Hashed Coordinate Storage of Sparse Tensors

Robert Lowe MeiLi Charles Amritpreet Signh

Southeast Missouri State University and University of Tennessee

Abstract

Tensors, or n-way arrays, are becoming increasingly important in many fields. In recent applications, tensors are extremely sparse and have such a high degree that dense storage becomes intractable. To address this, several sparse storage formats have been proposed,with most formats being some variation of coordinate-based storage, or COO. These formats are created to optimize the Matricized-Tensor-Times-Khatri-Rao product (MTTKRP) operation. The current state of the art, HiCOO, provides efficient MTTKRP operations, however it is expensive to build and update. HiCOO requires an $O(n \lg n)$ operation to provide z-order sorting of the indices before building its block structure. Furthermore, updating a HiCOO tensor would require at least a restructuring of blocks. This paper presents a new format called Hashed Coordinate (HaCOO). HaCOO uses a low-collision hash function to associate tensor indices with values. HaCOO improves upon the state-of-the-art by providing $O(1)$ amortized time complexity for arbitrary value insertion, update, and retrieval with little memory overhead.

Introduction and Background

- Dense storage of sparse tensors is intractable.
- Sparse tensor formats optimize MTTKRP at the expense of general operations.
- HaCOO provides efficient MTTKRP and general operations.

HaCOO Storage Format

The HaCOO storage format consists of a separate-chaining hash table with a load factor of 0.70. Each hash entry contains the following fields:

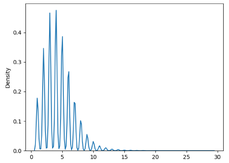
morton_index - The Morton encoded index of the entry.

value - The value of the entry at the index.

To allow for fast modulus operations via bit-masking, the size of the hash table is always a power of two. The initial size is arbitrarily set to 128. If its optimal load factor is exceeded, it doubles in size and all existing entries are rehashed into the new table. If the number of non-zero entries, nnz, is known in advance the hash table's size can be computed as in Equation 1.

$$size = 2^{\lceil \lg nnz/0.7 \rceil} \tag{1}$$

Hashing Method



Probe Depth Probability Density

The hash function, shown in Algorithm 2, has two main parts. The first step is to compute the Morton encoding [1] of the index. (The index being an n-length array of integers.) The result of this operation, assuming an integer of arbitrary length, is a unique integer representing the index's position on a z-order curve. This, however, is insufficient for hashing as the range of this integer far exceeds the capacity of the hash table. To arrive at the final hash, series of simple arithmetic and bit shift operations are performed. This step is inspired by Jenkin's one-at-a-time hash function [2]. The result of these operations is a hash value which is fairly evenly distributed over the size of the hash table. The final step of the algorithm is to extract k, the hashed index of this entry. This is performed by simple bitwise arithmetic with the mask.

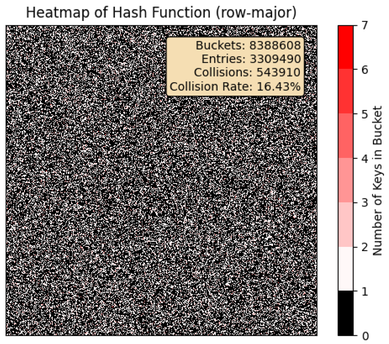
Algorithm 1 - Three shift values are computed such that each of the bits in the portion of the hash which will form the key have a roughly 0.25 probability of changing. This leads to a high degree of separation in the hash table, mitigating collisions.

Algorithm 2 - This determines four values: sx, sy, sz, and $mask$. These are driven by the size of the table, also known as the number of buckets ($nbuckets$). Because these values do not change unless the size of the hash table is changed, they are computed separately in Algorithm 1.

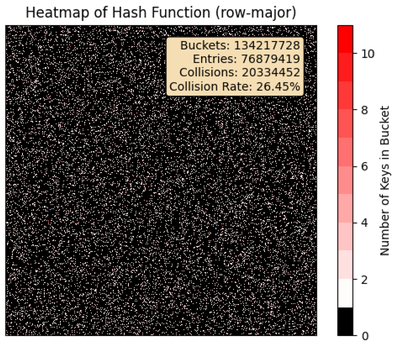
The hashing function used by HaCOO results in a well-distributed hash table as shown in the figure above and the heatmaps in the center. In all experimental cases, the mean, median, and mode of the probe depth are the same. Thus HaCOO has an insertion and search time of $O(1)$.

