Evaluating Non-Square Sparse Bilinear Forms on Multiple Vector Pairs in the I/O-Model

Gero Greiner and Riko Jacob

Technische Universität München

Abstract. We consider evaluating one bilinear form defined by a sparse $N_y \times N_x$ matrix $\mathbf{A}$ having h entries on w pairs of vectors The model of computation is the semiring I/O-model with main memory size M and block size B. For a range of low densities (small h), we determine the I/O-complexity of this task for all meaningful choices of N_x, N_y, w, M and B, as long as $M \geq B^2$ (tall cache assumption). To this end, we present asymptotically optimal algorithms and matching lower bounds. Moreover, we show that multiplying the matrix $\mathbf{A}$ with w vectors has the same worst-case I/O-complexity.

1 Introduction

We consider the problem of computing w scalars $z^{(i)} = {\mathbf{y}^{(i)}}^T \mathbf{A}\mathbf{x}^{(i)}$, $0 \leq i \leq w$, where $\mathbf{A}$ is a sparse matrix with h non-zero entries, and all $\mathbf{x}^{(i)}$ and $\mathbf{y}^{(i)}$ are (dense) vectors. This is highly related to the matrix vector products $\mathbf{A}\mathbf{x}^{(i)}$, $0 \leq i \leq w$, and we show that both tasks actually have asymptotically the same complexity in our model. While, from a traditional point of view, bilinear form and matrix vector product are easily obtained with a number of multiplications equal to the number of matrix entries, the sparseness sometimes induces irregular access patterns that lead to situations where memory access becomes the bottleneck of computation. Empirical studies show that for the naive algorithm, CPU-usage can be as low as 10% [6,9].

One way of dealing with this problem is the construction of algorithms where, instead of CPU-cycles, the movement of data between layers of the memory hierarchy is optimized. In this paper, we use a slight modification of the I/O-model [1], the semiring I/O-model with the parameters M and B, denoting the memory size, and the size of a block, see Section 2 for details. In this model, Bender, Brodal, Fagerberg, Jacob and Vicari [2] determined the I/O-complexity of computing the sparse matrix vector product for square matrices. These results are generalised here to the case of non-square matrices. Furthermore, we extend the results to the evaluation of multiple products, i.e., the matrix vector products of multiple vectors with the same matrix. Considering the evaluation of matrix vector products on multiple vectors is a step towards closing the gap between sparse matrix vector multiplication and sparse matrix dense matrix multiplication since the set of w vectors $\mathbf{x}^{(i)}$ constitutes a dense $N_x \times w$ matrix $\mathbf{X}$.

Related work Evaluating the matrix vector product $\mathbf{Ax}$ for an $N \times N$ matrix $\mathbf{A}$ with kN entries has been investigated in [2]. They show that the I/O-complexity of this task for matrices in the so called column major layout is $\Theta\left(\min\left\{\frac{kN}{B}\,\overline{\log}_{\frac{M}{B}}\frac{N}{\max\{M,k\}}, kN\right\}\right)$[1], and $\Theta\left(\min\left\{\frac{kN}{B}\,\overline{\log}_{\frac{M}{B}}\frac{N}{Mk}, kN\right\}\right)$ for a layout chosen by the program.

The multiplication of two dense matrices has been examined by Hong and Kung in [5], showing a bound of $\Theta\left(\frac{N^3}{B\sqrt{M}}\right)$ on the number of I/Os. Very recently, in [4] the multiplication of a sparse matrix with a dense matrix was considered. For sparse $N \times N$ matrices with kN entries, it is shown that for certain ranges of k, the performance can be increased by finding denser than average submatrices.

The evaluation of bilinear forms in the I/O-model has been considered as an optimisation problem in [7]. There it has been shown that an optimal program for the evaluation of matrix vector products is $\mathcal{NP}$-hard to find, even for $B = 1$. In [8], the I/O-complexity of evaluating the bilinear form for a (non-square) $N_y \times N_x$ matrix with h entries that form a diagonal band, i.e., entries are only placed near the diagonal, is determined to be $\Theta\left(\frac{h}{BM} + \frac{N_x+N_y}{B}\right)$.

Our results In this work, we consider the case where the number of entries for each submatrix in $\mathbf{A}$ is proportional to the number of rows and columns of the submatrix. This is possible if the average number of entries per column in $\mathbf{A}$ is some $k \leq \frac{N_y}{M^{1-\epsilon}N_x^{\epsilon}}$ with constant $\epsilon > 0$. For such k, a modification of the proof of [4] shows that the I/O-complexity of $\mathbf{AX}$ for any $w \geq B$ is $\Theta\left(\frac{wh}{B}\right)$. As a lower bound, this extends directly to the case of multiplying a sparse matrix with w vectors, even if the program is allowed to choose the layout of the vectors.

The case of $w \leq B$ is examined here in detail, forming a bridge between the results of [2] and [4]. For matrices of the described density ($h/N_x \leq \frac{N_y}{M^{1-\epsilon}N_x^{\epsilon}}$), we cover all dimensions of $\mathbf{A}$, and products with an arbitrary number of vectors. However, for all other choices of h, we present upper bounds in form of algorithms. For certain cases where B/w is small the algorithms are indeed optimal for all ranges of h. Furthermore, we show that evaluating the bilinear form has the same I/O-complexity as multiplying vectors with the same matrix.

