Incremental Sparse Tensor Format for Maximizing Efficiency in Tensor-Vector Multiplications

Xiaohe Niu, Georg Meyer[‡], Dimosthenis Pasadakis[‡], Albert-Jan Yzelman[§], Olaf Schenk[‡]
[‡]Università della Svizzera italiana, Lugano, Switzerland
[§]Huawei Zurich Research Center, Zurich, Switzerland
xiaohe@niu.pm, georg.meyer@usi.ch, dimosthenis.pasadakis@usi.ch,
albertjan.yzelman@huawei.com, olaf.schenk@usi.ch

***Abstract*—Current state-of-the-art sparse tensor formats achieve memory-efficient representations but often require extensive indexing or precomputation, thus limiting their flexibility and efficiency. We propose *Bit-IF* (Incremental Sparse Fibers with Bit Encoding), a format that only records index increments encoded by a compact bit array. This mode-independent approach allows for an arbitrary index traversal during tensor-vector multiplications (TVM), enabling the usage of space-filling curves. *Bit-IF*'s design characteristics significantly reduce memory overhead, improve data locality, and eliminate the need for multiple tensor copies or mode-specific preprocessing before performing a TVM. Our analysis and initial comparative studies show that *Bit-IF* reduces memory consumption and TVM computation time compared to *COO*-based approaches. The applicability of this method could be extended to other tensor operations, such as tensor-matrix and Khatri-Rao products.**

***Index Terms*—Sparse tensor format; Incremental indexes; Tensor algebra; High performance computing**

I. Introduction

Dense tensor representations of high-dimensional datasets significantly hinder computational analysis due to the exponential growth of storage requirements, which scale as $O(n^D)$ with respect to the dataset's dimensionality D, and the resulting computational complexities. Current state-of-the-art sparse tensor formats, such as Coordinate (*COO*), Compressed Sparse Fibers (*CSF*) [1] or Hierarchical Coordinate (*HiCOO*) [2], reduce storage requirements and computational complexities, but encounter limitations. For tensor operations like TVMs, mode dependencies, complex indexing, and high conversion complexities necessitate extensive precomputation and the storage of duplicate tensors with different mode orderings, thereby reducing efficiency and scalability.

This work presents ***Bit-IF*** (Incremental Sparse Fibers with Bit Encoding), a novel storage scheme based on incremental compression concepts previously explored for sparse matrices and tensors. Although *Bit-IF* is constructed by compressing a *COO* tensor, the approach significantly differs from the standard *COO* format, which stores indices for each non-zero element. Instead, *Bit-IF* only encodes index increments — changes in index values between consecutive nonzero elements — and a dedicated bit array. Each nonzero element is associated with a bit sequence corresponding to the number of modes. Every bit sequence signals which modes have changed, allowing us to reconstruct complete indices incrementally without explicitly storing them for each non-zero entry.

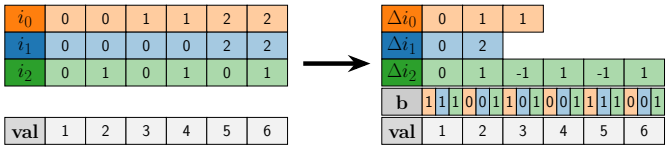


Fig. 1: Conversion of a 3rd-order tensor from *COO* format to the proposed *Bit-IF* encoding. i_k signify mode-wise indices; Δi_k only record index changes.

II. Bit-IF for TVM

Bit-IF extends incremental compression techniques previously explored for sparse matrices and is built around three central guidelines:

1) **Minimal prior knowledge:** No extensive preprocessing or reordering of the input tensor indices should be needed to perform TVM along arbitrary modes.
2) **Mode independence:** *Bit-IF* avoids dependence on a specific mode ordering, allowing flexible access and rearrangement of modes.
3) **Arbitrary index traversal:** Supports index access patterns that improve data locality for specific tensor operations beyond mode independence.

