# General Methods for Sparsity Structure Description and Cost Estimation

*Grace Dinh, James Demmel, Zhiru Zhang*

Abstract

Sparse tensor operations (especially matrix multiplications and tensor contractions in general) can be used to represent many problems in diverse fields such as genomics, machine learning, network analysis, and electronic design automation. Casting a domain-specific problem as a sparse linear algebra operation allows domain experts to leverage existing optimized software libraries and hardware. However the cost (in terms of flops, data movement/accesses, or memory footprint) of a sparse operation can vary significantly depending on the sparsity structure of its input, making the development of *general* high-performance tools for sparse linear algebra challenging. Estimating and bounding these costs is important for many applications: coming up with a performance objective for optimizing a software or hardware implementation, developing a notion of "peak performance" to compare a benchmark against, determining how much space to allocate for scratch or for the output of an operation, load balancing, and many more.

Cost estimation is straightforward for dense linear algebra, as the exact set of arithmetic instructions is always the same and known ahead of time. However, in the sparse case, this is not possible unless the *exact* sparsity structure (i.e. the locations of every nonzero) of the inputs is known. As a result, previous cost modeling approaches, e.g. [12], tend to either require that users provide a specific input matrix (precluding their use to develop and evaluate *general* tools) or provide results restricted to specific sparsity structures (e.g. uniformly distributed sparsity, block sparsity, band matrices). For input matrices that do not neatly fit into one of these predetermined categories, however, significant case-by-case work is required on the part of users to develop statistical models that both describe their matrices and provide good cost estimates and bounds.

This abstract sketches out an approach to *generalize* and *automate* the construction of such sparsity models, and to build cost *estimates* and *bounds* that take them into account, building on techniques from database and information theory. In Section 1, we describe a way to describe sparsity structure using *matrix statistics*. We then describe how to use these statistics to bound and estimate costs in Section 2, and to optimize storage formats for sparse matrices in Section 3.

## 1 Characterizing Sparsity Structure

Our goal in this section is to describe a framework for *matrix statistics* - quantities describing the sparsity structure of a matrix that are (a) well-defined for any sparse matrix, regardless of structure, and (b) can be effectively used to predict the performance of a tensor operation. The most well known matrix statistic is the number of nonzeros (nnz) of a matrix. However, nnz alone is clearly insufficient for the cost estimation problem. Consider, for instance, the square matrix $A$ whose first column is nonzero and whose other columns are all zero. Despite having the same input nnzs, $A^TA$ and $AA^T$ differ drastically in output memory footprint (and therefore data movement). As a result, accurate performance modeling requires additional statistics to describing a matrix's sparsity structure in more detail.

One way to do so is to count the number of nonzeros in each row and column, which we refer to as the *row* and *column counts*, as in [6]. These statistics require significantly more space to store than the nnzs (as they are vectors with length equal to the number of columns and rows, respectively, of a matrix), but provide significantly more information: taking the dot product the column count of $A$ and row count of $B$ gives the exact number of flops required to compute $A \times B$ . Furthermore, row and column counts can be *summarized* using by taking $L^p$ norms for a few small $p$. These norms provide a compact, *easily generalizable* way to represent how “skewed” the sparsity structure of a matrix is (e.g. how heavy-tailed the distribution of connections is in a social network graph) which can also be used to derive bounds on the cost of a matrix multiplication, as we will briefly discuss in Section 2 (see [1] for a discussion of these bounds from a database point of view).

However, row and column counts alone are insufficient to describe many forms of commonly seen sparsity patterns, e.g. band and block-sparse matrices. To represent these patterns, we will extend the notion of *indices* to *functions* of the rows and column index. For a concrete example, consider a tridiagonal matrix $A$ indexed by $(i, j)$. All of the locations of its nonzeros take on only three distinct values of $i - j$; as a result, “number of distinct values of $i - j$” is a useful statistic that allows us to encapsulate tridiagonal matrices (and band matrices in general).