Theorem 1. *Given a matrix $\mathbf{A}$ with fixed layout and fixed parameters M and B. Evaluating w bilinear forms with $\mathbf{A}$ has the same semiring I/O-complexity as evaluating the matrix vector product of $\mathbf{A}$ with w vectors if at least $\ell = \Omega\left(\frac{hw}{B}\right)$ I/Os are required.*

This result is explained in Section 3 and allows us to extend the results from [2] to matrices in row major layout, i.e., where the entries are given in external memory in a consecutive ordering by their row index, and their column index to break ties. Moreover, our main results hold for bilinear forms and matrix vector products, both, in column major and row major layout. Note that for Theorem 2, the dimensions N_x and N_y have to be swapped if $\mathbf{A}$ is given in row major layout.

[1] $\overline{\log}_b(x) := \max\{\log_b(x), 1\}$

For the complexity of the task, we are going to prove the following theorems depending on the layout of the matrix $\mathbf{A}$. Similar to [4], our bounds are only asymptotically tight if we assume a tall cache, i.e., $M \geq B^2$. However, this is only necessary to transpose $\mathbf{X}$ for some of the presented algorithms.

Our results for the case that the matrix $\mathbf{A}$ is stored in column major layout, are given in the following theorem.

Theorem 2. *Given an $N_y \times N_x$ matrix $\mathbf{A}$ with h entries in column major layout and parameters B, M. Assume $M \geq 4B$, $h/N_x \leq N_y^\epsilon$, and $\log N_x \leq N_y^\epsilon$ for some $0 < \epsilon < 1$. Then, evaluating w bilinear products with $\mathbf{A}$ has (worst-case) complexity in the semiring I/O-model*

$$\Theta\left(\min\left\{h\,,\ \tfrac{h\log N_y}{\log N_x}\,,\ \tfrac{h}{B}\log_{\frac{M}{B}}\min\left\{\tfrac{N_y}{M}, \tfrac{N_xN_y}{h}\right\} + \tfrac{hw}{B}\log_{\frac{M}{B}}\tfrac{N_xN_y}{hM}\right\}\right)$$

unless this term is asymptotically smaller than $\frac{hw}{B}$.

The terms are obtained by a modification of the proof in [2] for a single matrix vector multiplication, also keeping track of the different matrix dimensions. Additionally, the lower bounds of Theorem 3 apply which yields the second term of the sum.

On the algorithmic side, for very asymmetric matrices $\mathbf{A}$, where the number of columns is much higher than the number of rows, building a table of tuples with multiple dimensions of all $\mathbf{y}^{(i)}$ vectors can be superior to the direct algorithm. The direct algorithm simply scans over $\mathbf{A}$ and loads for each a_{ij} the corresponding vector elements $x_j^{(0)}, \ldots, x_j^{(w)}$ and $y_i^{(0)}, \ldots, y_i^{(w)}$ to create products.

In [2], an algorithm is presented based on sorting the matrix entries to build row sums. This sorting approach can be used to initially change the layout of $\mathbf{A}$ to the best-case layout. Then, the sorting algorithm for best-case layout can be applied for one vector pair after another.

For the best-case layout, i.e., if the algorithm is allowed to choose the layout of the matrix, the following theorem holds.

Theorem 3. *Given an $N_y \times N_x$ matrix $\mathbf{A}$ with h entries in best-case layout and parameters B, M. Assume $M \geq 4B$, for $N_y \leq N_x$, and $h/N_x \leq \sqrt[6]{N_y}$. Then, evaluating w bilinear products with $\mathbf{A}$ has (worst-case) complexity in the semiring I/O-model*

$$\Theta\left(\min\left\{\tfrac{hw}{B}\log_{\frac{M}{B}}\tfrac{N_xN_y}{hM}\,,\ h\,,\ h\log(\tfrac{wN_xN_y\log N_x}{BhM})/\log N_x\right\}\right)$$

unless this term is asymptotically smaller than $\frac{hw}{B}$.

All our algorithms require at least hw/B I/Os. In contrast, we do not know a corresponding lower bound that would hold for all choices of the parameters. However, if the dimensions are polynomially bounded in each other (which is expressed using a condition on the density in the lemma), the results of [4] can be extended to obtain a lower bound of $\Omega(hw/B)$ by the following lemma.

Lemma 1. *Let* $\mathbf{A}$ *be a sparse* $N_y \times N_x$ *matrix with* h *non-zero entries for* $N_x \geq N_y$*. For an average number of entries per column* $h/N_x \leq N_y/(M^{1-\epsilon}N_x^{\epsilon})$ *with constant* $\epsilon > 0$*, evaluating* w *bilinear products over a semiring requires* $\Omega\left(\frac{hw}{B}\right)$ *I/Os.*

The proof of this lemma can be found in [3]. Hence, for a $N_y \times N_x$ matrix $\mathbf{A}$ with h non-zero entries, for $N_x \geq N_y$ but $N_x^{2\epsilon} \leq N_y$, and average number of entries per column $h/N_x \leq \sqrt{N_y}/M$ a lower bound of $\Omega(hw/B)$ is given.