The key components of our approach are depicted in Fig. 1. Tensor indices are stored via an incremental indexing strategy along each mode Δi_m, reducing storage overhead by capturing only changes between consecutive indices. Then, a compact bit vector b encodes the presence and direction of increments for each non-zero entry, allowing efficient traversal and storage.

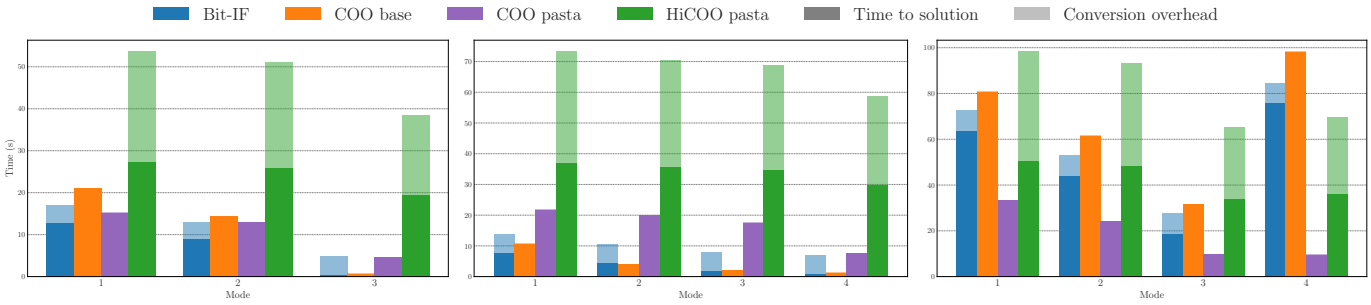
Definition 2.1 (kth-mode Tensor–Vector Multiplication): Let $\mathcal{A} \in \mathbb{R}^{n_0 \times n_1 \times \cdots \times n_{d-1}}, \mathbf{v} \in \mathbb{R}^{n_k}$. Then, the k-mode tensor–vector product

$$\mathcal{B} = \mathcal{A} \times_k \mathbf{v} \in \mathbb{R}^{n_0 \times \cdots \times n_{k-1} \times 1 \times n_{k+1} \times \cdots \times n_{d-1}} \quad (1)$$

is defined componentwise as

$$\mathcal{B}_{i_0,\ldots,i_{k-1},i_{k+1},\ldots,i_{d-1}} = \sum_{i_k=0}^{n_k-1} \mathcal{A}_{i_0,\ldots,i_k,\ldots,i_{d-1}} \, \mathbf{v}_{i_k}. \quad (2)$$

Spatial access patterns such as Z- or Hilbert curves, enable arbitrarily tensor traversal for TVM, bypassing the computationally expensive reordering of tensor entries before computation. This flexibility reduces preprocessing overhead, eliminates the need for multiple instances of the same tensor, and ensures efficient access patterns across different tensor modes. The adoption of a blocking strategy allows the algorithm to be parallelized.



(a) Nell-2 ($nnz = 7.69 \times 10^7$) **(b)** Synthetic tensor ($nnz = 10^8$) **(c)** Delicious-4d ($nnz = 1.4 \times 10^8$)

Fig. 2: Time-to-solution for a TVM on three tensor datasets under different storage formats. Each bar includes the pre-computation overhead required by an ad-hoc TVM. The extra cost of converting the baseline *COO* structure to the target format is shown separately. Labels marked *base* use our own *COO*-TVM implementation, whereas *pasta* employs the TVMs from the PASTA library [3]. All measurements were taken single-threaded on a Intel Xeon Platinum 8360Y.