To formalize and generalize, let us view a sparse matrix $A$ indexed by $(i, j)$ as a *set* consisting of the location of its nonzeros: $\{(i, j) : A_{ij} \neq 0\}$. Let $e_{1,\ldots} : \mathbb{Z}^2 \to \mathbb{Z}$ be some functions (such as $i - j$ in the above example), which we will refer to as *index maps*. Define the following two operations:

**Definition 1.** The *projection* operation $\pi_{e_k}$ projects its input onto dimension $e_k$ - that is, $\pi_{e_k}(A)$ has a nonzero at location $l$ if there exists some nonzero value in $A$ at location $i, j$ such that $e_k(i, j) = l$.

**Definition 2.** The *selection* operation $\sigma_{e_k=\eta}$ returns the subset of nonzero locations $(i, j)$ in $A$ such that $e_k(i, j) = \eta$. When no value for $\eta$ is given, the selection operator $\sigma_{e_k}$ will be used to represent the list $(\sigma_{e_k=\eta} : \eta \in \mathrm{Im}(e_k))$.

Let $\circ$ represent function composition. If the output of $g$ is a vector, let $f \vec{\circ} g$ denote the *vector* obtained by applying $f$ to every element of the output of $g$. Then many natural matrix statistics can be represented by choosing appropriate index maps $e_k$:

- Row counts: first select each row ($i$), then count the number of nonzeros in each ($|\cdot|$): $|\cdot| \vec{\circ} \sigma_i$. Column counts are identical, with $j$ replacing $i$.
- Band width of a band matrix: first project onto $i - j$, then count: $|\cdot| \circ \pi_{i-j}$
- Number of nonzero blocks in a block-sparse matrix with block size $b$: project onto blocks $(\lfloor i/b \rfloor, \lfloor j/b \rfloor)$, then count: $|\cdot| \circ \pi_{\lfloor i/b \rfloor, \lfloor j/b \rfloor}$
- Fine-grained structured sparsity (maximum number of nonzeros in each block): for each block (i.e. selection operator on $\lfloor i/b \rfloor, \lfloor j/b \rfloor$), count the number of nonzeros, then take the max: $\max \circ |\cdot| \vec{\circ} \sigma_{\lfloor x_1/b \rfloor, \lfloor x_2/b \rfloor}$

Furthermore, appropriately chosen index maps can be used to characterize matrices with sparsity structures that do not align with “standard” patterns. For example, the Tuma1[1] matrix could be decomposed into several components, each of which would have a very small value for $|\cdot| \circ \pi_{\alpha i - j}$ (for some constant $\alpha$). Preliminary experiments show that computer vision methods such as Hough transforms [7] as well as modern machine learning methods such as symbolic regression [9] can be used to extract descriptive index maps from many real-world matrices that can be used to derive useful bounds; we leave further experimentation to future work.

[1] https://sparse.tamu.edu/GHS_indef/tuma1

## 2 Bounds from Matrix Statistics

This section describes approaches to deriving cost bounds from matrix statistics derived in Section 1. While we focus on matrix multiplication here, our approach can generalize to most "nested loop" style programs acting on sparse data; we leave such generalization to future work. As in the previous section, we will view a sparse matrix as a set whose elements are its nonzero indices. Then a sparse matrix multiplication $A \times B$, where $A$ is indexed by $(i, j)$ and $B$ by $(j, k)$, can be viewed as the set of nontrivial arithmetic instructions - that is, $\{(i, j, k) : A_{ij} \neq 0, B_{jk} \neq 0\}$, which we denote $T$. Note that this matrix multiplication tensor can be viewed as the *database join* $A(i, j) \wedge B(j, k)$. Several cost functions immediately fall from this representation:

- The *number of flops* required to compute $A \times B$ is simply the cardinality of the matrix multiplication tensor $|A(i, j) \wedge B(j, k)|$.
- The *size of the output* is the size of the *projection* of the matrix multiply tensor onto the $i$, $k$ face $|\pi_{i,k} (A(i, j) \wedge B(j, k))|$.
- The *arithmetic intensity* of $A \times B$ on a system with fast memory $M$ can upper bounded by computing the maximum number of elements for any subset of of $T$ subject to the constraint that the projections of that subset onto the $(i, j)$ and $(j, k)$ dimensions are bounded by $M$. In previous work focusing on dense linear algebra [8, 5], this immediately provides a data movement lower bound of ($M$ × #total flops) / (max $T$-subset size); however, the number of flops may not be exactly known in the sparse setting, so we will focus on upper bounding the arithmetic intensity instead.