1.1 Outline

The outline of the paper is as follows: In Section 2, the model of computation is introduced along with the terminology used in this paper. The main results are then proven by providing (optimal) algorithms for upper bounding the complexity in Section 4 and up to constant factors matching lower bounds in Section 5. Due to size limitations, not all proofs are presented in full detail here, but they can be found the full version of the paper [3].

2 Model of Computation

We use a combination of the I/O-model described in [1] and the model used in [5], the so called *semiring I/O-model* introduced in [2]. It models two layers of the memory hierarchy, namely a fast memory of limited capacity M called *internal memory*. Calculations, namely addition, multiplication, copying, and deletion, can only be performed on the elements residing in internal memory, whereas the program inputs and any (intermediate) results are stored on an *external memory* (aka disk) of unbounded size which is organised in blocks (aka tracks) of size B. The I/O cost of a program is the number of transfers of a block between internal and external memory. Programs are assumed to work for an arbitrary semiring defining addition and multiplication, i.e., subtraction and division may not be used. Hence, all intermediate results have one of the following forms: $a_{jk}x_k^{(i)}$, $y_j^{(i)}a_{jk}$, $x_k^{(i)}y_j^{(i)}$ and $y_j^{(i)}a_{jk}x_k^{(i)}$, for $1 \leq i \leq w$, $j \in [N_x]$, $k \in [N_y]$, are referred to as *elementary products*, using the notation $[N] = \{1, \ldots, N\}$. Sums of the form $\sum_{k\in S'} a_{jk}x_k^{(i)}$, $\sum_{j\in S} y_j^{(i)}a_{jk}$, and $\sum_{j\in S}\sum_{k\in S'} y_j^{(i)}a_{jk}x_k^{(i)}$, with $1 \leq i \leq w$, $S \subseteq [N_x]$ and $S' \subseteq [N_y]$, are called *partial sums*. Altogether, the term *canonical partial result* refers to any of these forms. The detailed definition and the argument leading to this classification can be found in [3]. For the lower bounds we use the non-uniform notion that an *algorithm* is a family of programs where the program can be chosen according to the parameters underlying the problem. By $\ell(\mathcal{A})$, we denote the maximum number of I/Os induced by the algorithm $\mathcal{A}$ for all choices of the parameters, unless otherwise noted. In particular, for matrix multiplications, the parameters are the dimensions, the sparseness, the *conformation* of the matrix $\mathbf{A}$, i.e., the position of the non-zero entries in $\mathbf{A}$, the memory size M and the block size B.

Since we can assume that every program requires at least one I/O, when writing complexity using $\mathcal{O}$, Θ, or Ω at least 1 is meant.

3 Transformations

In this section, we discuss how a program that evaluates bilinear forms can be transformed into one that computes matrix vector multiplications.

Lemma 2. *In a normalised semiring I/O program evaluating a bilinear form on multiple vector pairs, no elementary product is created twice.*

For the proof of Theorem 1, we present transformations in both directions in the following lemmas. Note that the layout of the matrix $\mathbf{A}$ is not of concern, it just has to be the same for both tasks. We state the easy transformation without a proof:

Lemma 3. *If the matrix vector products $\mathbf{A}\mathbf{x}^{(i)}$ for $1 \leq i \leq w$ can be computed for an arbitrary semiring with ℓ I/Os, then the bilinear forms ${\mathbf{y}^{(i)}}^T\mathbf{A}\mathbf{x}^{(i)}$ can be evaluated with at most 3ℓ I/Os.*

Lemma 4. *If the bilinear forms ${\mathbf{y}^{(i)}}^T\mathbf{A}\mathbf{x}^{(i)}$, $1 \leq i \leq w$ can be evaluated in the semiring I/O-model with internal memory size M and block size B using ℓ I/Os, then the w products $\mathbf{c}^{(i)} = \mathbf{A}\mathbf{x}^{(i)}$ can be computed using $2\ell + 4wh/B$ I/Os with internal memory size $M + B$ and block size B.*

Proof. Here, we will only give a short description of the transformation of a program. A more detailed analysis can be found in [3].

Let P be a program to evaluate the w bilinear forms using ℓ I/Os. By Lemma 2, there is a program $\hat{P}$ for the same task which computes only canonical partial results with at most ℓ I/Os. We can then use $\hat{P}$ to construct a program for the matrix vector products. This construction is based on the following idea. During a simulation of $\hat{P}$, canonical partial results are extracted, and temporarily stored on disk. In a second phase, $\hat{P}$ is simulated time-reversed, as will be described later on, and the movement of $y_j^{(i)}$ variables in $\hat{P}$ can be used to lead the previously extracted results to the corresponding position in $\mathbf{y}^{(i)}$. In the end, the memory cells, where $\mathbf{y}^{(i)}$ is expected for P, constitute $\mathbf{c}^{(i)}$.

