Università della Svizzera italiana | Institute of Computing CI

Bit-IF: An Incremental Sparse Tensor Format for Maximizing Efficiency in Tensor-Vector Multiplications

Xiaohe Niu, Georg Meyer, Dimosthenis Pasadakis, Albert-Jan Yzelman[a], Olaf Schenk

[a]Computing Systems Lab, Huawei Zurich Research Center

- X. Niu, "Rethinking Sparse Tensor Storage: Incremental Formats as a Path Towards Maximizing Tensor-Vector Multiplication Efficiency," M.S. thesis, ETH, Zurich, Switzerland, 2023.
- 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 1, 2019.
- J. Li, J. Sun and R. Vuduc, "HiCOO: Hierarchical Storage of Sparse Tensors," SC18: International Conference for High Performance Computing, Networking, Storage and Analysis, Dallas, TX, USA, 2018.
- S. Smith, W. Choi, J. Li, R. Vuduc, J. Park, X. Liu, Xing and G. Karypis, "FROSTT: The Formidable Repository of Open Sparse Tensors and Tools," 2017.

Bit-If Compression

Incremental Sparse Fibers with Bit Encoding (*Bit-IF*) was designed to address the limitations of existing sparse tensor formats. It is based on incremental compression concepts previously explored for sparse matrices.
Bit-IF's three central design guidelines are:

- **Minimal prior knowledge:** No extensive preprocessing or reordering of the input tensor indices should be needed to perform TVM along arbitrary modes.
- **Mode independence:** With increments and bit encoding, *Bit-IF* avoids dependence on a specific mode ordering, enabling flexible access and rearrangement of modes.
- **Arbitrary index traversal:** This concept allows for index access patterns that improve data locality for specific tensor operations and performance optimizations besides mode independence.

Compression from COO

i_0	0	0	1	1	2	2
i_1	0	0	0	0	2	2
i_2	0	1	0	1	0	1
val	1	2	3	4	5	6

→

Δi_0	0	1	1			
Δi_1	0	2				
Δi_2	0	1	-1	1	-1	1
b	1 1 1	0 0 1	1 0 1	0 0 1	1 1 1	0 0 1
val	1	2	3	4	5	6

Key Components

- **Incremental Indexing:** Tensor indices are represented as increments along each mode, reducing storage overhead by capturing only the changes between consecutive indices.
- **Bit Encoding:** A compact bit vector encodes the presence and direction of increments for each nonzero entry, enabling efficient traversal and storage.

Traversal Curve Based Approach to TVM

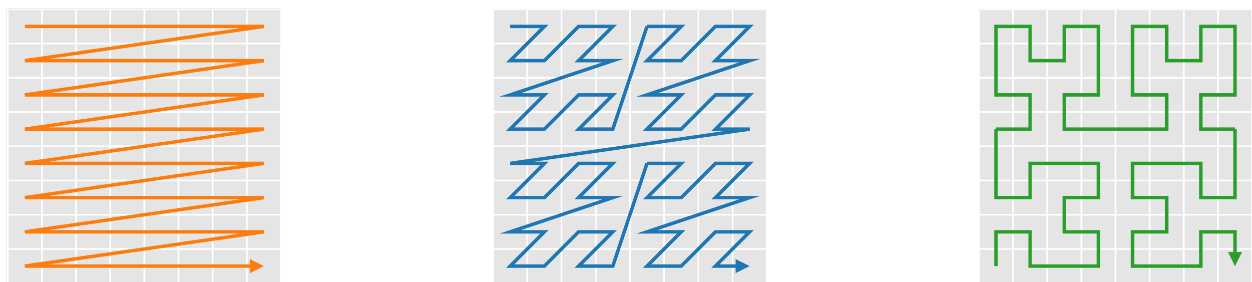
kth mode Tensor-Vector Multiplication (TVM)

$$\mathcal{A} \times_k \mathbf{v} = \mathcal{B}, \quad \text{where} \quad \mathcal{A} \in \mathbb{R}^{n_0 \times n_1 \times \cdots \times n_{d-1}}, \quad \mathbf{v} \in \mathbb{R}^{n_k}, \quad \mathcal{B} \in \mathbb{R}^{n_0 \times \cdots \times n_{k-1} \times 1 \times n_{k+1} \cdots \times n_{d-1}}:$$

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

Traversal curves enable arbitrary 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.

Traversal Curves



Lexicographical | Z-Curve | Hilbert

Blocking vs Non-blocking

Blocking synergizes with traversal curves to enhance TVM efficiency by fitting smaller tensor segments into the cache, optimizing data locality. It reduces data movement and computational overhead, thus improving performance, especially for large tensors. However, when selecting an optimal block size, special attention must be given to the often non-deterministic sparsity patterns, as it must balance computational overhead, cache efficiency, and workload distribution.