One approach we can take to bounding these quantities is to transform the indices of the nested loops in such a way that the resulting loop nest, when treated as a dense operation, produces useful bounds. For instance, suppose we wish to multiply two band matrices $A$ and $B$, which have band width $w_1$ and $w_2$ respectively:

$$\begin{aligned}&\text{for } i, j, k \in [0, N)^3\\ &\quad C(i, k) += A(i, j) \times B(j, k)\end{aligned}$$

As the two matrices are banded, we know that $|\cdot| \circ \pi_{i-j} = w_1$ and $|\cdot| \circ \pi_{k-j} = w_2$. As a result, if we let $e_1 = i - j$ and $e_2 = k - j$, we can rewrite this nested of loops as:

$$\begin{aligned}&\text{for } e_1 \in [0, w_1), e_2 \in [0, w_2), j \in [0, N)\\ &\quad C(e_1 + j, e_2 + j) += A(e_1 + j, j) \times B(j, e_2 + j)\end{aligned}$$

which provides an upper bound for flops of $w_1 w_2 N$. Furthermore, using Brascamp-Lieb inequalities [5, 10, 4] provides an arithmetic intensity upper bound (on a system with cache size $M$) of $\sqrt{M}/2$.

Unfortunately, this method is not easily generalized: we were able to transform indices $i$ and $k$ into new indices that could easily be bounded using the given matrix statistics because $A$ and $B$ shared band structure; this would not be possible if they were not. To address this problem, we adapt information-theoretic techniques previously used for database cardinality estimation [3, 2]. Specifically, given *any* probability distribution over set of arithmetic instructions $T$ in the sparse matrix multiplication, let $h$ denote the Shannon entropies of its marginal distributions $h$ (e.g. use $h(ij)$ to denote the entropy of the marginal distribution over $i$,$j$). Clearly, $h(ijk)$ is upper bounded by $\lg |T|$, the number of flops of the matrix multiplication. Furthermore, notice that for an instruction $(i, j, k)$ to be in $T$, $A_{ij}$ and $B_{jk}$ must both be nonzero; as a result, the entropies $h(ij)$ and $h(jk)$ are upper bounded by $\lg \mathrm{nnz}(A)$ and $\lg \mathrm{nnz}(B)$ respectively. These inequalities can be combined with those inherent to entropy (nonnegativity, submodularity, and subadditivity) to produce bounds on cost.

For example, it can be shown that for *any* distribution on $i, j, k$:

$$3h(ijk) \leq h(i,j) + h(j,k) + h(i,k) + h(j|i) + h(k|j) + h(i|k)$$

Letting the distribution be the uniform distribution over $T$ sets the left side of the above inequality to $\lg\left(\#\text{flops}^3\right)$, while $h(i,j)$, $h(j,i)$, and $h(i,k)$ are upper bounded by $\lg \mathrm{nnz} A$, $\lg \mathrm{nnz} B$, and $\lg \mathrm{nnz} C$ respectively. Furthermore, $h(j|i)$ is upper bounded by the log of the maximum number of nonzero elements in any row of $A$ (similarly for the remaining terms), giving an inequality that ties together computation cost, output size, and memory footprint. In this framework, all of the cost functions above can be described: number of flops and output size are immediately derivable from entropic inequalities, and arithmetic intensity can be found by adding constraints the entropies $h(ij)$ and $h(jk)$ are upper bounded by $\lg M$. In order to adapt matrix statistics using arbitrary index maps (e.g. $e_1 = i - j$), we can add additional constraints: specifically that $h(e_1|ij) = h(i|je_1) = h(j|ie_1) = 0$. This allows for the *automated* construction of new lower bounds for, say, the cost of multiplying of a band matrix by a block-sparse one, based on statistics such as the number of dense blocks sharing an index with a given band.