Construction For the first phase, we create a program P_F for the semiring I/O-model with internal memory size $M + B$. We use the first M cells in memory for a simulation of $\hat{P}$, and reserve the last B cells $(m_{M+1}, \ldots, m_{M+B}) =: \mathcal{B}$ for further output operations. As soon as $\mathcal{B}$ is entirely full, i.e., no element in $\mathcal{B}$ is 0, the block is moved to disk.

During the simulation of $\hat{P}$, the following additional operations are performed. If a computation σ in $\hat{P}$ performs a multiplication of an element $m_l = y_j^{(i)}$ or $m_l = y_j^{(i)}x_k^{(i)}$ with an element $m_{l'}$ then $m_{l'}$ is copied into an empty position of $\mathcal{B}$ immediately before σ is performed. Furthermore, whenever in $\hat{P}$ a computation σ involves an element $m_l = x_k^{(i)}$, the result can only be of the form $x_k^{(i)}a_{jk}$, $x_k^{(i)}y_j^{(i)}$, $x_k^{(i)}(y_j^{(i)}a_{jk})$, or $x_k^{(i)}(\sum_{j\in S} y_j^{(i)}a_{jk})$. For the latter two cases, $x_k^{(i)}$ is copied into an empty cell of $\mathcal{B}$ before performing σ. We call these newly created copy operation *snapshot* and σ its associated operation.

The program P_F is executed with input $\mathbf{A}$ and $\mathbf{x}^{(i)}$ as given, but $\mathbf{y}^{(i)} = (1, \ldots, 1)$ for all $1 \leq i \leq w$. For each elementary product created in $\hat{P}$, there are at most two elements copied to $\mathcal{B}$ (the corresponding $x_k^{(i)}$ and a_{jk}). Recall that in $\hat{P}$ elementary products are only created once. Since there are at most wh elementary products necessary, P_F performs no more than $\ell + \left\lceil \frac{2wh}{B} \right\rceil$ I/Os.

For the second phase, we have to time-reverse P_F. In this phase, we consider only the elements that consist of a polynomial containing a $y_j^{(i)}$, all other elements are ignored. When time is inverted, naturally an input becomes an output and vice versa. Internal computation operations are mapped in the following way. To this end, each copy operation of $\hat{P}$ that sets $m_k := m_l$ becomes a sum operation $m_l := m_l + m_k$ in the time-reversed program P_B. Each sum operation $m_i := m_j + m_k$ in P_F becomes a copy operation $m_j := m_k := m_i$ in P_B. Delete operations of P_F are simply ignored in P_B, i.e., nothing is created. The additional snapshot operations introduced in P_F are only made when an $y_j^{(i)}$ is involved in a computation operation σ. Considering the different cases of associated operations, the elements that were extracted in P_F are now copied, multiplied, or added into one of the cells that are accessed by σ.

Correctness Since every canonical partial result that includes some $y_j^{(i)}$ has an input of the element $y_j^{(i)}$ as its predecessor in P_F, in the time-reversed P_B, all created partial results can be transferred to the initial position of $y_j^{(i)}$. Furthermore, since all hw elementary products have to be created for the bilinear product, and an $y_j^{(i)}$ is a predecessor for each, the created vectors $\mathbf{c}^{(i)}$, $1 \leq i \leq w$ are complete, i.e., the summation does not lack summands.

4 Algorithms

Note that all the presented algorithms can be used for both, evaluating matrix vector products and bilinear forms, by Theorem 1.

4.1 Direct Algorithm

The computation of $w \leq B$ bilinear forms is possible with $\mathcal{O}(h)$ I/Os by considering the non-zero entries of $\mathbf{A}$ in an arbitrary order. For every entry a_{jk} the elementary products $x_k^{(0)} a_{jk} y_j^{(0)}, \ldots, x_k^{(w)} a_{jk} y_j^{(w)}$ are added to the respective current partial sums $z^{(0)}, \ldots, z^{(w)}$. For this to occur only a constant number of I/Os, the values $x_k^{(0)}, \ldots, x_k^{(w)}$ need to be stored in one block (or at least consecutively on disk), similarly to $y_j^{(0)}, \ldots, y_j^{(w)}$. This can be achieved by transposing the matrices $\mathbf{X} = \left[\mathbf{x}^{(0)} \ \ldots \ \mathbf{x}^{(w)}\right]$ and $\mathbf{Y} = \left[\mathbf{y}^{(0)} \ \ldots \ \mathbf{y}^{(w)}\right]$, which takes $\mathcal{O}((N_x + N_y)/B) = \mathcal{O}(h)$ I/Os [1], given the tall cache assumption $M \geq B^2$.

4.2 Sorting Based Algorithm

In [2] a sorting based approach for evaluating the product $\mathbf{Ax}$ for square matrices is presented. These algorithms can be extended straightforwardly to the matrix vector product of a non-square matrix $\mathbf{A}$ with one vector $\mathbf{x}$. For column major layout the vector $\mathbf{c} := \mathbf{Ax}$ can be created with $\mathcal{O}\left(\frac{h}{B}\,\overline{\log}_{M/B}\min\left\{\frac{N_xN_y}{h}, \frac{N_y}{M}\right\}\right)$ I/Os.