Tensor	Modes	Non-Zeros	Collision Rate	Mean Probe Depth	Median Probe Depth	Mode Probe Depth	Max Probe Depth
uber	4	3,309,490	16.43%	1.20	1	1	7
nell-2	3	76,879,419	26.45%	1.36	1	1	11
enron	4	54,202,099	37.53%	1.60	1	1	37
nips	4	3,101,609	77.31%	4.41	4	4	29

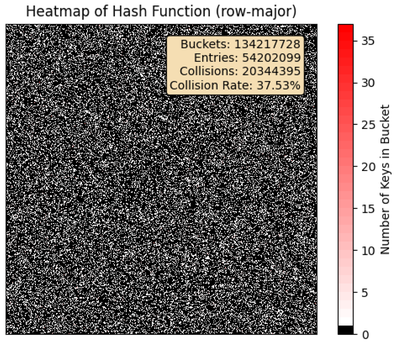
Hash Storage Results for Several FROSTT Tensors [3]



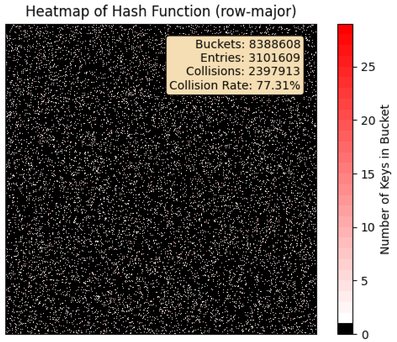
(a) Uber



(b) Nell-2



(c) Enron



(d) Nips

Algorithm 1: Hash Values

Input: $nbuckets$
Output: sx, sy, sz, $mask$

$bits \leftarrow \lg nbuckets$
$sx \leftarrow$ ceil($bits$ / 8) $-$ 1
$sy \leftarrow 4 * sx$- 1
$sz \leftarrow$ ceil($bits$ / 2) $-$ 1
$mask \leftarrow \sim nbuckets$

Algorithm 2: Hash Algorithm

Input: $index$
Output: m, k

$m \leftarrow$ morton($index$)
$hash \leftarrow hash + hash \ll sx$
$hash \leftarrow hash \oplus hash \gg sy$
$hash \leftarrow hash + hash \ll sz$
$k \leftarrow hash \ \& \ mask$

Advantages of HaCOO

The current state of the art, HiCOO, provides efficient MTTKRP operations, however it is expensive to build and update. HiCOO requires an $O(n \lg n)$ operation to provide z-order sorting of the indices before building its block structure. Furthermore, updating a HiCOO tensor would require at least a restructuring of blocks. If the underlying data structure in which HiCOO is implemented does not support arbitrary insertions, then HiCOO would require a repeat of its sort operation. HaCOO does not suffer from this problem, as its insertions and updates take place in constant time. HaCOO provides lower overhead in the simple loading and decomposition of tensors while allowing for online tensor operations where updates would be desirable.

Results and Analysis

Experiments were run by simulating the storage of a sparse tensor and plotting the distribution of entries within the hash table. Heat-map style plots reveal that entries are distributed among buckets relatively evenly throughout the entire hash table. This is further supported by the average probe depth results in the Table at the left, where the average probe depth is typically just over 1. In all cases the median and mode depth are the same as the mean probe depth.
Additionally, the nips tensor in Figure 1(b) serves as a near-worst case scenario. Even with a collision rate of 77.31% and maximum probe depth of 29, it still maintains an average probe depth of 4.41. These results show an expected near-constant $O(1)$ access time for the retrieval, storage, or deletion of an entry.
Because the traversal of the tensor is not sensitive to ordering, any sparse MTTKRP algorithm (such as the ones found in CSF and HiCOO), may be used. Also, because all access and write operations are performed in constant time, all other tensor operations can be performed efficiently using this format. That is to say, this data structure does not dominate the runtime complexity of any tensor operation.

References

[1] Guy M Morton.
A computer oriented geodetic data base and a new technique in file sequencing.
1966.

[2] Bob Jenkins, 2013.

[3] Shaden Smith, Jee W. Choi, Jiajia Li, Richard Vuduc, Jongsoo Park, Xing Liu, and George Karypis.
FROSTT: The formidable repository of open sparse tensors and tools, 2017.

[4] Jitesh Shetty and Jafar Adibi.
The enron email dataset database schema and brief statistical report.
Information sciences institute technical report, University of Southern California, 4, 2004.

[5] Shaden Smith and George Karypis.
Tensor-matrix products with a compressed sparse tensor.
In *Proceedings of the 5th Workshop on Irregular Applications: Architectures and Algorithms*, pages 1–7, 2015.

[6] Baugtian Liu, Chengyao Wen, Anand D Sarwate, and Maryam Mehri Dehnavi.
A unified optimization approach for sparse tensor operations on gpus.
In *2017 IEEE international conference on cluster computing (CLUSTER)*, pages 47–57. IEEE, 2017.

[7] J. Li, J. Sun, and R. Vuduc.
Hicoo: Hierarchical storage of sparse tensors.
In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 238–252, 2018.

[8] Tamara G Kolda and Brett W Bader.
Tensor decompositions and applications.
SIAM review, 51(3):455–500, 2009.

Links and Contact Information

https://gymmyp1.github.io/sptensor-python/

- GitHub: https://github.com/gymmyp1/sptensor-python
- **Dr. Robert Lowe**, Southeast Missouri State University, Email: rlowe@semo.edu
- **MeiLi Charles**, University of Tennessee, Email: jcharl12@vols.utk.edu
- **Amritpreet Singh**, Southeast Missouri State University, Email: asingh2s@semo.edu

Southeast Missouri State University · 1873®

The University of Tennessee Knoxville