Algorithms

Algorithm 1 From *COO* to *Bit-IF*

```
Input: Input indices in COO format
Output: ΔA, b_A
Initialize ΔA_j for j = 0, ..., d − 1
Store indices of COO i_{0,j} in ΔA_j
Set b_j = 1, j = 0, ..., d − 1
for r = 0, 1, ..., nnz_A do
    Compute increments Δi = i_r − i_{r−1}
    for each mode j = 0, ..., d − 1 do
        if Δi_j ≠ 0 then
            Add Δi_j to ΔA_j
            Add 1 to b_A
        else
            Add 0 to b_A
        end if
    end for
end for
```

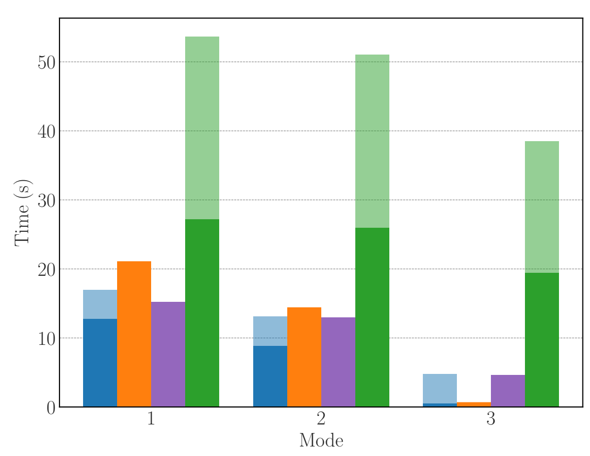
$d \in \mathbb{N}$	Order of the tensor
n_j	Size of dimension/mode j
$\Delta\mathcal{X}$	Increment arrays of $\mathcal{X}$
Δi	Current increments
$\text{val}_{\mathcal{X}}$	Non-zero values of $\mathcal{X}$
$b_{\mathcal{X}}$	Bit encoding array of $\mathcal{X}$
$\mathcal{V}_{\mathcal{B}}$	Map storing the values of $\mathcal{B}$
$nnz_{\mathcal{X}} \in \mathbb{N}$	Number of non-zero entries

Algorithm 2 TVM for Arbitrary Traversal Orders

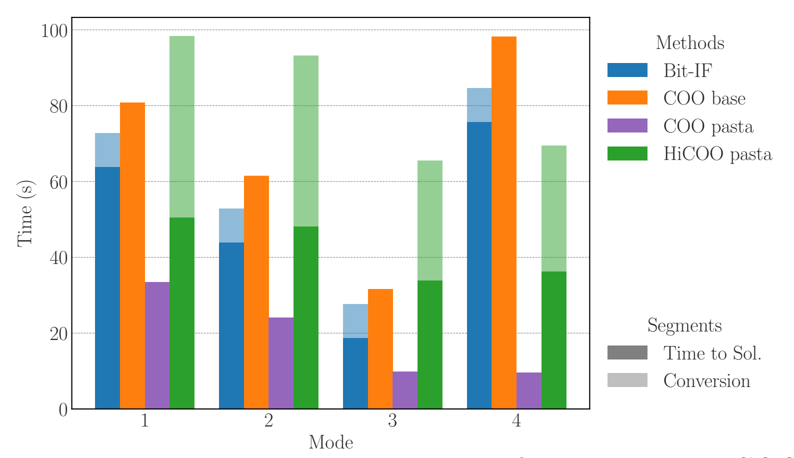
```
Input: ΔA, b_A, val_A, v
Output: ΔB, b_B, val_B
Initialize temporary value T
for each set b in b_A do
    if only b_{i−k} = 1 then
        i_{k+1} += Δi_{k−1}
        Update T += v(i_{k−1}) · val_A(i_{k−1})
    else if b_j = 1, j ≠ k − 1 or b_{k−1} = 0 then
        for remaining modes j do
            if b_j = 1 then
                Get Δi_j from ΔA_j
                i_j += Δi_j
            else
                j += 1
            end if
        end for
        if any b_{j≠k−1} = 1 then
            Get i_key of i_0, ..., i_{d−2}
            Update V_B(i_key) with i_0, ..., i_{d−2} if not contained
            val_B(i_key) ← T
        end if
    end if
end for
Compute ΔB, b_B with U_B according to Alg. 1
```