For the best-case layout the algorithm uses $\mathcal{O}\left(\frac{h}{B}\,\overline{\log}_{M/B}\frac{N_xN_y}{Mh}\right)$ I/Os. With slight modifications, this algorithm can also be described for a broader class of layouts. For this class of best-case layouts, the matrix $\mathbf{A}$ is given as a split-up of its columns into meta-columns where each meta-column is written in row major layout. Columns of a meta-column have to be continuous, each column is assigned to only one meta-column, and each meta-columns consist of an arbitrary number of columns, but at most $M - B$. Additionally, the number of meta-columns is at most $\lceil N_x/B\rceil + 2\lceil h/N_y\rceil$. Since each meta-column consists of no more than $M - B$ continuous columns, for each meta-column, the corresponding elements of $\mathbf{x}$ can all be loaded into internal memory, and the meta-column is scanned to create elementary products which are then written back to $\mathbf{A}$. Afterwards, if $N_x/B > h/N_y$, meta-columns are merged together using Merge sort until there are at most h/N_y runs. Since in this case there are no more than $3\lceil N_x/B\rceil$ meta-columns, $\mathcal{O}\left(\frac{h}{B}\,\overline{\log}_{M/B}\frac{N_xN_y}{Bh}\right) = \mathcal{O}\left(\frac{h}{B}\,\overline{\log}_{M/B}\frac{N_xN_y}{Mh}\right)$ I/Os are sufficient. Otherwise, if $h/N_y \geq N_x/B$, there are at most $3\lceil h/N_y\rceil$ meta-columns. Since meta-columns and runs are in row major layout, with one scan of each meta-column / run, elements from the same row can be summed together, and meta-columns / runs become a single column. All created columns can then be summed into the first column with $\mathcal{O}\left(\frac{h}{N_y}\cdot\frac{N_y}{B}\right)$ I/Os. Hence, in all cases, the matrix vector product for a matrix $\mathbf{A}$ given in a layout meeting the conditions described can be determined with $\mathcal{O}\left(\frac{h}{B}\,\overline{\log}_{M/B}\frac{N_xN_y}{Mh}\right)$ I/Os.

Multiple Vectors For the evaluation of w matrix vector products, the algorithms can simply be run for each single vector, which increases the running time by a factor w. However, for column major layout, it can be faster to transform the layout of $\mathbf{A}$ into one belonging to the class of generalized best-case layouts described above, and then use that algorithm for each single vector.

The transformation of the layout has two cases, depending on the parameters. The first case handles situations with $N_x \leq h/(M - B)$, where the average column consists of more than $M - B$ already sorted entries. Then the N_x columns are bottom up merged using the M/B-way Merge sort, each time reducing the number of meta-columns by a factor of M/B. This is continued as long as the resulting meta columns have width $\leq M - B$, and the number of meta columns is greater than h/N_y. Hence, the running time of this merging is $\mathcal{O}\left(\frac{h}{B}\log_{M/B}\min\left\{M, \frac{N_xN_y}{h}\right\}\right)$ I/Os, and there are at most $\max\left\{\lceil N_x/(M-B)\rceil, \lceil h/N_y\rceil\right\}$ meta-columns that can contain less than $N_y/2$ entries, i.e., they form a generalised best case layout.

The second case assumes $h/(M-B) \leq N_x$, and mimics the creation of initial runs of length $M-B$. The possibility of columns having vastly different number of entries makes this slightly more involved. First, the columns of $\mathbf{A}$ are split into $2h/N_y$ continuous groups such that each group contains at most N_y entries. Then, each group that spans at most $M-B$ columns is transformed into row major layout using the classical M/B-way Merge sort in $\mathcal{O}\left(\frac{h}{B}\,\overline{\log}_{M/B}\,\frac{N_y}{M}\right)$ I/Os Groups that span more than $M-B$ columns are divided into subgroups that span at most $M-B$. This can be achieved by greedily assigning the blocks of a group to subgroups such that each subgroup spans no more than $M-B$ columns. Splitting blocks that belong to two columns is possible with $\mathcal{O}(N_x/M)$ I/Os. The subgroups are then transformed into row major layout using Merge sort. Since there are at most N_y/B blocks per group, this can be done with another $\mathcal{O}\left(\frac{h}{B}\,\overline{\log}_{M/B}\,\frac{N_y}{M}\right)$ I/Os.

Because one block spans at most B columns, each of the subgroups, except at most one per group, spans at least $M-2B$ columns. Since we assume $M \geq 4B$ in this paper, for each group, there can be at most one meta-column with span less than $2B$. Hence, in the created layout, we have at most $N_x/(2B) + 2h/N_y$ meta-columns, each spanning at most $M-B$ columns, and the algorithm for the class of generalised best-case layouts can be applied.

4.3 Table Based Algorithm

For very asymmetric cases of $\mathbf{A}$ where $N_x \gg N_y$, the construction of tuples of rows of $\mathbf{Y}$, such that arbitrary dimensions of each vector can be loaded within one I/O. We present the following algorithms in the setting of evaluating of bilinear products.