## 3 Matrix Format Optimizations

We also wish to find efficient ways to *store* sparse matrices. Consider, for example, a band matrix with a small band width. Standard sparse matrix formats, such as CSR, would require significantly more storage for metadata (row pointers and column indices) than a similar format indexed by $i-j$ and $j$ [11]. Furthermore, the *order* in which the indices are stored can significantly affect size and performance too - just as $(i, j)$ (CSR) and $(j, i)$ (CSC) are significantly different formats, so would $(i - j, j)$ and $(j, i - j)$.

The choice of data structures and layouts directly impacts computing performance. For instance, to efficiently use the Gustavson algorithm, the operand tensors should ideally be stored in row-major formats. We will describe how entropic bounds (specifically, the chain bound) can suggest optimal *orderings* and *data structures* for sparse matrix storage formats.

However, performance is often heavily affected by the underlying hardware architecture. For parallel processing systems like GPUs, maintaining workload balance often outweighs achieving a high compression ratio in terms of format selection. As a result, formats with zero padding, such as ELLPACK, are commonly preferred over those that store only non-zero elements. Blocking formats, while introducing additional memory access and metadata overhead on architectures with a unified memory model, are well-suited for many-core architectures with banked memory. Work is ongoing to extend our cost models to account for hardware-specific performance factors.

# References

[1] Mahmoud Abo Khamis, Vasileios Nakos, Dan Olteanu, and Dan Suciu. Join Size Bounds using lp-Norms on Degree Sequences. *Proc. ACM Manag. Data*, 2(2):96:1–96:24, May 2024.

[2] Mahmoud Abo Khamis, Hung Q. Ngo, and Dan Suciu. Computing Join Queries with Functional Dependencies. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '16, pages 327–342, New York, NY, USA, June 2016. Association for Computing Machinery.

[3] Mahmoud Abo Khamis, Hung Q. Ngo, and Dan Suciu. What Do Shannon-type Inequalities, Submodular Width, and Disjunctive Datalog Have to Do with One Another? In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '17, pages 429–444, New York, NY, USA, May 2017. Association for Computing Machinery.

[4] Anthony Chen, James Demmel, Grace Dinh, Mason Haberle, and Olga Holtz. Communication bounds for convolutional neural networks. In *Proceedings of the Platform for Advanced Scientific Computing Conference*, PASC '22, New York, NY, USA, 2022. Association for Computing Machinery.

[5] Michael Christ, James Demmel, Nicholas Knight, Thomas Scanlon, and Katherine A. Yelick. Communication Lower Bounds and Optimal Algorithms for Programs that Reference Arrays - Part 1:. Technical report, Defense Technical Information Center, Fort Belvoir, VA, May 2013.

[6] Kyle Deeds, Willow Ahrens, Magda Balazinska, and Dan Suciu. Galley: Modern Query Optimization for Sparse Tensor Programs, August 2024.

[7] Paul VC Hough. Method and means for recognizing complex patterns, December 1962.

[8] Dror Irony, Sivan Toledo, and Alexander Tiskin. Communication lower bounds for distributed-memory matrix multiplication. *Journal of Parallel and Distributed Computing*, 64(9):1017–1026, 2004.

[9] William La Cava, Bogdan Burlacu, Marco Virgolin, Michael Kommenda, Patryk Orzechowski, Fabrício Olivetti de França, Ying Jin, and Jason H. Moore. Contemporary Symbolic Regression Methods and their Relative Performance. *Advances in Neural Information Processing Systems*, 2021(DB1):1–16, December 2021.

[10] Auguste Olivry, Julien Langou, Louis-Noël Pouchet, P. Sadayappan, and Fabrice Rastello. Automated derivation of parametric data movement lower bounds for affine programs. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '20, pages 808–822, New York, NY, USA, 2020. Association for Computing Machinery.

[11] Yousef Saad. *Iterative Methods for Sparse Linear Systems.* Society for Industrial and Applied Mathematics, second edition, January 2003.

[12] Yannan Nellie Wu, Po-An Tsai, Angshuman Parashar, Vivienne Sze, and Joel S. Emer. Sparseloop: An Analytical Approach To Sparse Tensor Accelerator Modeling. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1377–1395, Chicago, IL, USA, October 2022. IEEE.