Performance Comparison

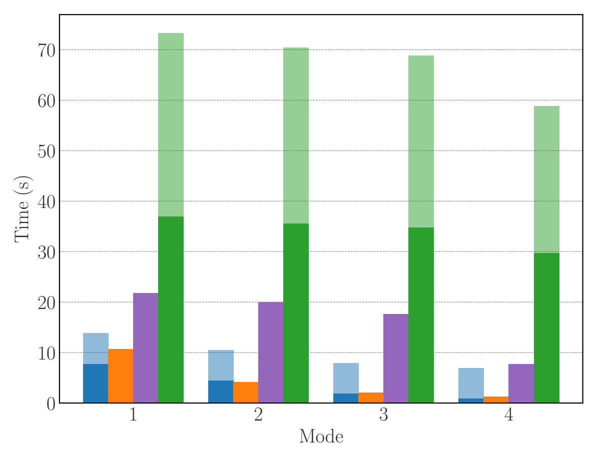
nell-2 ($nnz = 7.69 \cdot 10^7$)



delicious-4d ($nnz = 1.4 \cdot 10^8$)



synthetic tensor ($nnz = 10^8$)



We compare the **time to solution** of *Bit-IF* and *HiCOO* for TVM using different real-world and synthetic tensors. We also explore the **conversion time** needed from an initial *COO* presentation. We present *PASTA*'s and our *COO* implementations as baselines, illustrating *Bit-IF*'s current unblocked implementation state and potential for further improvements. The synthetic tensor — a fourth-order tensor with identical dimensions and *nnz*-rates — tests mode-dependent behavior.

***Bit-IF* vs. *HiCOO* Speedup ($\bar{S}_{geom}$):**

4.20× TVM Computation | **5.08×** Conversion

Comparative Storage Requirement Study

Theoretical Analysis

$$\text{COO:} \quad nnz_{\mathcal{X}} \cdot (w_{val} + d \cdot w_{int})$$

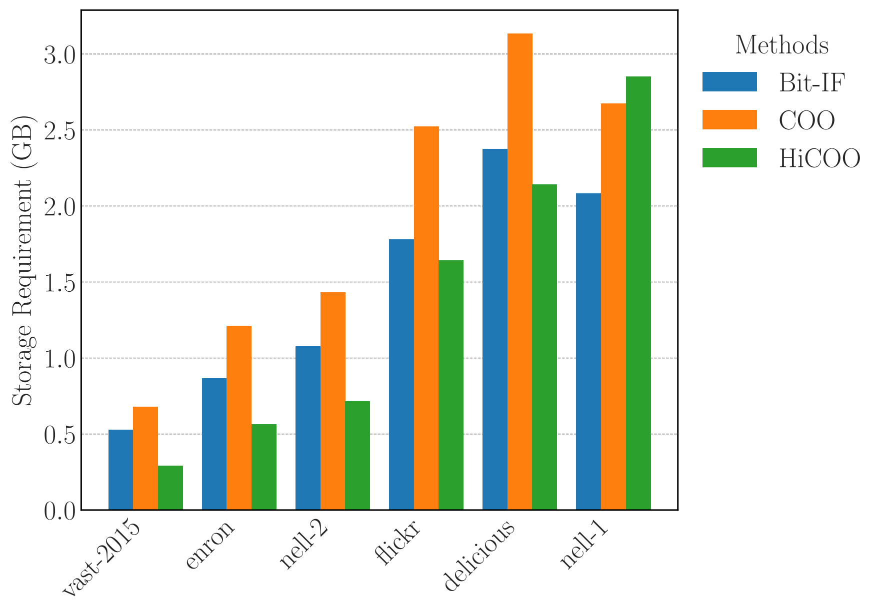
$$\text{Bit-IF:} \quad nnz_{\mathcal{X}} \cdot \left(w_{val} + d \cdot w_{bit} + \textstyle\sum_{j=0}^{d-1} q_j \cdot w_{inc,s}\right)$$

$$\text{HiCOO:} \quad nnz_{\mathcal{X}} \cdot (w_{val} + \alpha_b \cdot w_{long} + \alpha_b \cdot d \cdot w_{int} + d \cdot w_{byte})$$

w_{val}	Storage size for a tensor value.
$w_{inc,s}$	Storage size for short integer increments.
w_x	Storage size for a certain datatype.
q_j	Ratio of index changes in mode j.
α_b	number of blocks per nonzero entry in HiCOO.

COO maintains integer indices for every mode and every nonzero 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, allowing the use of smaller data types for the increments. Like *HiCOO*, *Bit-IF* can use a two-level block-based scheme [1].

Comparison



For these measurements, 32-bit integers are used for *COO* indices and *Bit-IF* increments. While *Bit-IF* offers **significant storage savings over *COO* (∼27%)**, further improvements over *HiCOO* are achievable with blocking and smaller data types for increments. Unlike *COO* and *HiCOO*, *Bit-IF* eliminates the need for multiple tensor instances for different traversal orders.

Future Work

- Investigate impact of smaller data types for *Bit-IF* increments paired with blocking.
- Further optimize the TVM traversal curve based approach for single thread execution.
- Parallelize the TVM traversal curve based approach (Single- / Multi-Node).
- Prepare a comparative study of strong and weak scaling for the TVM.
- Implement further Tensor Operations based on the *Bit-IF* format.

Thesis and Code

github.com/xniuuu/SparseTensorComputations