Column Major Layout The bilinear forms $\mathbf{y}^{(i)}\mathbf{A}\mathbf{x}^{(i)}$ for w vector pairs can be evaluated with $\mathcal{O}\left(\max\left\{\frac{wh}{B}, h\frac{\log N_y}{\log N_x}\right\}\right)$ I/Os as follows. The algorithm starts by creating a table of all c-tuples of rows of $\mathbf{Y}$ in lexicographical order of the row indices. To this end, define $c := \min\left\{\left\lfloor\frac{B}{w}\right\rfloor - 1, \left\lfloor\frac{\log N_x}{2\log N_y}\right\rfloor\right\}$ such that a c-tuple of rows of $\mathbf{Y}$ does not exceed one block. Since we assume that $\mathbf{Y}$ is in column major layout, it first has to be transposed which is possible with $\mathcal{O}(wN_y/B)$ I/Os (cf. Section 4.1). Further, since we assume a tall cache, i.e. $M \geq B^2$, internal memory can hold c blocks at a time and one for the output of a created tuple.

A table of all c-tuples has size $wcN_y^c \leq wc\sqrt{N_x} \leq wN_x$, where the last inequality relies upon $c \leq \frac{1}{2}\log N_x\sqrt{N_x}$, which is true for all N_x. The table can easily be created in $\mathcal{O}(wN_x/B)$ I/Os, a term dominated by the I/Os needed to read $\mathbf{X}$.

After creating a table of all c-tuples of rows of $\mathbf{Y}$, the algorithm simultaneously scans the entries of $\mathbf{A}$ and the corresponding elements of $\mathbf{X}$. Since we have $M \geq 4B$, we use one block for the scanning of $\mathbf{A}$, one for elements of $\mathbf{X}$, one for a c-tuple of $\mathbf{Y}$, and the last block to sum elementary results together for each of the $w \leq B$ vectors. Throughout the scanning of $\mathbf{A}$, for each $c \leq B$ entries

$a_{i_1,j_1}, \ldots, a_{i_c,j_c}$, the c-tuple containing the corresponding rows $i_1, \ldots i_c$ of $\mathbf{Y}$ is loaded, and elementary products for each pair of vectors are created. These can be summed immediately into the block reserved for the results. Hence, $\mathcal{O}(h/c)$ I/Os are sufficient to evaluate the w bilinear forms.

Best-case Layout If $w < B$ and $N_x \geq N_y^2$, the table-based approach for column major layout can be improved using a different layout of $\mathbf{A}$. In the following, we assume $c \leq B/w$, otherwise the algorithm for column major layout is applied. Similarly, the matrices $\mathbf{X}$ and $\mathbf{Y}$ are transposed in the beginning.

Again, we create a table of c-tuples of rows of $\mathbf{Y}$, but for different c. This time, the matrix $\mathbf{A}$ is read in tiles such that each tile contains on average $(M-B)/w$ entries, and the layout of $\mathbf{A}$ reflects these tiles. Using this, a tile can be loaded and all elementary products can be created while still one free block is available in internal memory. For a tile of height $\frac{wc^2}{B}\frac{2N_xN_y}{hM}$ rows and width $\frac{MB}{2wc}$ columns, we get a performance of

$$\mathcal{O}\left(h\frac{\log\left(\frac{wN_xN_y}{hBM}\log N_x\right)}{\log N_x}\right).$$

The details of the calculations can be found in the full version of the paper [3].

5 Lower bounds

For the lower bounds, we only consider matrix vector products. By Theorem 1 this also implies lower bounds for bilinear products.

5.1 Column Major Layout

The following lower bound is only for single matrix vector products. However, together with the lower bound for the best-case layout, multiple evaluations are covered too.

Lemma 5. *Computing over an arbitrary semiring the bilinear product with an $N_y \times N_x$ matrix A with h entries, stored in column major layout has (worst-case) I/O-complexity for B, M with $M \geq 4B$*

$$\Omega\left(\min\left\{\frac{h}{B}\log_{\frac{M}{B}}\frac{N_y}{M} \quad , \quad \frac{h}{B}\log_{\frac{M}{B}}\frac{N_xN_y}{h} \quad , \quad h \quad , \quad h\max\left\{\frac{1}{B}, \frac{\log N_y}{\log N_x}\right\}\right\}\right).$$

To proof this lemma, the dimensions of $\mathbf{A}$ have to be simply replaced in the proof in [2]. The calculations can be found in [3].

5.2 Best-case Layout

As described in Section 2, for the best-case layout it is up to the program to choose the layout of $\mathbf{A}$. The proof of Lemma 5 is based on the task of computing

row sums. To obtain a lower bound for the best-case layout, we have to use a different approach because producing row sums is trivial when using a row major layout. Therefore, we consider the sequence of configurations of a program and follow the movement of input variables of $\mathbf{X}$ and partial results of $\mathbf{Y}$. Furthermore, we allow accessing $\mathbf{A}$ for free. This can only weaken the lower bound.