III. Comparison of Storage Requirements

We compare the storage requirements for a tensor with $nnz_{\mathcal{X}}$ nonzeros in d modes:

COO	$nnz_{\mathcal{X}}\left(w_{val} + d\, w_{int}\right)$
Bit-IF	$nnz_{\mathcal{X}}\left(w_{val} + d\, w_{bit} + \sum_{j=0}^{d-1} q_j\, w_{inc,s}\right)$
HiCOO	$nnz_{\mathcal{X}}\left(w_{val} + \alpha_b\, w_{long} + \alpha_b\, d\, w_{int} + d\, w_{byte}\right)$

w_x	storage size of datatype x
q_j	ratio of index changes in mode j
α_b	number of blocks per non-zero in HiCOO

COO maintains integer indices for every mode and every non-zero entry. *HiCOO* exploits the hierarchical structure of sparse tensors by storing blocks of nonzero entries, thus enabling the use of smaller data types for block relative coordinate indices. *HiCOO*, derived from *COO*, may incur memory overhead for tensors with predominantly single-mode index changes due to limited compression for sparsely populated fibers. *Bit-IF* reduces storage requirements by encoding index changes using bits and increments, additionally allowing the use of smaller data types for increments. For the tensor results illustrated represented in Fig. 3, we reduce the required storage by an average of 25% compared to the *COO* representation.

IV. Performance Results

In Fig. 2 we compare total time-to-solution of *Bit-IF* (blue) and *HiCOO* (green) for TVM on synthetic and real-world tensors [4], and report conversion time from an initial *COO* representation. We also include *PASTA-COO* [3] (purple) and a simple *COO* baseline (orange), to indicate *Bit-IF*'s current, unblocked implementation state and the room for further low-level optimizations. The synthetic tensor (Fig. 2b) — a fourth-order tensor with identical dimensions and nnz-rates — tests mode-dependent behavior. Across the representative tensors, *Bit-IF*'s compute time is competitive or faster than *HiCOO* in close to all cases, with some exceptions reported in Fig. 2c. Relative to *HiCOO*, we observe a 4.20× ($\overline{S}_{geom}$) / 4.65× ($\overline{S}_{arithm}$) Speedup in the computation phase and a 5.08× ($\overline{S}_{geom}$) / 5.15× ($\overline{S}_{arithm}$) reduction in the conversion time from *COO*.

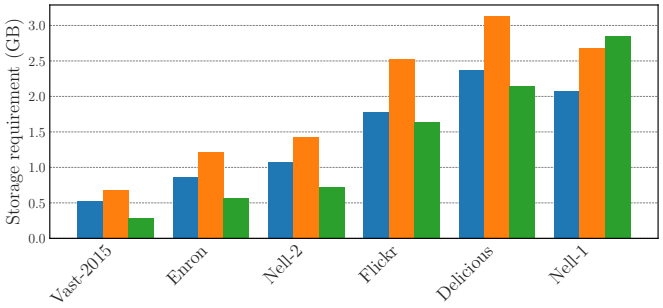


Fig. 3: Storage requirements of *COO*, *HiCOO*, and *Bit-IF* using 32-bit integers for indices and increments. Benchmark tensor datasets are drawn from the FROSTT collection [4]. *Bit-IF* reduces storage by an average of 25% compared to *COO* and avoids multiple tensor instances for different traversal orders; further gains over *HiCOO* are possible with smaller increment types.

Acknowledgment

This work was financially supported by the joint DFG - 470857344 and SNSF - 204817 project, and by the Huawei Zurich Research Center. We acknowledge the scientific support and HPC resources from the Erlangen National High Performance Computing Center (NHR@FAU) of the Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU) under the NHR project 80227.

References

[1] S. Smith, N. Ravindran, N. D. Sidiropoulos, and G. Karypis, "Splatt: Efficient and parallel sparse tensor-matrix multiplication," in 2015 IEEE International Parallel and Distributed Processing Symposium, 2015, pp. 61–70.

[2] J. Li, J. Sun, and R. Vuduc, "HiCOO: Hierarchical storage of sparse tensors," in Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '18), 2018.

[3] J. Li, Y. Ma, X. Wu, A. Li, and K. Barker, "PASTA: A parallel sparse tensor algorithm benchmark suite," CCF Transactions on High Performance Computing, vol. 1, p. 111–130, 2019.

[4] S. Smith, W. Choi, J. Li, R. Vuduc, J. Park, X. Liu, Y. Xing, and G. Karypis, "FROSTT: The formidable repository of open sparse tensors and tools," 2017, repository and tools available online.