (2009)
9. Amossen, R.R., Pagh, R.: Faster join-projects and sparse matrix multiplications. In: Proceedings of the 12th International Conference on Database Theory, ICDT 2009, pp. 121–126. ACM, New York (2009)
10. Lingas, A.: A fast output-sensitive algorithm for Boolean matrix multiplication. In: Fiat, A., Sanders, P. (eds.) ESA 2009. LNCS, vol. 5757, pp. 408–419. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04128-0_37
11. Pagh, R.: Compressed matrix multiplication. In: Proceedings of the 3rd Innovations in Theoretical Computer Science Conference, pp. 442–451. ACM (2012)
12. Williams, R., Yu, H.: Finding orthogonal vectors in discrete structures. In: Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Philadelphia, PA, USA, pp. 1867–1877 (2014)
13. Jacob, R., Stöckel, M.: Fast output-sensitive matrix multiplication. In: Bansal, N., Finocchi, I. (eds.) ESA 2015. LNCS, vol. 9294, pp. 766–778. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48350-3_64
14. Van Gucht, D., Williams, R., Woodruff, D.P., Zhang, Q.: 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 2015, pp. 199–212. ACM, New York (2015)
15. Hong, J.W., Kung, H.T.: I/O complexity: the red-blue pebble game. In: Proceedings of the 13th Annual ACM Symposium on Theory of Computing, STOC 1981, pp. 326–333. ACM, New York (1981)
16. Pagh, R., Stöckel, M.: The input/output complexity of sparse matrix multiplication. In: Schulz, A.S., Wagner, D. (eds.) ESA 2014. LNCS, vol. 8737, pp. 750–761. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44777-2_62
17. Chazelle, B., Guibas, L.J.: Fractional cascading: I. A data structuring technique. Algorithmica **1**(1), 133–162 (1986)

18. Demaine, E.D., Gopal, V., Hasenplaugh, W.: Cache-oblivious iterated predecessor queries via range coalescing. In: Dehne, F., Sack, J.-R., Stege, U. (eds.) WADS 2015. LNCS, vol. 9214, pp. 249–262. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21840-3_21
19. Brodal, G.S., Fagerberg, R.: Cache oblivious distribution sweeping. In: Widmayer, P., Eidenbenz, S., Triguero, F., Morales, R., Conejo, R., Hennessy, M. (eds.) ICALP 2002. LNCS, vol. 2380, pp. 426–438. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45465-9_37
20. Bender, M.A., Brodal, G.S., Fagerberg, R., Jacob, R., Vicari, E.: Optimal sparse matrix dense vector multiplication in the I/O-model. Theory Comput. Syst. **47**(4), 934–962 (2010)