We count the number of different matrix conformations that can be handled by programs for matrix vector multiplication with ℓ I/Os. For a given program, the conformation of a matrix can be identified by considering multiplication operations including input variables, and their results: When there is an input variable $x_j^{(i)}$ loaded, and it is used to form an elementary product that is a predecessor of $c_k^{(i)}$, this describes the existence of a non-zero entry a_{ik} in $\mathbf{A}$. Hence, by tracking all copies of input variables $x_j^{(i)}$ and all elements that are predecessors of a unique result $c_k^{(i)}$ (this can be elementary products or partial sums), and by choosing the positions in a program where multiplications involving such elements are performed, the conformation of a matrix is uniquely determined. To do this, it suffices to consider the tracking of elements only for one of the w matrix vectors multiplication. All these information will be called *trace* in the following.

In order to describe the trace, we normalise programs which changes the number of I/Os only by constant factors. The following normalisation is a variation of [5, Theorem 3.1].

Lemma 6. *Assume there is an I/O program $\mathcal{A}$ performing ℓ I/Os for parameters M and B. Then there is an I/O program $\mathcal{B}$ computing the same function performing at most $3\ell + M/B$ I/Os for parameters $2M$ and B, that works in rounds: Each round consists of $2M/B$ input operations, an arbitrary number of computation operations followed by $2M/B$ output operations such that after each round internal memory is empty.*

In the following, we consider programs in rounds according to the above lemma. To determine the traces of input variables and result predecessors in a round-based program, we consider the transfer of blocks between rounds, i.e. a block that is output by one round and input by another.

The movements of input variables can be described as follows. For a vector $\mathbf{x}^{(k)}$, we consider the subset $\mathcal{T}_{\mathcal{V},i} \subseteq [N_x]$ of indices of elements $x_i^{(k)}$ in a block i and trace the copying and deletion of variables in each round. For the trace of a predecessor of a unique result $c_j^{(k)}$, we abstract from the element itself, and consider only the index of the result j. Hence, we have the subset $\mathcal{T}_{\mathcal{R},i} \subseteq [N_y]$ of indices of unique result predecessors transferred by block i.

As written before, it suffices to consider the traces for one pair of input and result vector only. Every block of an I/O can be separated into values belonging to the w different tasks implied by the different pairs of vectors. Hence, for the l-th I/O, we have the number $u_l^{(k)}$ of elements belonging to vector pair k. By averaging we have $\sum_{0 \leq l \leq \ell} u_l^{(k)} \leq B\ell/w$ for some k, and we determine the traces for this pair of vectors in the following.

Describing the traces Because we will describe the traces of programs by blocks transferred between rounds, we view the input variables as output of rounds with no cost. Further, we are only interested in lower bounds below h such that we can assume $\ell \leq h$. Let R be the total number of rounds. For each block that is an output of a round, and input of another round there are $R^2 \leq h^2$ possibilities to choose the origin and destination of the block. Because there are $\ell/2$ blocks transferred, h^ℓ is an upper bound on the total number of possible macroscopic structures of how blocks travel between rounds.

Further, the values of $u_l^{(k)}$ can be chosen which yields at most B^ℓ possibilities. Every traced element that is transferred by a block can terminate at the destination, i.e., it is not copied further. Hence, there are $2^{\ell B/w}$ choices of terminating elements. Each of the s_i non-terminal incoming elements of round i can appear up to M/B times in the outgoing blocks, namely once per outgoing block. Hence, there are $\binom{s_i M/B}{t_i}$ possibilities to choose the t_i outgoing elements of round i, for some $t_i \geq s_i$. Since we have $\sum_{1\leq i\leq R} t_i \leq B\ell/w$, the total number of possibilities for this is bounded by $\binom{M\ell/w}{B\ell/w}$.

Finally, we have to specify the subset of possible multiplications that are actually performed. To this end, let W_i be the number of partial results output by round i. Together with the number of vector variables U_i loaded in round i, there are $\sum_{i\leq \ell\frac{B}{M}} U_i W_i$ possible multiplications with matrix entries during the program. Additionally, we have the conditions $U_i \leq M$, $W_i \leq M$, and $\sum_{i\leq \ell\frac{B}{M}}(U_i + W_i) \leq \frac{\ell B}{w}$. The term $\sum_{i\leq \ell\frac{B}{M}} U_i W_i$ is hence maximised for $U_i = W_i = M$, for some indices $i \in \mathcal{I}$, $\mathcal{I} \subseteq [\ell\frac{B}{M}]$ with $|\mathcal{I}| = \frac{\ell B}{2Mw}$, and the size of the set of possible multiplications is at most $\frac{\ell B}{2Mw} \cdot M^2 = \frac{\ell MB}{2w}$. From this, we select a subset of size h, yielding no more than $\binom{\ell MB/(2w)}{h}$ possibilities.

Calculations With the above discussion, we get

$$\binom{N_x N_y}{h} \leq h^\ell \cdot B^\ell \cdot 2^{\ell B/w} \cdot \binom{\ell M/w}{\ell B/w} \cdot \binom{\frac{\ell MB}{2w}}{h}.$$

W.l.o.g. we assume $N_x \geq N_y$. Define $k = h/N_x$, i.e., the average number of entries per column. Rearranging terms yields $\ell \geq h\frac{\log\frac{N_y}{k} - \log\frac{e\ell MB}{wh}}{\log h + \log B + \frac{B}{w}(1+\log\frac{eM}{B})}$. In the following, we can assume $h \geq M \geq B$, and thus $\frac{\ell}{h} \geq \frac{\log\left(\frac{N_y}{k}\cdot\frac{wh}{e\ell MB}\right)}{2\log h + \frac{B}{w}(\log\frac{6M}{B})}$. Otherwise, if $h \leq M$, the task is trivial and a scanning bound of $\Omega\left(\frac{h}{B}\right)$ for reading $\mathbf{A}$ suffices. Applying Lemma B.2 in [2] ($x = \ell/h$, $t = 2\log h + \frac{B}{w}(\log\frac{6M}{B})$, $s = \frac{wN_y}{ekMB}$), and estimating $t \geq \log N_x + 3\frac{B}{w}$, we get $\frac{2\ell}{h} \geq \frac{\log\left(\frac{wN_y}{ekMB}\cdot\left(\frac{3B}{w}+\log N_x\right)\right)}{2\log h + \frac{B}{w}(\log\frac{6M}{B})}$. Now, it remains to distinguish according to the leading term in the denominator.

Case 1 ($2\log h \leq \frac{B}{w}(\log\frac{6M}{B})$): $\frac{\ell}{h} \geq \frac{\log\frac{N_y}{kM}}{4\frac{B}{w}(\log\frac{6M}{B})}$ Using $M > 4B$ yields $\ell \geq \frac{hw}{B}\frac{\log\frac{N_y}{kM}}{4\cdot\frac{3}{2}\log\frac{M}{B}} = \frac{hw}{6B}\log_{\frac{M}{B}}\frac{N_y}{kM}$ which matches the sorting based bound.

Case 2 ($2\log h > \frac{B}{w}(\log \frac{6M}{B})$): $\frac{2\ell}{h} \geq \frac{\log\left(\frac{wN_y}{BekM}\log N_x\right)}{4\log h}$ matches the table based bound. Recall that the table based upper bound requires $N_x \geq N_y^2$.

However, for $N_x \leq N_y^2$, a linear lower bound is obtained as follows. Assuming $k \leq \sqrt[6]{N_y}$, $B \geq 16w$, and $N_x \geq 2^{30}$ implies $\frac{N_y}{BkeM} \geq \sqrt[8]{N_x}$ (for details see [3]), such that $\frac{2\ell}{h} \geq \frac{\log \frac{wN_y}{BekM}}{2\log h} \geq \frac{\log \sqrt[8]{N_y}}{4(1+1/2)\log N_y} \geq \frac{1}{48}$.

Otherwise, if $N_y \leq N_x < 2^{30}$, also h is bounded from above, and thus, the task is trivially possible in $\mathcal{O}(1)$. Together with the presented algorithms and the lower bound from Lemma 1, this discussion yields Theorem 3. Furthermore, together with the lower bounds for column major layout, the proof of Theorem 2 is completed.

6 Acknowledgement

Thanks to Dan Roche and Clement Pernet for asking the question about multiplying many vector pairs. Many thanks to an anonymous referee for suggestions on an earlier draft of this article.

References

1. A. Aggarwal and J. S. Vitter. The input/output complexity of sorting and related problems. *Communications of the ACM*, 31(9):1116–1127, 1988.
2. M. A. Bender, G. S. Brodal, R. Fagerberg, R. Jacob, and E. Vicari. Optimal sparse matrix dense vector multiplication in the I/O-model. In *Proceedings of SPAA '07*, pages 61–70, New York, NY, USA, 2007. ACM.
3. G. Greiner and R. Jacob. Evaluating non-square sparse bilinear forms on multiple vector pairs in the I/O-model. Technical report, Technische Universität München, June 2010.
4. G. Greiner and R. Jacob. The I/O complexity of sparse matrix dense matrix multiplication. In *Proceedings of LATIN'10*, volume 6034 of *Lecture Notes in Computer Science*, pages 143–156. Springer, 2010.
5. Hong, Jia-Wei and H. T. Kung. I/O complexity: The red-blue pebble game. In *Proceedings of STOC '81*, pages 326–333, New York, NY, USA, 1981. ACM.
6. R. Jacob and M. Schnupp. Experimental performance of I/O-optimal sparse matrix dense vector multiplication algorithms within main memory. Technical report, Technische Universität München, June 2010.
7. T. Lieber. Combinatorial approaches to optimizing sparse matrix dense vector multiplication in the I/O-model. Master's thesis, Informatik Technische Universität München, 2009.
8. F. F. Roos, R. Jacob, J. Grossmann, B. Fischer, J. M. Buhmann, W. Gruissem, S. Baginsky, and P. Widmayer. Pepsplice: cache-efficient search algorithms for comprehensive identification of tandem mass spectra. *Bioinformatics*, 23(22):3016–3023, 2007.
9. R. W. Vuduc. *Automatic Performance Tuning of Sparse Matrix Kernels*. PhD thesis, University of California, Berkeley, Fall 2003.